
aiogram Documentation

Реліз 3.7.0

aiogram Team

трав. 10, 2024

1	Особливості	3
1.1	Приклад використання	4
1.2	Usage without dispatcher	5
2	Зміст	7
2.1	Встановлення	7
2.1.1	З PyPI	7
2.1.2	З репозиторію Arch Linux	7
2.1.3	З PyPI	7
2.1.4	З GitHub	7
2.2	FAQ по переходу з версії 2.x на 3.0	8
2.2.1	Dispatcher	8
2.2.2	Фільтрація подій	9
2.2.3	Bot API	9
2.2.4	Проміжне ПО (Middlewares)	9
2.2.5	Розмітка клавіатури	10
2.2.6	Callbacks data	10
2.2.7	Скінчений автомат	10
2.2.8	Надсилення файлів	10
2.2.9	Webhook	10
2.2.10	Сервер Telegram API	11
2.2.11	Telegram objects transformation (to dict, to json, from json)	11
2.3	Бот API	12
2.3.1	Bot	12
2.3.2	Client session	14
2.3.3	Types	19
2.3.4	Methods	314
2.3.5	Enums	489
2.3.6	Як завантажити файл?	503
2.3.7	Як відвантажити файл?	505
2.4	Обробка подій	507
2.4.1	Маршрутизатор	508
2.4.2	Диспетчер	513
2.4.3	Dependency injection	516
2.4.4	Фільтрування подій	518
2.4.5	Long-polling	530

2.4.6	Webhook	532
2.4.7	Кінцевий автомат (FSM)	541
2.4.8	Проміжні програми	570
2.4.9	Errors	573
2.4.10	Маркери	575
2.4.11	Обробники на основі класів	577
2.5	Утиліти	582
2.5.1	Конструктор клавіатури	582
2.5.2	Переклад	586
2.5.3	Відправник дій у чаті	591
2.5.4	Веб Застосунок (WebApp)	593
2.5.5	Callback answer	598
2.5.6	Formatting	601
2.5.7	Media group builder	608
2.5.8	Deep Linking	612
2.6	Changelog	613
2.6.1	3.7.0 [UNRELEASED DRAFT] (2024-05-10)	613
2.6.2	3.6.0 (2024-05-06)	613
2.6.3	3.5.0 (2024-04-23)	613
2.6.4	3.4.1 (2024-02-17)	614
2.6.5	3.4.0 (2024-02-16)	614
2.6.6	3.3.0 (2023-12-31)	615
2.6.7	3.2.0 (2023-11-24)	616
2.6.8	3.1.1 (2023-09-25)	617
2.6.9	3.1.0 (2023-09-22)	617
2.6.10	3.0.0 (2023-09-01)	617
2.6.11	3.0.0rc2 (2023-08-18)	617
2.6.12	3.0.0rc1 (2023-08-06)	618
2.6.13	3.0.0b9 (2023-07-30)	619
2.6.14	3.0.0b8 (2023-07-17)	620
2.6.15	3.0.0b7 (2023-02-18)	622
2.6.16	3.0.0b6 (2022-11-18)	624
2.6.17	3.0.0b5 (2022-10-02)	625
2.6.18	3.0.0b4 (2022-08-14)	626
2.6.19	3.0.0b3 (2022-04-19)	627
2.6.20	3.0.0b2 (2022-02-19)	628
2.6.21	3.0.0b1 (2021-12-12)	628
2.6.22	3.0.0a18 (2021-11-10)	629
2.6.23	3.0.0a17 (2021-09-24)	630
2.6.24	3.0.0a16 (2021-09-22)	630
2.6.25	3.0.0a15 (2021-09-10)	631
2.6.26	3.0.0a14 (2021-08-17)	631
2.6.27	2.14.3 (2021-07-21)	632
2.6.28	2.14.2 (2021-07-26)	632
2.6.29	2.14 (2021-07-27)	632
2.6.30	2.13 (2021-04-28)	632
2.6.31	2.12.1 (2021-03-22)	633
2.6.32	2.12 (2021-03-14)	633
2.6.33	2.11.2 (2021-11-10)	634
2.6.34	2.11.1 (2021-11-10)	634
2.6.35	2.11 (2021-11-08)	634
2.6.36	2.10.1 (2021-09-14)	634
2.6.37	2.10 (2021-09-13)	634
2.6.38	2.9.2 (2021-06-13)	635

2.6.39	2.9 (2021-06-08)	635
2.6.40	2.8 (2021-04-26)	636
2.6.41	2.7 (2021-04-07)	636
2.6.42	2.6.1 (2021-01-25)	636
2.6.43	2.6 (2021-01-23)	636
2.6.44	2.5.3 (2021-01-05)	636
2.6.45	2.5.2 (2021-01-01)	637
2.6.46	2.5.1 (2021-01-01)	637
2.6.47	2.5 (2021-01-01)	637
2.6.48	2.4 (2021-10-29)	637
2.6.49	2.3 (2021-08-16)	638
2.6.50	2.2 (2021-06-09)	638
2.6.51	2.1 (2021-04-18)	638
2.6.52	2.0.1 (2021-12-31)	639
2.6.53	2.0 (2021-10-28)	639
2.6.54	1.4 (2021-08-03)	639
2.6.55	1.3.3 (2021-07-16)	639
2.6.56	1.3.2 (2021-05-27)	639
2.6.57	1.3.1 (2018-05-27)	640
2.6.58	1.3 (2021-04-22)	640
2.6.59	1.2.3 (2018-04-14)	640
2.6.60	1.2.2 (2018-04-08)	640
2.6.61	1.2.1 (2018-03-25)	640
2.6.62	1.2 (2018-02-23)	640
2.6.63	1.1 (2018-01-27)	641
2.6.64	1.0.4 (2018-01-10)	641
2.6.65	1.0.3 (2018-01-07)	641
2.6.66	1.0.2 (2017-11-29)	641
2.6.67	1.0.1 (2017-11-21)	641
2.6.68	1.0 (2017-11-19)	641
2.6.69	0.4.1 (2017-08-03)	642
2.6.70	0.4 (2017-08-05)	642
2.6.71	0.3.4 (2017-08-04)	642
2.6.72	0.3.3 (2017-07-05)	642
2.6.73	0.3.2 (2017-07-04)	642
2.6.74	0.3.1 (2017-07-04)	642
2.6.75	0.2b1 (2017-06-00)	642
2.6.76	0.1 (2017-06-03)	642
2.7	Contributing	642
2.7.1	Developing	642
2.7.2	Star on GitHub	645
2.7.3	Guides	645
2.7.4	Take answers	645
2.7.5	Funding	645

Python Module Index	647
----------------------------	------------

Индекс	653
---------------	------------

aiogram це сучасний та повністю асинхронний фреймворк для розробки чат-ботів [Telegram Bot API](#) на Python 3.8 з використанням [asyncio](#) та [aiohttp](#).

Зробіть своїх ботів швидшими та потужнішими!

Документація

- [English](#)
- [Українською](#)

- Асинхронність ([asyncio docs](#), [PEP 492](#))
- Має анотації типів ([PEP 484](#)) та може використовуватись з [mypy](#)
- Працює з [PyPy](#)
- Supports [Telegram Bot API 7.3](#) and gets fast updates to the latest versions of the Bot API
- Код інтеграції з Bot API є автогенерованим що надає змогу дуже легко оновлювати фреймворк до останніх версій АПІ
- Має роутери подій (Blueprints)
- Має вбудований кінцевий автомат
- Uses powerful [magic filters](#)
- Підтримує мідлвари (для вхідних подій від АПІ та для вихідних запитів до АПІ)
- Підтримує можливість відповіді у вебхук
- Має вбудовану інтеграцію для використання інтернаціоналізації та локалізації GNU Gettext (або Fluent)

Попередження: Наполегливо рекомендується навчитись працювати з `asyncio` перед тим, як починати використовувати цей фреймворк. [asyncio](#)

Якщо є якісь додаткові запитання, ласкаво просимо до онлайн-спільнот:

- [@aiogram](#)
- [@aiogramua](#)
- [@aiogram_uz](#)
- [@aiogram_kz](#)
- [@aiogram_ru](#)

- @aiogram_fa
- @aiogram_it
- @aiogram_br

1.1 Приклад використання

```
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'
    ↪ 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:
```

(continues on next page)

(continued from previous page)

```

        # Send a copy of the received message
        await message.send_copy(chat_id=message.chat.id)
    except TypeError:
        # But not all the types is supported to be copied so need to handle it
        await message.answer("Nice try!")

async def main() -> None:
    # Initialize Bot instance with 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.INFO, 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(

```

(continues on next page)

(continued from previous page)

```
        token=token,
        default=DefaultBotProperties(
            parse_mode=ParseMode.HTML,
        ),
    ) as bot:
        await bot.send_message(chat_id=chat_id, text=message)

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

2.1 Встановлення

2.1.1 З PyPI

```
pip install -U aiogram
```

2.1.2 З репозиторію Arch Linux

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

Бета-версія (3.x)

2.1.3 З PyPI

```
pip install -U aiogram
```

2.1.4 З GitHub

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

2.2 FAQ по переходу з версії 2.x на 3.0

Небезпека: Цей посібник все ще в розробці.

Ця версія містить численні суттєві зміни та архітектурні покращення. Вона допомагає зменшити кількість глобальних змінних у вашому коді, надає корисні механізми для модуляризації вашого коду та дозволяє створювати спільні модулі за допомогою пакетів на PyPI. Крім того, серед інших покращень, він робить проміжне програмне забезпечення (мідлварі) та фільтри більш контрольованими.

На цій сторінці ви можете прочитати про зміни, внесені в останню стабільну версію 2.x.

Примітка: Ця сторінка більше нагадує детальний список змін, ніж посібник з міграції, але вона буде оновлюватися в майбутньому.

Не соромтеся зробити свій внесок у цю сторінку, якщо ви знайшли щось, про що тут не згадано.

2.2.1 Dispatcher

- Клас `Dispatcher` більше не приймає екземпляр `Bot` у своєму ініціалізаторі. Замість цього екземпляр `Bot` слід передавати диспетчеру тільки для запуску поліну або обробки подій з вебхуків. Такий підхід також дозволяє використовувати декілька екземплярів бота одночасно («мульти-бот»).
- Клас `Dispatcher` тепер можна розширити ще одним об'єктом на кшталт диспетчера з назвою `Router` (*Детальніше* »).
- За допомогою роутерів ви можете легко модулювати свій код і потенційно перевикористовувати ці модулі між проектами.
- Видалено суфікс `_handler` з усіх декораторів обробників подій та методів реєстрації. (*Детальніше* »)
- The `Executor` has been entirely removed; you can now use the `Dispatcher` directly to start poll the API or handle webhooks from it.
- Метод дроселювання (`Throttling`) повністю вилучено; тепер ви можете використовувати проміжне програмне забезпечення (`middleware`) для керування контекстом виконання та реалізовувати будь-який механізм дроселювання за вашим бажанням.
- Вилучено глобальні контекстні змінні з типів `API`, об'єктів `Bot` та `Dispatcher`. Відтепер, якщо ви хочете отримати доступ до поточного екземпляру бота в обробниках або фільтрах, ви повинні приймати аргумент `bot: Bot` і використовувати його замість `Bot.get_current()`. У проміжному програмному забезпеченні (`middleware`) доступ до нього можна отримати через `data["bot"]`.
- Щоб пропустити очікувані оновлення, тепер вам слід викликати метод `aiogram.methods.delete_webhook.DeleteWebhook` безпосередньо, а не передавати `skip_updates=True` до методу запуску поліну.

2.2.2 Фільтрація подій

- Фільтри за ключовими словами більше не можна використовувати; використовуйте фільтри явно. (*Детальніше »*)
- У зв'язку з вилученням keyword фільтрів, всі раніше ввімкнені за замовчуванням фільтри (такі як state і content_type) тепер вимкнено. Якщо ви бажаєте їх використовувати, ви повинні вказати їх явно. Наприклад, замість `@dp.message_handler(content_types=ContentType.PHOTO)` слід використовувати `@router.message(F.photo)`.
- Most common filters have been replaced with the «magic filter.» (*Read more »*)
- За замовчуванням обробник повідомлень тепер отримує будь-який тип вмісту. Якщо вам потрібен певний тип, просто додайте відповідні фільтри (Magic або будь-який інший).
- Фільтр стану більше не вмикається за замовчуванням. Це означає, що якщо ви використовували `state="*"` у v2, вам не слід передавати фільтр стану у v3. І навпаки, якщо стан не було вказано у v2, вам потрібно буде вказати його у v3.
- Додано можливість реєстрації глобальних фільтрів для кожного роутера, що допомагає зменшити повторення коду і полегшує контроль призначення кожного роутера.

2.2.3 Bot API

- Всі методи API тепер є класами з валідацією, реалізованими через *pydantic* <<https://docs.pydantic.dev/>>. Ці виклики API також доступні як методи в класі Bot.
- Додано більше попередньо визначених enums та переміщено їх до підпакету *aiogram.enums*. Наприклад, enum типу чату тепер має вигляд `aiogram.enums.ChatType` замість `aiogram.types.chat.ChatType`.
- Клієнтська сесія HTTP була відокремлена в контейнер, який можна повторно використовувати для різних екземплярів бота в додатку.
- Виключення API більше не класифікуються за конкретними повідомленнями, оскільки Telegram не має задокументованих кодів помилок. Проте всі помилки класифікуються за кодами статусу HTTP, і для кожного методу з певним кодом може бути пов'язаний лише один тип помилки. Тому в більшості випадків слід перевіряти лише тип помилки (за кодом статусу), не перевіряючи повідомлення про помилку.

2.2.4 Проміжне ПО (Middlewares)

- Проміжне програмне забезпечення тепер може керувати контекстом виконання, наприклад, за допомогою менеджерів контексту. (*Детальніше »*)
- Всі контекстні дані тепер наскрізно використовуються між проміжним програмним забезпеченням, фільтрами та обробниками. Наприклад, тепер ви можете легко передати деякі дані в контекст у проміжному програмному забезпеченні і отримати їх у шарі фільтрів так само, як і в обробниках через аргументи ключових слів.
- Додано механізм з назвою **flags**, який допомагає налаштовувати поведінку обробника у поєднанні з проміжним програмним забезпеченням. (*Детальніше про »*)

2.2.5 Розмітка клавіатури

- Тепер `aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup` та `aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup` більше не мають методів для розширення, натомість вам слід використовувати будівники розмітки `aiogram.utils.keyboard.ReplyKeyboardBuilder` та `aiogram.utils.keyboard.InlineKeyboardBuilder` відповідно (*Детальніше »*)

2.2.6 Callbacks data

- Фабрику даних зворотного виклику тепер строго типізовано за допомогою моделей `pydantic`. (*Детальніше »*)

2.2.7 Скінченний автомат

- Фільтри станів більше не будуть автоматично додаватися до всіх обробників; вам потрібно буде вказати стан, якщо ви хочете його використати.
- Додано можливість змінювати стратегію FSM. Наприклад, якщо ви хочете контролювати стан для кожного користувача на основі топиків чату, а не користувача в чаті, ви можете вказати це в Диспетчері.
- Тепер `aiogram.fsm.state.State` та `aiogram.fsm.state.StateGroup` не мають допоміжних методів, таких як `.set()`, `.next()` тощо.
- Замість цього вам слід встановлювати стани, передаючи їх безпосередньо до `aiogram.fsm.context.FSMContext` (*Детальніше »*)
- Проксі стану є застарілим; вам слід оновити дані стану, викликавши `state.set_data(...)` та `state.get_data()` відповідно.

2.2.8 Надсилання файлів

- Відтепер перед відправкою файлів слід обертати їх в об'єкт `InputFile` замість того, щоб передавати об'єкт вводу-виводу безпосередньо до методу API. (*Детальніше »*)

2.2.9 Webhook

- Спрощено налаштування веб-застосунку `aiohttp`.
- 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 Сервер Telegram API

- Параметр `server` було перенесено з екземпляра `Bot` до `api` в `BaseSession`.
- Константа `aiogram.bot.api.TELEGRAM_PRODUCTION` була переміщена на `aiogram.client.telegram.PRODUCTION`.

2.2.11 Telegram objects transformation (to dict, to json, from json)

- Methods `TelegramObject.to_object()`, `TelegramObject.to_json()` and `TelegramObject.to_python()` have been removed due to the use of `pydantic` models.
- `TelegramObject.to_object()` should be replaced by `TelegramObject.model_validate()` ([Read more](#))
- `TelegramObject.as_json()` should be replaced by `TelegramObject.model_dump_json()` ([Read more](#))
- `TelegramObject.to_python()` should be replaced by `TelegramObject.model_dump()` ([Read more](#))

Here are some usage examples:

- Creating an object from a dictionary representation of an object

```
# Version 2.x
message_dict = {"id": 42, ...}
message_obj = Message.to_object(message_dict)
print(message_obj)
# id=42 name='n' ...
print(type(message_obj))
# <class 'aiogram.types.message.Message'>

# Version 3.x
message_dict = {"id": 42, ...}
message_obj = Message.model_validate(message_dict)
print(message_obj)
# id=42 name='n' ...
print(type(message_obj))
# <class 'aiogram.types.message.Message'>
```

- Creating a json representation of an object

```
async def handler(message: Message) -> None:
    # Version 2.x
    message_json = message.as_json()
    print(message_json)
    # {"id": 42, ...}
    print(type(message_json))
    # <class 'str'>

    # Version 3.x
    message_json = message.model_dump_json()
    print(message_json)
    # {"id": 42, ...}
    print(type(message_json))
    # <class 'str'>
```

- Creating a dictionary representation of an object

```
async def handler(message: Message) -> None:
    # Version 2.x
    message_dict = message.to_python()
    print(message_dict)
    # {"id": 42, ...}
    print(type(message_dict))
    # <class 'dict'>

    # Version 3.x
    message_dict = message.model_dump()
    print(message_dict)
    # {"id": 42, ...}
    print(type(message_dict))
    # <class 'dict'>
```

2.3 Бот API

aiogram наразі повністю підтримує [Telegram Bot API](#)

Усі методи та типи повністю автоматично згенеровані з документації Telegram Bot API за допомогою парсера з генератором коду.

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.

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

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

Bases: object

```
__init__(token: str, session: BaseSession / None = None, parse_mode: str / None = None,
          disable_web_page_preview: bool / None = None, protect_content: bool / None = None,
          default: DefaultBotProperties / None = None) -> None
```

Bot class

Параметри

- `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.

Викликає

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

property `token: str`

property `id: int`

Get bot ID from token

Повертає

`context(auto_close: bool = True) → AsyncIterator[Bot]`

Generate bot context

Параметри

`auto_close` – close session on exit

Повертає

`async me() → User`

Cached alias for `getMe` method

Повертає

`async download_file(file_path: str, destination: BinaryIO | Path | str | None = None, timeout: int = 30, chunk_size: int = 65536, seek: bool = True) → BinaryIO | None`

Download file by `file_path` to destination.

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

Параметри

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

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

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

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

Параметри

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

2.3.2 Client session

Client sessions is used for interacting with API server.

Use Custom API server

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

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

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

Base config for API Endpoints

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

Generate URL for API methods

Параметри

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

Повертає

URL

`base: str`

Base URL

`file: str`

Files URL

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

Generate URL for downloading files

Параметри

- `token` – Bot token

- path – file path

Повертає

URL

```
classmethod from_base(base: str, **kwargs: Any) → TelegramAPIServer
```

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

Параметри

base – Base URL

Повертаєinstance of *TelegramAPIServer*

```
is_local: bool = False
```

Mark this server is in *local mode*.

```
wrap_local_file: FilePathWrapper = <aiogram.client.telegram.BareFilePathWrapper
object>
```

Callback to wrap files path in local mode

Base

Abstract session for all client sessions

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

This is base class for all HTTP sessions in aiogram.

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

```
check_response(bot: Bot, method: TelegramMethod[TelegramType], status_code: int, content: str)
→ Response[TelegramType]
```

Check response status

```
abstract async close() → None
```

Close client session

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

Make request to Telegram Bot API

Параметри

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

Повертає

Викликає

TelegramApiError –

`prepare_value(value: Any, bot: Bot, files: Dict[str, Any], _dumps_json: bool = True) → Any`

Prepare value before send

`abstract async stream_content(url: str, headers: Dict[str, Any] | None = None, timeout: int = 30, chunk_size: int = 65536, raise_for_status: bool = True) → AsyncGenerator[bytes, None]`

Stream reader

aiohhttp

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

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

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

Usage example

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

session = AiohttpSession()
bot = Bot('42:token', session=session)
```

Proxy requests in AiohttpSession

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

Binding session to bot:

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

session = AiohttpSession(proxy="protocol://host:port/")
bot = Bot(token="bot token", session=session)
```

Примітка: 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 `aihttp.BasicAuth` containing login and password.

Consider examples:

```
from aihttp import BasicAuth
from aiogram.client.session.aihttp 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")
```

Примітка: 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 `aihttp-socks` supports proxy chains, you're able to use them in aiogram

Example of chain proxies:

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

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

Client session middlewares

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

Some useful cases that is:

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

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

How to register client session middleware?

Register using register method

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

Register using decorator

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

Example

Class based session middleware

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

Примітка: 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

```

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

```

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

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

file_id: str

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

file_unique_id: str

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

width: int

Video width as defined by sender

height: int

Video height as defined by sender

duration: int

Duration of the video in seconds as defined by sender

```
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: PhotoSize | None
```

Optional. Animation thumbnail as defined by sender

```
file_name: str | None
```

Optional. Original animation filename as defined by sender

```
mime_type: str | None
```

Optional. MIME type of the file as defined by sender

```
file_size: int | None
```

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

Audio

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

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

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

```
file_id: str
```

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

```
file_unique_id: str
```

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

```
duration: int
```

Duration of the audio in seconds as defined by sender

```
performer: str | None
```

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

```
title: str | None
```

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

```
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_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

BackgroundFill

`class aiogram.types.background_fill.BackgroundFill(**extra_data: Any)`

This object describes the way a background is filled based on the selected colors. Currently, it can be one of

- `aiogram.types.background_fill_solid.BackgroundFillSolid`
- `aiogram.types.background_fill_gradient.BackgroundFillGradient`
- `aiogram.types.background_fill_freeform_gradient.BackgroundFillFreeformGradient`

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

`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.

BackgroundFillFreeformGradient

```
class aiogram.types.background_fill_freeform_gradient.BackgroundFillFreeformGradient(*,
    type:
        Literal['freeform_gradient']
    colors:
        List[int],
    **extra_data:
        Any)
```

The background is a freeform gradient that rotates after every message in the chat.

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

`type: Literal['freeform_gradient']`

Type of the background fill, always „freeform_gradient“

`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.

`colors: List[int]`

A list of the 3 or 4 base colors that are used to generate the freeform gradient in the RGB24 format

BackgroundFillGradient

```
class aiogram.types.background_fill_gradient.BackgroundFillGradient(*, type: Literal['gradient']
                                                                    = 'gradient', top_color:
                                                                    int, bottom_color: int,
                                                                    rotation_angle: int,
                                                                    **extra_data: Any)
```

The background is a gradient fill.

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

`type: Literal['gradient']`

Type of the background fill, always „gradient“

`top_color: int`

Top color of the gradient in the RGB24 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.

`bottom_color: int`

Bottom color of the gradient in the RGB24 format

`rotation_angle: int`

Clockwise rotation angle of the background fill in degrees; 0-359

BackgroundFillSolid

```
class aiogram.types.background_fill_solid.BackgroundFillSolid(*, type: Literal['solid'] = 'solid',
                                                             color: int, **extra_data: Any)
```

The background is filled using the selected color.

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

`type: Literal['solid']`

Type of the background fill, always „solid“

`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.

`color: int`

The color of the background fill in the RGB24 format

BackgroundType

```
class aiogram.types.background_type.BackgroundType(**extra_data: Any)
```

This object describes the type of a background. Currently, it can be one of

- `aiogram.types.background_type_fill.BackgroundTypeFill`
- `aiogram.types.background_type_wallpaper.BackgroundTypeWallpaper`
- `aiogram.types.background_type_pattern.BackgroundTypePattern`
- `aiogram.types.background_type_chat_theme.BackgroundTypeChatTheme`

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

```
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.

BackgroundTypeChatTheme

```
class aiogram.types.background_type_chat_theme.BackgroundTypeChatTheme(*, type:
    Literal['chat_theme']
    = 'chat_theme',
    theme_name: str,
    **extra_data: Any)
```

The background is taken directly from a built-in chat theme.

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

```
type: Literal['chat_theme']
```

Type of the background, always „chat_theme“

```
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.

```
theme_name: str
```

Name of the chat theme, which is usually an emoji

BackgroundTypeFill

```
class aiogram.types.background_type_fill.BackgroundTypeFill(*, type: Literal['fill'] = 'fill', fill:
    BackgroundFillSolid /
    BackgroundFillGradient /
    BackgroundFillFreeformGradient,
    dark_theme_dimming: int,
    **extra_data: Any)
```

The background is automatically filled based on the selected colors.

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

`type: Literal['fill']`
Type of the background, always „fill“

`fill: BackgroundFillSolid | BackgroundFillGradient | BackgroundFillFreeformGradient`
The background fill

`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.

`dark_theme_dimming: int`
Dimming of the background in dark themes, as a percentage; 0-100

BackgroundTypePattern

```
class aiogram.types.background_type_pattern.BackgroundTypePattern(*, type: Literal['pattern'] =  
    'pattern', document:  
    Document, fill:  
    BackgroundFillSolid /  
    BackgroundFillGradient /  
    BackgroundFi-  
    llFreeformGradient,  
    intensity: int, is_inverted:  
    bool / None = None,  
    is_moving: bool / None =  
    None, **extra_data: Any)
```

The background is a PNG or TGV (gzipped subset of SVG with MIME type „application/x-tgwallpattern“) pattern to be combined with the background fill chosen by the user.

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

`type: Literal['pattern']`
Type of the background, always „pattern“

`document: Document`
Document with the pattern

`fill: BackgroundFillSolid | BackgroundFillGradient | BackgroundFillFreeformGradient`
The background fill that is combined with the pattern

`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.

`intensity: int`
Intensity of the pattern when it is shown above the filled background; 0-100

`is_inverted: bool | None`
Optional. True, if the background fill must be applied only to the pattern itself. All other pixels are black in this case. For dark themes only

`is_moving: bool | None`
Optional. True, if the background moves slightly when the device is tilted

BackgroundTypeWallpaper

```
class aiogram.types.background_type_wallpaper.BackgroundTypeWallpaper(*, type:
    Literal['wallpaper'] =
    'wallpaper', document:
    Document,
    dark_theme_dimming:
    int, is_blurred: bool |
    None = None,
    is_moving: bool | None
    = None, **extra_data:
    Any)
```

The background is a wallpaper in the JPEG format.

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

type: `Literal['wallpaper']`

Type of the background, always „wallpaper“

document: `Document`

Document with the wallpaper

dark_theme_dimming: `int`

Dimming of the background in dark themes, as a percentage; 0-100

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.

is_blurred: `bool | None`

Optional. `True`, if the wallpaper is downscaled to fit in a 450x450 square and then box-blurred with radius 12

is_moving: `bool | None`

Optional. `True`, if the background moves slightly when the device is tilted

Birthdate

```
class aiogram.types.birthdate.Birthdate(*, day: int, month: int, year: int | None = None,
    **extra_data: Any)
```

Describes the birthdate of a user.

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) → 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:
                                                                                                     Li-
                                                                                                     teral[
                                                                                                     =
                                                                                                     BotC
                                                                                                     **ext
                                                                                                     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) → None

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

BotCommandScopeAllGroupChats

```
class aiogram.types.bot_command_scope_all_group_chats.BotCommandScopeAllGroupChats(*, type:
                                                                                                     Li-
                                                                                                     teral[BotCommandScope
                                                                                                     =
                                                                                                     BotCommandScopeType.
                                                                                                     **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) → 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(*,
                                                                                       type: Li-
                                                                                       teral[BotCommandScopeType.ALL_PRIVATE_CHATS]
                                                                                       =
                                                                                       BotCommandScopeType.ALL_PRIVATE_CHATS,
                                                                                       **extra_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) → None`

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

BotCommandScopeChat

```
class aiogram.types.bot_command_scope_chat.BotCommandScopeChat(*, type: Li-
                                                                 teral[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:
                                                                                             Li-
                                                                                             teral[BotCom
                                                                                             =
                                                                                             BotCommanda
                                                                                             chat_id:
                                                                                             int
                                                                                             /
                                                                                             str,
                                                                                             **extra_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#botcommandscopeschatadministrators>

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*) → 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`)

BotCommandScopeChatMember

```
class aiogram.types.bot_command_scope_chat_member.BotCommandScopeChatMember(*, type: Li-
                                                                                   teral[BotCommandScopeType.CH
                                                                                   =
                                                                                   BotCommandScopeType.CHAT_
                                                                                   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#botcommandscopeschatmember>

type: `Literal[BotCommandScopeType.CHAT_MEMBER]`

Scope type, must be *chat_member*

chat_id: `int | str`

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

```
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: Literal[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) → None
```

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

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

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

This object represents the bot's short description.

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

short_description: str

The bot's short description

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.

BusinessConnection

```
class aiogram.types.business_connection.BusinessConnection(*, id: str, user: User, user_chat_id:
                                                           int, date: datetime, can_reply: bool,
                                                           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)
```

Contains information about the start page settings of a Telegram Business account.

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) → 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

```
class aiogram.types.business_location.BusinessLocation(*, address: str, location: Location | None = None, **extra_data: Any)
```

Contains information about the location of a Telegram Business account.

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]
```

The 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)
```

Describes the opening hours of a business.

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) → 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(*, opening_minute: int, closing_minute: int, **extra_data: Any)
```

Describes an interval of time during which a business is open.

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) → 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

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

This object represents an incoming callback query from a callback button in an [inline keyboard](#). If the button that originated the query was attached to a message sent by the bot, the field `message` will be present. If the button was attached to a message sent via the bot (in [inline mode](#)), the field `inline_message_id` will be present. Exactly one of the fields `data` or `game_short_name` will be present.

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

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

id: str

Unique identifier for this query

from_user: User

Sender

`chat_instance: str`

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

`message: Message | 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) → 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) → 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>

Параметри

- `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.

Повертає

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

Chat

```
class aiogram.types.chat.Chat(*, id: int, type: str, title: str | None = None, username: str | None =
    None, first_name: str | None = None, last_name: str | None = None,
    is_forum: bool | None = None, accent_color_id: int | None = None,
    active_usernames: List[str] | None = None, available_reactions:
    List[ReactionTypeEmoji | ReactionTypeCustomEmoji] | None =
    None, background_custom_emoji_id: str | None = None, bio: str |
    None = None, birthdate: Birthdate | None = None, business_intro:
    BusinessIntro | None = None, business_location: BusinessLocation |
    None = None, business_opening_hours: BusinessOpeningHours |
    None = None, can_set_sticker_set: bool | None = None,
    custom_emoji_sticker_set_name: str | None = None, description: str
    | None = None, emoji_status_custom_emoji_id: str | None = None,
    emoji_status_expiration_date: datetime | None = None,
    has_aggressive_anti_spam_enabled: bool | None = None,
    has_hidden_members: bool | None = None, has_private_forwards:
    bool | None = None, has_protected_content: bool | None = None,
    has_restricted_voice_and_video_messages: bool | None = None,
    has_visible_history: bool | None = None, invite_link: str | None =
    None, join_by_request: bool | None = None, join_to_send_messages:
    bool | None = None, linked_chat_id: int | None = None, location:
    ChatLocation | None = None, message_auto_delete_time: int | None
    = None, permissions: ChatPermissions | None = None, personal_chat:
    Chat | None = None, photo: ChatPhoto | None = None,
    pinned_message: Message | None = None, profile_accent_color_id:
    int | None = None, profile_background_custom_emoji_id: str | None
    = None, slow_mode_delay: int | None = None, sticker_set_name: str
    | None = None, unrestrict_boost_count: int | 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 the 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)

`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`.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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`.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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`.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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`.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`bio: str | None`

Optional. Bio of the other party in a private chat. Returned only in `aiogram.methods.get_chat.GetChat`.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`birthdate: Birthdate | None`

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

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`business_intro: BusinessIntro | None`

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

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`business_location: BusinessLocation | None`

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

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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*.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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*.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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*.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`description`: str | None

Optional. Description, for groups, supergroups and channel chats. Returned only in *aiogram.methods.get_chat.GetChat*.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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*.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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*.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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*.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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*.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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`.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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`.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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`.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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`.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`invite_link: str | None`

Optional. Primary invite link, for groups, supergroups and channel chats. Returned only in `aiogram.methods.get_chat.GetChat`.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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`.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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`.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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`.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`location: ChatLocation | None`

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

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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`.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`permissions: ChatPermissions | None`

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

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`personal_chat: Chat | None`

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

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`photo: ChatPhoto | None`

Optional. Chat photo. Returned only in `aiogram.methods.get_chat.GetChat`.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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.

`pinned_message: Message | None`

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

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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`.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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`.

Застаріло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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`.

Застапіло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`sticker_set_name: str | None`

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

Застапіло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`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`.

Застапіло починаючи з версії API:7.3: <https://core.telegram.org/bots/api-changelog#may-6-2024>

`property shifted_id: int`

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

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

`property full_name: str`

Get full name of the Chat.

For private chat it is `first_name + last_name`. For other chat types it is title.

`ban_sender_chat(sender_chat_id: int, **kwargs: Any) → BanChatSenderChat`

Shortcut for method `aiogram.methods.ban_chat_sender_chat.BanChatSenderChat` will automatically fill method attributes:

- `chat_id`

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

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

Параметри

`sender_chat_id` – Unique identifier of the target sender chat

Повертає

instance of method `aiogram.methods.ban_chat_sender_chat.BanChatSenderChat`

`unban_sender_chat(sender_chat_id: int, **kwargs: Any) → UnbanChatSenderChat`

Shortcut for method `aiogram.methods.unban_chat_sender_chat.UnbanChatSenderChat` will automatically fill method attributes:

- `chat_id`

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

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

Параметри

`sender_chat_id` – Unique identifier of the target sender chat

Повертає

instance of method `aiogram.methods.unban_chat_sender_chat.UnbanChatSenderChat`

`get_administrators(**kwargs: Any) → GetChatAdministrators`

Shortcut for method `aiogram.methods.get_chat_administrators.GetChatAdministrators` will automatically fill method attributes:

- `chat_id`

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

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

Повертає

instance of method `aiogram.methods.get_chat_administrators.GetChatAdministrators`

`delete_message(message_id: int, **kwargs: Any) → DeleteMessage`

Shortcut for method `aiogram.methods.delete_message.DeleteMessage` will automatically fill method attributes:

- `chat_id`

Use this method to delete a message, including service messages, with the following limitations:

- A message can only be deleted if it was sent less than 48 hours ago.
- Service messages about a supergroup, channel, or forum topic creation can't be deleted.
- A dice message in a private chat can only be deleted if it was sent more than 24 hours ago.
- Bots can delete outgoing messages in private chats, groups, and supergroups.
- Bots can delete incoming messages in private chats.
- Bots granted `can_post_messages` permissions can delete outgoing messages in channels.
- If the bot is an administrator of a group, it can delete any message there.
- If the bot has `can_delete_messages` permission in a supergroup or a channel, it can delete any message there.

Returns `True` on success.

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

Параметри

`message_id` – Identifier of the message to delete

Повертає

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

`revoke_invite_link(invite_link: str, **kwargs: Any) → RevokeChatInviteLink`

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

- `chat_id`

Use this method to revoke an invite link created by the bot. If the primary link is revoked, a new link is automatically generated. The bot must be an administrator in the chat for this to work and must have the appropriate administrator rights. Returns the revoked invite link as `aiogram.types.chat_invite_link.ChatInviteLink` object.

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

Параметри

`invite_link` – The invite link to revoke

Повертає

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

`edit_invite_link(invite_link: str, name: str | None = None, expire_date: datetime.datetime | datetime.timedelta | int | None = None, member_limit: int | None = None, creates_join_request: bool | None = None, **kwargs: Any) → EditChatInviteLink`

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

- `chat_id`

Use this method to edit a non-primary invite link created by the bot. The bot must be an administrator in the chat for this to work and must have the appropriate administrator rights. Returns the edited invite link as a `aiogram.types.chat_invite_link.ChatInviteLink` object.

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

Параметри

- `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

Повертає

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

`create_invite_link(name: str | None = None, expire_date: datetime.datetime | datetime.timedelta | int | None = None, member_limit: int | None = None, creates_join_request: bool | None = None, **kwargs: Any) → CreateChatInviteLink`

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

- `chat_id`

Use this method to create an additional invite link for a chat. The bot must be an administrator in the chat for this to work and must have the appropriate administrator rights. The link can be revoked using the method `aiogram.methods.revoke_chat_invite_link.RevokeChatInviteLink`. Returns the new invite link as `aiogram.types.chat_invite_link.ChatInviteLink` object.

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

Параметри

- `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

Повертає

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

`export_invite_link(**kwargs: Any) → ExportChatInviteLink`

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

- `chat_id`

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

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

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

Повертає

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) → 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>

Параметри

- **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

Повертає

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

`delete_sticker_set(**kwargs: Any) → DeleteChatStickerSet`

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

- **chat_id**

Use this method to delete a group sticker set from a supergroup. The bot must be an administrator in the chat for this to work and must have the appropriate administrator rights. Use the field `can_set_sticker_set` optionally returned in `aiogram.methods.get_chat.GetChat` requests to check if the bot can use this method. Returns **True** on success.

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

Повертає

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

`set_sticker_set(sticker_set_name: str, **kwargs: Any) → SetChatStickerSet`

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

- **chat_id**

Use this method to set a new group sticker set for a supergroup. The bot must be an administrator in the chat for this to work and must have the appropriate administrator rights. Use the field `can_set_sticker_set` optionally returned in `aiogram.methods.get_chat.GetChat` requests to check if the bot can use this method. Returns **True** on success.

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

Параметри

sticker_set_name – Name of the sticker set to be set as the group sticker set

Повертає

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

`get_member(user_id: int, **kwargs: Any) → GetChatMember`

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

- `chat_id`

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

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

Параметри

`user_id` – Unique identifier of the target user

Повертає

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>

Повертає

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

`leave(**kwargs: Any) → LeaveChat`

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

- `chat_id`

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

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

Повертає

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

`unpin_all_messages(**kwargs: Any) → UnpinAllChatMessages`

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

- `chat_id`

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

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

Повертає

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

`unpin_message(message_id: int | None = None, **kwargs: Any) → UnpinChatMessage`

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

- `chat_id`

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

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

Параметри

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

Повертає

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

`pin_message(message_id: int, disable_notification: bool | None = None, **kwargs: Any) → PinChatMessage`

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

- **chat_id**

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

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

Параметри

- **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.

Повертає

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

`set_administrator_custom_title(user_id: int, custom_title: str, **kwargs: Any) → SetChatAdministratorCustomTitle`

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

- **chat_id**

Use this method to set a custom title for an administrator in a supergroup promoted by the bot. Returns **True** on success.

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

Параметри

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

Повертає

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

```
set_permissions(permissions: ChatPermissions, use_independent_chat_permissions: bool / None = None, **kwargs: Any) → SetChatPermissions
```

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

- `chat_id`

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

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

Параметри

- `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.

Повертає

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

```
promote(user_id: int, is_anonymous: bool / None = None, can_manage_chat: bool / None = None, can_delete_messages: bool / None = None, can_manage_video_chats: bool / None = None, can_restrict_members: bool / None = None, can_promote_members: bool / None = None, can_change_info: bool / None = None, can_invite_users: bool / None = None, can_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) → 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>

Параметри

- `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, post stories to the chat page, pin chat stories, and access the chat's story archive
- `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

Повєртає

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

`restrict(user_id: int, permissions: ChatPermissions, use_independent_chat_permissions: bool | None = None, until_date: datetime.datetime | datetime.timedelta | int | None = None, **kwargs: Any) → RestrictChatMember`

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

- `chat_id`

Use this method to restrict a user in a supergroup. The bot must be an administrator in the supergroup for this to work and must have the appropriate administrator rights. Pass `True` for all permissions to lift restrictions from a user. Returns `True` on success.

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

Параметри

- `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

Повертає

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

`unban(user_id: int, only_if_banned: bool | None = None, **kwargs: Any) → UnbanChatMember`

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

- `chat_id`

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

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

Параметри

- `user_id` – Unique identifier of the target user
- `only_if_banned` – Do nothing if the user is not banned

Повертає

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

`ban(user_id: int, until_date: datetime.datetime | datetime.timedelta | int | None = None, revoke_messages: bool | None = None, **kwargs: Any) → BanChatMember`

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

- `chat_id`

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

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

Параметри

- `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.

Повертає

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

`set_description(description: str / None = None, **kwargs: Any) → SetChatDescription`

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

- `chat_id`

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

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

Параметри

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

Повертає

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

`set_title(title: str, **kwargs: Any) → SetChatTitle`

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

- `chat_id`

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

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

Параметри

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

Повертає

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

`delete_photo(**kwargs: Any) → DeleteChatPhoto`

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

- `chat_id`

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

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

Повертає

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

`set_photo(photo: InputFile, **kwargs: Any) → SetChatPhoto`

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

- `chat_id`

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

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

Параметри

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

Повертає

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

`unpin_all_general_forum_topic_messages(**kwargs: Any) →`

`UnpinAllGeneralForumTopicMessages`

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

- chat_id

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

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

Повертає

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 / None = None,
    can_edit_messages:
    bool / None = None,
    can_pin_messages:
    bool / None = None,
    can_manage_topics:
    bool / None = None,
    **extra_data: Any)
```

Represents the rights of an administrator in a chat.

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

`is_anonymous: bool`

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

`can_manage_chat: bool`

True, if the administrator can access the chat event log, 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) → 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, post stories to the chat page, pin chat stories, and access the chat's story archive

`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

ChatBackground

```
class aiogram.types.chat_background.ChatBackground(*, type: BackgroundTypeFill /  
                                                    BackgroundTypeWallpaper /  
                                                    BackgroundTypePattern /  
                                                    BackgroundTypeChatTheme, **extra_data:  
                                                    Any)
```

This object represents a chat background.

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

`type: BackgroundTypeFill | BackgroundTypeWallpaper | BackgroundTypePattern |
BackgroundTypeChatTheme`

Type of the background

`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.

ChatBoost

```
class aiogram.types.chat_boost.ChatBoost(*, boost_id: str, add_date: datetime, expiration_date:  
datetime, source: ChatBoostSourcePremium /  
ChatBoostSourceGiftCode / ChatBoostSourceGiveaway,  
**extra_data: Any)
```

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) → 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

```
class aiogram.types.chat_boost_removed.ChatBoostRemoved(*, chat: Chat, boost_id: str,
                                                         remove_date: datetime, source:
                                                         ChatBoostSourcePremium /
                                                         ChatBoostSourceGiftCode /
                                                         ChatBoostSourceGiveaway,
                                                         **extra_data: Any)
```

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.

model_post_init(_ModelMetaclass__context: Any) → None

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

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(_ModelMetaclass__context: Any) → 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: Li-  
teral[ChatBoostSourceType.GIFT_CODE],  
= ChatBoostSourceType.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) → 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: Literal[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: Literal[ChatBoostSourceType.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) → 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

`class aiogram.types.chat_boost_updated.ChatBoostUpdated(*, chat: Chat, boost: ChatBoost, **extra_data: Any)`

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

ChatFullInfo


```

class aiogram.types.chat_full_info.ChatFullInfo(*, 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, accent_color_id:
                                                int, active_usernames: List[str] / None = None,
                                                available_reactions: List[ReactionTypeEmoji /
                                                ReactionTypeCustomEmoji] / None = None,
                                                background_custom_emoji_id: str / None =
                                                None, bio: str / None = None, birthdate:
                                                Birthdate / None = None, business_intro:
                                                BusinessIntro / None = None, business_location:
                                                BusinessLocation / None = None,
                                                business_opening_hours: BusinessOpeningHours
                                                / None = None, can_set_sticker_set: bool / None
                                                = None, custom_emoji_sticker_set_name: str /
                                                None = None, description: str / None = None,
                                                emoji_status_custom_emoji_id: str / None =
                                                None, emoji_status_expiration_date: datetime /
                                                None = None,
                                                has_aggressive_anti_spam_enabled: bool / None
                                                = None, has_hidden_members: bool / None =
                                                None, has_private_forwards: bool / None =
                                                None, has_protected_content: bool / None =
                                                None,
                                                has_restricted_voice_and_video_messages: bool
                                                / None = None, has_visible_history: bool / None
                                                = None, invite_link: str / None = None,
                                                join_by_request: bool / None = None,
                                                join_to_send_messages: bool / None = None,
                                                linked_chat_id: int / None = None, location:
                                                ChatLocation / None = None,
                                                message_auto_delete_time: int / None = None,
                                                permissions: ChatPermissions / None = None,
                                                personal_chat: Chat / None = None, photo:
                                                ChatPhoto / None = None, pinned_message:
                                                Message / None = None,
                                                profile_accent_color_id: int / None = None,
                                                profile_background_custom_emoji_id: str /
                                                None = None, slow_mode_delay: int / None =
                                                None, sticker_set_name: str / None = None,
                                                unrestrict_boost_count: int / None = None,
                                                max_reaction_count: int, **extra_data: Any)

```

This object contains full information about a chat.

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

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 the chat, can be either „private“, „group“, „supergroup“ or „channel“

`accent_color_id: int`

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.

`max_reaction_count: int`

The maximum number of reactions that can be set on a message in the chat

`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

`active_usernames: List[str] | None`

Optional. If non-empty, the list of all [active chat usernames](#); for private chats, supergroups and channels

`birthdate: Birthdate | None`

Optional. For private chats, the date of birth of the user

`business_intro: BusinessIntro | None`

Optional. For private chats with business accounts, the intro of the business

`business_location: BusinessLocation | None`

Optional. For private chats with business accounts, the location of the business

`business_opening_hours: BusinessOpeningHours | None`

Optional. For private chats with business accounts, the opening hours of the business

`personal_chat: Chat | None`

Optional. For private chats, the personal channel of the user

`available_reactions: List[ReactionTypeEmoji | ReactionTypeCustomEmoji] | None`

Optional. List of available reactions allowed in the chat. If omitted, then all [emoji reactions](#) are allowed.

`background_custom_emoji_id: str | None`

Optional. Custom emoji identifier of the emoji chosen by the chat for the reply header and link preview background

`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.

`profile_background_custom_emoji_id: str | None`

Optional. Custom emoji identifier of the emoji chosen by the chat for its profile background

`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

`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

`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.

`bio: str | None`

Optional. Bio of the other party in a private chat

`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

`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

`join_to_send_messages: bool | None`

Optional. True, if users need to join the supergroup before they can send messages

`join_by_request: bool | None`

Optional. True, if all users directly joining the supergroup need to be approved by supergroup administrators

`description: str | None`

Optional. Description, for groups, supergroups and channel chats

`invite_link: str | None`

Optional. Primary invite link, for groups, supergroups and channel chats

`pinned_message: Message | None`

Optional. The most recent pinned message (by sending date)

`permissions: ChatPermissions | None`

Optional. Default chat member permissions, for groups and supergroups

`slow_mode_delay: int | None`

Optional. For supergroups, the minimum allowed delay between consecutive messages sent by each unprivileged user; in seconds

`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

`message_auto_delete_time: int | None`

Optional. The time after which all messages sent to the chat will be automatically deleted; in seconds

`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.

`has_hidden_members: bool | None`

Optional. True, if non-administrators can only get the list of bots and administrators in the chat

`has_protected_content: bool | None`

Optional. True, if messages from the chat can't be forwarded to other chats

`has_visible_history: bool | None`

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

`sticker_set_name: str | None`

Optional. For supergroups, name of the group sticker set

`can_set_sticker_set: bool | None`

Optional. True, if the bot can change the group sticker set

`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.

`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.

`location: ChatLocation | None`

Optional. For supergroups, the location to which the supergroup is connected

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) → 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) → *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>

Повєрѡє

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

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

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

- chat_id
- user_id

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

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

Повєрѡє

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>

Параметри

- 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
- `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

Повертас

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>

Параметри

- `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
- `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

Повєртає

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>

Параметри

- **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
- **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

Повертає

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>

Параметри

- **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
- `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

Повєтрає

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>

Параметри

- `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
- `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

Повєртає

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>

Параметри

- **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
- **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

Повертає

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>

Параметри

- `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
- `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

Повертає

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>

Параметри

- `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
- `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

Повертає

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>

Параметри

- **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
- **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

Повєртає

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>

Параметри

- **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
- `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

Повєртає

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>

Параметри

- `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.
- `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

Повертає

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) → 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>

Параметри

- `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.
- `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

Повертає

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>

Параметри

- `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

Повертає

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>

Параметри

- `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

Повертає

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>

Параметри

- `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 during which the location will be updated (see [Live Locations](#), should be between 60 and 86400, or 0x7FFFFFFF for live locations that can be edited indefinitely.
- `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
- `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

Повертає

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>

Параметри

- `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 during which the location will be updated (see [Live Locations](#), should be between 60 and 86400, or 0x7FFFFFFF for live locations that can be edited indefinitely.
- `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

- `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

Повертає

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>

Параметри

- `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

Повертає

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>

Параметри

- `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

Повертає

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>

Параметри

- **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
- **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

Повертає

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>

Параметри

- **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
- **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

Повертає

instance of method `aiogram.methods.send_photo.SendPhoto`

```
answer_poll(question: str, options: List[Union[InputPollOption, str]], business_connection_id:
    Optional[str] = None, message_thread_id: Optional[int] = None,
    question_parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>,
    question_entities: Optional[List[MessageEntity]] = 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>

Параметри

- `question` – Poll question, 1-300 characters
- `options` – A JSON-serialized list of 2-10 answer options
- `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
- `question_parse_mode` – Mode for parsing entities in the question. See [formatting options](#) for more details. Currently, only custom emoji entities are allowed
- `question_entities` – A JSON-serialized list of special entities that appear in the poll question. It can be specified instead of `question_parse_mode`
- `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. It can be specified instead of `explanation_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
- `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

Повертас

instance of method `aiogram.methods.send_poll.SendPoll`

```
answer_poll_pm(question: str, options: List[Union[InputPollOption, str]], business_connection_id:
    Optional[str] = None, message_thread_id: Optional[int] = None,
    question_parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>,
    question_entities: Optional[List[MessageEntity]] = 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>

Параметри

- `question` – Poll question, 1-300 characters
- `options` – A JSON-serialized list of 2-10 answer options
- `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
- `question_parse_mode` – Mode for parsing entities in the question. See [formatting options](#) for more details. Currently, only custom emoji entities are allowed
- `question_entities` – A JSON-serialized list of special entities that appear in the poll question. It can be specified instead of `question_parse_mode`
- `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. It can be specified instead of `explanation_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
- `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

Повертає

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>

Параметри

- `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
- `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

Повертає

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) → 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>

Параметри

- `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
- `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

Повертає

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>

Параметри

- `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 a 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

Повєртає

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>

Параметри

- `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 a 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

Повєртає

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>

Параметри

- `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
- `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

Повертає

instance of method `aiogram.methods.send_venue.SendVenue`

```
answer_venue_pm(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>

Параметри

- `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
- `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

Повєртає

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>

Параметри

- `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
- **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

Повєрѡє

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) → 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>

Параметри

- **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
- `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

Повертає

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>

Параметри

- `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
- **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

Повєртає

instance of method *aiogram.methods.send_video_note.SendVideoNote*

```
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) → 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>

Параметри

- **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
- `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

Повертає

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, or in .MP3 format, or in .M4A format (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>

Параметри

- **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
- **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

Повертає

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, or in .MP3 format, or in .M4A format (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>

Параметри

- **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
- **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

Повертає

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) → 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: Li-
                                                                    teral[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 | None = None,
                                                                    can_edit_messages:
                                                                    bool | None = None,
                                                                    can_pin_messages:
                                                                    bool | None = None,
                                                                    can_manage_topics:
                                                                    bool | None = None,
                                                                    custom_title: str |
                                                                    None = None,
                                                                    **extra_data: 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) → 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, post stories to the chat page, pin chat stories, and access the chat's story archive

`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

```
class aiogram.types.chat_member_left.ChatMemberLeft(*, status: Literal[ChatMemberStatus.LEFT]
    = ChatMemberStatus.LEFT, user: User,
    **extra_data: Any)
```

Represents a `chat member` that isn't currently a member of the chat, but may join it themselves.

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

status: `Literal[ChatMemberStatus.LEFT]`

The member's status in the chat, always „left“

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

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*) → 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: Li-
                                                                teral[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

```
class aiogram.types.chat_member_updated.ChatMemberUpdated(*, chat: Chat, from_user: User, date:
    datetime, old_chat_member:
        ChatMemberOwner /
        ChatMemberAdministrator /
        ChatMemberMember /
        ChatMemberRestricted /
        ChatMemberLeft /
        ChatMemberBanned,
    new_chat_member:
        ChatMemberOwner /
        ChatMemberAdministrator /
        ChatMemberMember /
        ChatMemberRestricted /
        ChatMemberLeft /
        ChatMemberBanned, invite_link:
        ChatInviteLink / None = None,
    via_join_request: bool / None =
        None, via_chat_folder_invite_link:
        bool / None = None, **extra_data:
        Any)
```

This object represents changes in the status of a chat member.

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

chat: *Chat*

Chat the user belongs to

from_user: *User*

Performer of the action, which resulted in the change

date: *DateTime*

Date the change was done in Unix time

old_chat_member: *ChatMemberOwner* | *ChatMemberAdministrator* | *ChatMemberMember* | *ChatMemberRestricted* | *ChatMemberLeft* | *ChatMemberBanned*

Previous information about the chat member

new_chat_member: *ChatMemberOwner* | *ChatMemberAdministrator* | *ChatMemberMember* | *ChatMemberRestricted* | *ChatMemberLeft* | *ChatMemberBanned*

New information about the chat member

invite_link: *ChatInviteLink* | *None*

Optional. Chat invite link, which was used by the user to join the chat; for joining by invite link events only.

via_join_request: *bool* | *None*

Optional. True, if the user joined the chat after sending a direct join request and being approved by an administrator

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 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>

Параметри

- **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
- `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

Повєртає

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>

Параметри

- `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
- **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

Повертає

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>

Параметри

- **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
- `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

Повєртає

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>

Параметри

- `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
- `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

Повертає

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>

Параметри

- `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
- `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

Поведение

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>

Параметры

- `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.
- `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

Повєртає

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>

Параметри

- `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

Повертає

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>

Параметри

- `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 during which the location will be updated (see [Live Locations](#), should be between 60 and 86400, or 0x7FFFFFFF for live locations that can be edited indefinitely.
- `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
- `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

Повертаєinstance 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>

Параметри

- `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

Повертаєinstance 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>

Параметри

- `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
- `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

Повєтрає

instance of method `aiogram.methods.send_photo.SendPhoto`

```
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.


```

answer_poll(question: str, options: List[Union[InputPollOption, str]], business_connection_id:
    Optional[str] = None, message_thread_id: Optional[int] = None,
    question_parse_mode: Optional[Union[str, Default]] = <Default('parse_mode')>,
    question_entities: Optional[List[MessageEntity]] = 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>

Параметри

- `question` – Poll question, 1-300 characters
- `options` – A JSON-serialized list of 2-10 answer options
- `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
- `question_parse_mode` – Mode for parsing entities in the question. See [formatting options](#) for more details. Currently, only custom emoji entities are allowed
- `question_entities` – A JSON-serialized list of special entities that appear in the poll question. It can be specified instead of `question_parse_mode`
- `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. It can be specified instead of `explanation_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
- `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

Повєртає

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>

Параметри

- `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
- `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

Повертає

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>

Параметри

- `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 a 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

Повертає

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>

Параметри

- `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
- `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

Повертає

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>

Параметри

- `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
- **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

Повєтрає

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>

Параметри

- **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
- **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

Повертае

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, or in .MP3 format, or in .M4A format (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>

Параметри

- **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
- **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

Повертає

instance of method `aiogram.methods.send_voice.SendVoice`

ChatPermissions


```
class aiogram.types.chat_permissions.ChatPermissions(*, can_send_messages: bool | None =
    None, can_send_audios: bool | None =
    None, can_send_documents: bool | None =
    None, can_send_photos: bool | None =
    None, can_send_videos: bool | None =
    None, can_send_video_notes: bool | None
    = None, can_send_voice_notes: bool |
    None = None, can_send_polls: bool | None
    = None, can_send_other_messages: bool |
    None = None,
    can_add_web_page_previews: bool | None
    = None, can_change_info: bool | None =
    None, can_invite_users: bool | None =
    None, can_pin_messages: bool | None =
    None, can_manage_topics: bool | None =
    None, **extra_data: Any)
```

Describes actions that a non-administrator user is allowed to take in a chat.

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

can_send_messages: bool | None

Optional. True, if the user is allowed to send text messages, contacts, 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

```
class aiogram.types.chat_photo.ChatPhoto(*, small_file_id: str, small_file_unique_id: str,
                                          big_file_id: str, big_file_unique_id: str, **extra_data:
                                          Any)
```

This object represents a chat photo.

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

`small_file_id: str`

File identifier of small (160x160) chat photo. This `file_id` can be used only for photo download and only for as long as the photo is not changed.

`small_file_unique_id: str`

Unique file identifier of small (160x160) chat photo, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.

`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.

`big_file_id: str`

File identifier of big (640x640) chat photo. This `file_id` can be used only for photo download and only for as long as the photo is not changed.

`big_file_unique_id: str`

Unique file identifier of big (640x640) chat photo, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.

ChatShared

```
class aiogram.types.chat_shared.ChatShared(*, request_id: int, chat_id: int, title: str | None =
                                           None, username: str | None = None, photo:
                                           List[PhotoSize] | None = None, **extra_data: Any)
```

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

```
class aiogram.types.contact.Contact(*, phone_number: str, first_name: str, last_name: str | None
    = None, user_id: int | None = None, vcard: str | None =
    None, **extra_data: Any)
```

This object represents a phone contact.

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

phone_number: str
Contact's phone number

first_name: str
Contact's first name

last_name: str | None
Optional. Contact's last name

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 | 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) → 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

```
class aiogram.types.document.Document(*, file_id: str, file_unique_id: str, thumbnail: PhotoSize /  
None = None, file_name: str / None = None, mime_type:  
str / None = None, file_size: int / None = None,  
**extra_data: Any)
```

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

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

file_id: str

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

file_unique_id: str

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

thumbnail: [PhotoSize](#) | None

Optional. Document thumbnail as defined by sender

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_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(*, origin: MessageOriginUser /
    MessageOriginHiddenUser /
    MessageOriginChat /
    MessageOriginChannel, chat: Chat /
    None = None, message_id: int /
    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,
    has_media_spoiler: bool / None =
    None, contact: Contact / None =
    None, dice: Dice / None = None,
    game: Game / None = None,
    giveaway: Giveaway / None = None,
    giveaway_winners: GiveawayWinners
    / None = None, invoice: Invoice /
    None = None, location: Location /
    None = None, poll: Poll / None =
    None, venue: Venue / None = None,
    **extra_data: Any)
```

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) → 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

```
class aiogram.types.force_reply.ForceReply(*, force_reply: Literal[True] = True,
                                           input_field_placeholder: str | None = None, selective:
                                           bool | None = None, **extra_data: Any)
```

Upon receiving a message with this object, Telegram clients will display a reply interface to the user (act as if the user has selected the bot's message and tapped „Reply“). This can be extremely useful if you want to create user-friendly step-by-step interfaces without having to sacrifice [privacy mode](#). Not supported in channels and for messages sent on behalf of a Telegram Business account.

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

```
class aiogram.types.forum_topic.ForumTopic(*, message_thread_id: int, name: str, icon_color: int,
                                           icon_custom_emoji_id: str | None = None,
                                           **extra_data: Any)
```

This object represents a forum topic.

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

message_thread_id: `int`

Unique identifier of the forum topic

`name: str`

Name of the topic

`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`

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) → None`

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

ForumTopicCreated

`class aiogram.types.forum_topic_created.ForumTopicCreated(*, name: str, icon_color: int, icon_custom_emoji_id: str | None = None, **extra_data: Any)`

This object represents a service message about a new forum topic created in the chat.

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

`name: str`

Name of the topic

`icon_color: int`

Color of the topic icon in RGB format

`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

```
class aiogram.types.forum_topic_edited.ForumTopicEdited(*, name: str | None = None,
                                                         icon_custom_emoji_id: str | None =
                                                         None, **extra_data: Any)
```

This object represents a service message about an edited forum topic.

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

name: str | None

Optional. New name of the topic, if it was edited

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. 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*) → 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*) → 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) → None
```

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

Giveaway

```
class aiogram.types.giveaway.Giveaway(*, chats: List[Chat], winners_selection_date: datetime,
    winner_count: int, only_new_members: bool | None = None,
    has_public_winners: bool | None = None, prize_description: str | None = None,
    country_codes: List[str] | None = None, premium_subscription_month_count: int | None = None,
    **extra_data: Any)
```

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) → 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) → 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) → 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

```
class aiogram.types.inaccessible_message.InaccessibleMessage(*, chat: Chat, message_id: int,
                                                            date: Literal[0] = 0,
                                                            **extra_data: Any)
```

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) → 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(*, text: str, url: str | None =
    None, callback_data: str |
    None = None, web_app:
    WebAppInfo | None = None,
    login_url: LoginUrl | None =
    None, switch_inline_query:
    str | None = None, swi-
    tch_inline_query_current_chat:
    str | None = None, swi-
    tch_inline_query_chosen_chat:
    SwitchInli-
    neQueryChosenChat | 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. Not supported for messages sent on behalf of a Telegram Business account.

`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. Not supported for messages sent on behalf of a Telegram Business account.

`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) → 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. Not supported for messages sent on behalf of a Telegram Business account.

`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. Not supported for messages sent on behalf of a Telegram Business account.

`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)
    → BufferedInputFile
```

Create buffer from file

Параметри

- `path` – Path to file
- `filename` – Filename to be propagated to telegram. By default, will be parsed from `path`
- `chunk_size` – Uploading chunk size

Повертає

instance of *BufferedInputFile*

```
async read(bot: Bot) → AsyncGenerator[bytes, None]
```

```
class aiogram.types.input_file.FSInputFile(path: str | Path, filename: str | None = None,
    chunk_size: int = 65536)
```

```
async read(bot: Bot) → AsyncGenerator[bytes, None]
```

```
class aiogram.types.input_file.URLInputFile(url: str, headers: Dict[str, Any] | None = None,
    filename: str | None = None, chunk_size: int = 65536, timeout: int = 30, bot: 'Bot' | None = None)
```

```
async read(bot: Bot) → AsyncGenerator[bytes, None]
```

InputMedia

```
class aiogram.types.input_media.InputMedia(**extra_data: Any)
```

This object represents the content of a media message to be sent. It should be one of

- *aiogram.types.input_media_animation.InputMediaAnimation*
- *aiogram.types.input_media_document.InputMediaDocument*
- *aiogram.types.input_media_audio.InputMediaAudio*
- *aiogram.types.input_media_photo.InputMediaPhoto*
- *aiogram.types.input_media_video.InputMediaVideo*

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


```
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.

InputMediaAnimation

```
class aiogram.types.input_media_animation.InputMediaAnimation(*, type: ~typing.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.MessageEntity]
| None = None, width: int |
None = None, height: int | None
= None, duration: int | None =
None, has_spoiler: bool | None
= None, **extra_data:
~typing.Any)
```

Represents an animation file (GIF or H.264/MPEG-4 AVC video without sound) to be sent.

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

type: `Literal[InputMediaType.ANIMATION]`

Type of the result, must be *animation*

media: `str | InputFile`

File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet, or pass „attach://<file_attach_name>“ to upload a new one using multipart/form-data under <file_attach_name> name. *More information on Sending Files »*

thumbnail: `InputFile | None`

Optional. Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass „attach://<file_attach_name>“ if the thumbnail was uploaded using multipart/form-data under <file_attach_name>. *More information on Sending Files »*

caption: `str | None`

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

`parse_mode: str | 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) → 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: ~typi-
    ng.List[~aiogram.types.message_entity.MessageEntity]
    | None = None, duration: int | None =
    None, performer: str | None = None,
    title: str | None = None, **extra_data:
    ~typing.Any)
```

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

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

`type: Literal[InputMediaType.AUDIO]`

Type of the result, must be *audio*

`media: str | InputFile`

File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet, or pass „attach://<file_attach_name>“ to upload a new one using multipart/form-data under <file_attach_name> name. *More information on Sending Files »*

`thumbnail: InputFile | None`

Optional. Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass „attach://<file_attach_name>“ if the thumbnail was uploaded using multipart/form-data under <file_attach_name>. *More information on Sending Files »*

`caption: str | None`

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

`parse_mode: str | 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: ~typing.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 server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass `,attach://<file_attach_name>` if the thumbnail was uploaded using multipart/form-data under `<file_attach_name>`. *More information on Sending Files »*

`caption: str | None`

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

`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 document caption. See *formatting options* for more details.

`caption_entities: List[MessageEntity] | None`

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

`disable_content_type_detection: bool | None`

Optional. Disables automatic server-side content type detection for files uploaded using multipart/form-data. Always `True`, if the document is sent as part of an album.

InputMediaPhoto

```
class aiogram.types.input_media_photo.InputMediaPhoto(*, type:
    ~typing.Literal[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: ~typi-
    ng.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: ~typi-
    ng.List[~aiogram.types.message_entity.MessageEntity]
    | None = None, width: int | None =
    None, height: int | None = None,
    duration: int | None = None,
    supports_streaming: bool | None = None,
    has_spoiler: bool | None = None,
    **extra_data: ~typing.Any)
```

Represents a video to be sent.

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

`type: Literal[InputMediaType.VIDEO]`

Type of the result, must be *video*

`media: str | InputFile`

File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet, or pass `„attach://<file_attach_name>“` to upload a new one using multipart/form-data under `<file_attach_name>` name. *More information on Sending Files »*

`thumbnail: InputFile | None`

Optional. Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file, so you can pass `„attach://<file_attach_name>“` if the thumbnail was uploaded using multipart/form-data under `<file_attach_name>`. *More information on Sending Files »*

`caption: str | None`

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

`parse_mode: str | 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) → 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

InputPollOption

```
class aiogram.types.input_poll_option.InputPollOption(*, text: str, text_parse_mode: str |
    ~aiogram.client.default.Default | None =
    ~Default('parse_mode')>, text_entities:
    ~typing.List[~aiogram.types.message_entity.MessageEntity]
    / None = None, **extra_data:
    ~typing.Any)
```

This object contains information about one answer option in a poll to send.

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

`text: str`

Option text, 1-100 characters

`text_parse_mode: str | Default | None`

Optional. Mode for parsing entities in the text. See [formatting options](#) for more details. Currently, only custom emoji entities are allowed

`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_entities: List[MessageEntity] | None`

Optional. A JSON-serialized list of special entities that appear in the poll option text. It can be specified instead of `text_parse_mode`

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.

Застаріло починаючи з версії API:7.0: <https://core.telegram.org/bots/api-changelog#december-29-2023>

KeyboardButtonPollType

```
class aiogram.types.keyboard_button_poll_type.KeyboardButtonPollType(*, type: str | None =  
                                                                    None, **extra_data:  
                                                                    Any)
```

This object represents type of a poll, which is allowed to be created and sent when the corresponding button is pressed.

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

`type: str | None`

Optional. If *quiz* is passed, the user will be allowed to create only polls in the quiz mode. If *regular* is passed, only regular polls will be allowed. Otherwise, the user will be allowed to create a poll of any type.

`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 | None =
    None,
    chat_has_username:
    bool | None =
    None,
    chat_is_created:
    bool | None =
    None,
    user_administrator_rights:
    ChatAdmini-
    stratorRights /
    None = None,
    bot_administrator_rights:
    ChatAdmini-
    stratorRights /
    None = None,
    bot_is_member:
    bool | None =
    None,
    request_title:
    bool | None =
    None,
    request_username:
    bool | None =
    None,
    request_photo:
    bool | 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 chat 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 »](#)

Застаріло починаючи з версії 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 | None =
                                                                    None,
                                                                    user_is_premium:
                                                                    bool | None =
                                                                    None,
                                                                    max_quantity:
                                                                    int | None =
                                                                    None,
                                                                    request_name:
                                                                    bool | None =
                                                                    None,
                                                                    request_username:
                                                                    bool | None =
                                                                    None,
                                                                    request_photo:
                                                                    bool | 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 names

`request_username: bool | None`

Optional. Pass **True** to request the users' usernames

`request_photo: bool | None`

Optional. Pass **True** to request the users' photos

LinkPreviewOptions

```
class aiogram.types.link_preview_options.LinkPreviewOptions(*, is_disabled: bool |
    ~aiogram.client.default.Default |
    None =
    <Default('link_preview_is_disabled')>,
    url: str | 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 =
    <Default('link_preview_prefer_large_media')>,
    show_above_text: bool |
    ~aiogram.client.default.Default |
    None =
    <Default('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) → 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

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

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) → 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

```
class aiogram.types.login_url.LoginUrl(*, url: str, forward_text: str | None = None,
                                       bot_username: str | None = None, request_write_access:
                                       bool | None = None, **extra_data: Any)
```

This object represents a parameter of the inline keyboard button used to automatically authorize a user. Serves as a great replacement for the [Telegram Login Widget](#) when the user is coming from Telegram. All the user needs to do is tap/click a button and confirm that they want to log in: Telegram apps support these buttons as of [version 5.7](#).

Sample bot: [@discussbot](#)

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

url: str

An HTTPS URL to be opened with user authorization data added to the query string when the button is pressed. If the user refuses to provide authorization data, the original URL without information about the user will be opened. The data added is the same as described in [Receiving authorization data](#).

forward_text: str | None

Optional. New text of the button in forwarded messages.

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.

MaybeInaccessibleMessage

```
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.

model_post_init(_ModelMetaclass__context: Any) → None

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

MenuButton

```
class aiogram.types.menu_button.MenuButton(*, type: str, text: str | None = None, web_app:
                                         WebAppInfo | None = None, **extra_data: Any)
```

This object describes the bot's menu button in a private chat. It should be one of

- `aiogram.types.menu_button_commands.MenuButtonCommands`
- `aiogram.types.menu_button_web_app.MenuButtonWebApp`
- `aiogram.types.menu_button_default.MenuButtonDefault`

If a menu button other than `aiogram.types.menu_button_default.MenuButtonDefault` is set for a private chat, then it is applied in the chat. Otherwise the default menu button is applied. By default, the menu button opens the list of bot commands.

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

`type: str`

Type of the button

`text: str | None`

Optional. Text on the button

`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: Li-
                                                            teral[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.AnswerWebAppQuery](#).

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,
    chat_background_set: ChatBackground / 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: Розділ 2. Зміст
    GiveawayWinners / None = None, giveaway_completed:
    GiveawayCompleted / None = None, video_chat_scheduled:
    VideoChatScheduled / None = None, video_chat_started:

```

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

chat_background_set: *ChatBackground* | None

Optional. Service message: chat background set

forum_topic_created: *ForumTopicCreated* | None

Optional. Service message: forum topic created

forum_topic_edited: *ForumTopicEdited* | None

Optional. Service message: forum topic edited

forum_topic_closed: *ForumTopicClosed* | None

Optional. Service message: forum topic closed

forum_topic_reopened: *ForumTopicReopened* | None

Optional. Service message: forum topic reopened

general_forum_topic_hidden: *GeneralForumTopicHidden* | None

Optional. Service message: the „General“ forum topic hidden

general_forum_topic_unhidden: *GeneralForumTopicUnhidden* | None

Optional. Service message: the „General“ forum topic unhidden

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_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

Застаріло починаючи з версії 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

Застаріло починаючи з версії 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

Застаріло починаючи з версії 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

Застаріло починаючи з версії 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

Застаріло починаючи з версії 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

Застаріло починаючи з версії 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

Застаріло починаючи з версії 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>

Параметри

- **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
- `allow_sending_without_reply` – Pass `True` if the message should be sent even if the specified replied-to message is not found

Повєртає

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>

Параметри

- `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
- **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

Повертае

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>

Параметри

- **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
- **allow_sending_without_reply** – Pass `True` if the message should be sent even if the specified replied-to message is not found

Повертає

instance of method `aiogram.methods.send_audio.SendAudio`

```

answer_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, 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`
- `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>

Параметри

- **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
- `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

Повєртає

instance of method [aiogram.methods.send_audio.SendAudio](#)

```
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) → SendContact
```

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>

Параметри

- `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
- `allow_sending_without_reply` – Pass `True` if the message should be sent even if the specified replied-to message is not found

Повертаєinstance 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>

Параметри

- `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
- `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

Повертаєinstance of method `aiogram.methods.send_contact.SendContact`


```
reply_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, **kwargs: Any) → SendDocument
```

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>

Параметри

- **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
- **allow_sending_without_reply** – Pass `True` if the message should be sent even if the specified replied-to message is not found

Повертає

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>

Параметри

- **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
- `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

Повєртає

instance of method `aiogram.methods.send_document.SendDocument`

```
reply_game(game_short_name: str, 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) → SendGame
```

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>

Параметри

- `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.
- `allow_sending_without_reply` – Pass `True` if the message should be sent even if the specified replied-to message is not found

Повертаєinstance of method `aiogram.methods.send_game.SendGame`

```
answer_game(game_short_name: str, 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`
- `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>

Параметри

- `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.
- `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

Повертаєinstance 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>

Параметри

- `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

Повєрѡє

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>

Параметри

- `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

Повєртає

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>

Параметри

- `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 during which the location will be updated (see `Live Locations`, should be between 60 and 86400, or 0x7FFFFFFF for live locations that can be edited indefinitely.

- **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
- **allow_sending_without_reply** – Pass **True** if the message should be sent even if the specified replied-to message is not found

Повєртає

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>

Параметри

- **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 during which the location will be updated (see [Live Locations](#), should be between 60 and 86400, or 0x7FFFFFFF for live locations that can be edited indefinitely.
- **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
- `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

Повертає

instance of method `aiogram.methods.send_location.SendLocation`

```
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)
    → 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>

Параметри

- `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

Повертає

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>

Параметри

- `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

Повертає

instance of method `aiogram.methods.send_message.SendMessage`

```
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>

Параметри

- `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
- `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

Повертає

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>

Параметри

- `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
- `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

Повертає

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>

Параметри

- `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
- **allow_sending_without_reply** – Pass **True** if the message should be sent even if the specified replied-to message is not found

Повєрѡє

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>

Пєрємєтєрѡ

- **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
- **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

Повєрѡає

instance of method [aiogram.methods.send_photo.SendPhoto](#)

```
reply_poll(question: str, options: List[Union[InputPollOption, str]], question_parse_mode:
Optional[Union[str, Default]] = <Default('parse_mode')>, question_entities:
Optional[List[MessageEntity]] = 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, **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>

Параметри

- **question** – Poll question, 1-300 characters
- **options** – A JSON-serialized list of 2-10 answer options
- **question_parse_mode** – Mode for parsing entities in the question. See [formatting options](#) for more details. Currently, only custom emoji entities are allowed
- **question_entities** – A JSON-serialized list of special entities that appear in the poll question. It can be specified instead of *question_parse_mode*
- **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. It can be specified instead of *explanation_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
- **allow_sending_without_reply** – Pass **True** if the message should be sent even if the specified replied-to message is not found

Повертає

instance of method *aiogram.methods.send_poll.SendPoll*


```

answer_poll(question: str, options: List[Union[InputPollOption, str]], question_parse_mode:
    Optional[Union[str, Default]] = <Default('parse_mode')>, question_entities:
    Optional[List[MessageEntity]] = 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`
- `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>

Параметри

- `question` – Poll question, 1-300 characters
- `options` – A JSON-serialized list of 2-10 answer options
- `question_parse_mode` – Mode for parsing entities in the question. See [formatting options](#) for more details. Currently, only custom emoji entities are allowed
- `question_entities` – A JSON-serialized list of special entities that appear in the poll question. It can be specified instead of `question_parse_mode`
- `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. It can be specified instead of `explanation_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
- `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

Повєртає

instance of method `aiogram.methods.send_poll.SendPoll`

```
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) → SendDice
```

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>

Параметри

- `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
- `allow_sending_without_reply` – Pass `True` if the message should be sent even if the specified replied-to message is not found

Повертає

instance of method `aiogram.methods.send_dice.SendDice`

```
answer_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,
            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`
- `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>

Параметри

- `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
- `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

Повертає

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]] =
            <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) →
            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>

Параметри

- `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 a 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

Повертає

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>

Параметри

- **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 a 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

Повєтрає

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>

Параметри

- **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
- `allow_sending_without_reply` – Pass `True` if the message should be sent even if the specified replied-to message is not found

Повєтрає

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>

Параметри

- `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
- `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

Повертає

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>

Параметри

- `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
- **allow_sending_without_reply** – Pass **True** if the message should be sent even if the specified replied-to message is not found

Повєртає

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>

Параметри

- `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
- `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

Повертає

instance of method `aiogram.methods.send_video.SendVideo`

```
reply_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, **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`
- `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>

Параметри

- **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
- **allow_sending_without_reply** – Pass `True` if the message should be sent even if the specified replied-to message is not found

Повептрає

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>

Параметри

- **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
- **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

Повєртає

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, or in .MP3 format, or in .M4A format (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>

Параметри

- `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

- `allow_sending_without_reply` – Pass `True` if the message should be sent even if the specified replied-to message is not found

Повертає

instance of method `aiogram.methods.send_voice.SendVoice`

```
answer_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, 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`
- `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, or in .MP3 format, or in .M4A format (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>

Параметри

- `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

- `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

Повертає

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`

Примітка: This method doesn't use the API method named `copyMessage` and historically implemented before the similar method is added to API

Параметри

- `chat_id` –
- `disable_notification` –
- `reply_to_message_id` –
- `reply_parameters` –
- `reply_markup` –
- `allow_sending_without_reply` –
- `message_thread_id` –
- `parse_mode` –

Повертає

```
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>

Параметри

- `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 a 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

Повертає

instance of method `aiogram.methods.copy_message.CopyMessage`

```
edit_text(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>

Параметри

- `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

Повертає

instance of method `aiogram.methods.edit_message_text.EditMessageText`

```
forward(chat_id: Union[int, str], message_thread_id: Optional[int] = None, disable_notification:
Optional[bool] = None, protect_content: Optional[Union[bool, Default]] =
<Default('protect_content')>, **kwargs: Any) → ForwardMessage
```

Shortcut for method `aiogram.methods.forward_message.ForwardMessage` will automatically fill method attributes:

- `from_chat_id`
- `message_id`

Use this method to forward messages of any kind. Service messages 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>

Параметри

- `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

Повертає

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>

Параметри

- `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.

Повертає

instance of method `aiogram.methods.edit_message_media.EditMessageMedia`

`edit_reply_markup(inline_message_id: str | None = None, reply_markup: InlineKeyboardMarkup | None = None, **kwargs: Any) → EditMessageReplyMarkup`

Shortcut for method `aiogram.methods.edit_message_reply_markup.EditMessageReplyMarkup` will automatically fill method attributes:

- `chat_id`
- `message_id`

Use this method to edit only the reply markup of messages. On success, if the edited message is not an inline message, the edited `aiogram.types.message.Message` is returned, otherwise `True` is returned.

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

Параметри

- `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.

Повертає

instance of method `aiogram.methods.edit_message_reply_markup.EditMessageReplyMarkup`

`delete_reply_markup(inline_message_id: str | None = None, **kwargs: Any) → EditMessageReplyMarkup`

Shortcut for method `aiogram.methods.edit_message_reply_markup.EditMessageReplyMarkup` will automatically fill method attributes:

- `chat_id`
- `message_id`
- `reply_markup`

Use this method to edit only the reply markup of messages. On success, if the edited message is not an inline message, the edited `aiogram.types.message.Message` is returned, otherwise `True` is returned.

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

Параметри

`inline_message_id` – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message

Повертає

instance of method `aiogram.methods.edit_message_reply_markup`.
`EditMessageReplyMarkup`

`edit_live_location(latitude: float, longitude: float, inline_message_id: str / None = None, live_period: int / None = None, horizontal_accuracy: float / None = None, heading: int / None = None, proximity_alert_radius: int / None = None, reply_markup: InlineKeyboardMarkup / None = None, **kwargs: Any) → EditMessageLiveLocation`

Shortcut for method `aiogram.methods.edit_message_live_location`.
`EditMessageLiveLocation` will automatically fill method attributes:

- `chat_id`
- `message_id`

Use this method to edit live location messages. A location can be edited until its `live_period` expires or editing is explicitly disabled by a call to `aiogram.methods.stop_message_live_location`.
`StopMessageLiveLocation`. On success, if the edited message is not an inline message, the edited `aiogram.types.message.Message` is returned, otherwise `True` is returned.

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

Параметри

- `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
- `live_period` – New period in seconds during which the location can be updated, starting from the message send date. If `0x7FFFFFFF` is specified, then the location can be updated forever. Otherwise, the new value must not exceed the current `live_period` by more than a day, and the live location expiration date must remain within the next 90 days. If not specified, then `live_period` remains unchanged
- `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.

Повертає

instance of method `aiogram.methods.edit_message_live_location`.
`EditMessageLiveLocation`

`stop_live_location(inline_message_id: str / None = None, reply_markup: InlineKeyboardMarkup / None = None, **kwargs: Any) → StopMessageLiveLocation`

Shortcut for method `aiogram.methods.stop_message_live_location.StopMessageLiveLocation` will automatically fill method attributes:

- `chat_id`
- `message_id`

Use this method to stop updating a live location message before `live_period` expires. On success, if the message is not an inline message, the edited `aiogram.types.message.Message` is returned, otherwise `True` is returned.

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

Параметри

- `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.

Повертає

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>

Параметри

- `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.

Повертає

instance of method `aiogram.methods.edit_message_caption.EditMessageCaption`

```
delete(**kwargs: Any) → DeleteMessage
```

Shortcut for method `aiogram.methods.delete_message.DeleteMessage` will automatically fill method attributes:

- `chat_id`
- `message_id`

Use this method to delete a message, including service messages, with the following limitations:

- A message can only be deleted if it was sent less than 48 hours ago.
- Service messages about a supergroup, channel, or forum topic creation can't be deleted.
- A dice message in a private chat can only be deleted if it was sent more than 24 hours ago.
- Bots can delete outgoing messages in private chats, groups, and supergroups.
- Bots can delete incoming messages in private chats.
- Bots granted `can_post_messages` permissions can delete outgoing messages in channels.
- If the bot is an administrator of a group, it can delete any message there.
- If the bot has `can_delete_messages` permission in a supergroup or a channel, it can delete any message there.

Returns **True** on success.

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

Повєтрає

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>

Параметри

`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.

Повєтрає

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

`unpin(**kwargs: Any) → UnpinChatMessage`

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

- `chat_id`
- `message_id`

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

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

Повертає

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

`get_url(force_private: bool = False) → str | None`

Returns message URL. Cannot be used in private (one-to-one) chats. If chat has a username, returns URL like `https://t.me/username/message_id` Otherwise (or if `{force_private}` flag is set), returns `https://t.me/c/shifted_chat_id/message_id`

Параметри

`force_private` – if set, a private URL is returned even for a public chat

Повертає

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>

Параметри

- `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

Повертає

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

```
class aiogram.types.message_entity.MessageEntity(*, type: str, offset: int, length: int, url: str |
    None = None, user: User | None = None,
    language: str | None = None, custom_emoji_id:
    str | None = None, **extra_data: Any)
```

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

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

type: str

Type of the entity. Currently, can be „mention“ (@username), „hashtag“ (#hashtag), „cashtag“ (\$USD), „bot_command“ (/start@jobs_bot), „url“ (<https://telegram.org>), „email“ (do-not-reply@telegram.org), „phone_number“ (+1-212-555-0123), „bold“ (**bold text**), „italic“ (*italic text*), „underline“ (underlined text), „strikethrough“ (strikethrough text), „spoiler“ (spoiler message), „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) → str
```

MessageId

```
class aiogram.types.message_id.MessageId(*, message_id: int, **extra_data: Any)
```

This object represents a unique message identifier.

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

message_id: int

Unique message identifier

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.

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.

model_post_init(*_ModelMetaclass__context: Any*) → None

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: Li-  
                                                                    teral[MessageOriginType.CHANNEL]  
                                                                    = MessageOri-  
                                                                    ginalType.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[MessageType.CHAT] =
    MessageType.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[MessageType.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) → 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: Li-  
                                                                    teral[MessageOriginType.HIDDEN_US  
                                                                    = MessageOrig  
                                                                    nType.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) → 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) → 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) → 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) → 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) → 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

```
class aiogram.types.poll.Poll(*, id: str, question: str, options: List[PollOption], total_voter_count:
    int, is_closed: bool, is_anonymous: bool, type: str,
    allows_multiple_answers: bool, question_entities: List[MessageEntity]
    / None = None, correct_option_id: int / None = None, explanation:
    str / None = None, explanation_entities: List[MessageEntity] / None
    = None, open_period: int / None = None, close_date: datetime /
    None = None, **extra_data: Any)
```

This object contains information about a poll.

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

```
id: str
    Unique poll identifier

question: str
    Poll question, 1-300 characters

options: List[PollOption]
    List of poll options

total_voter_count: int
    Total number of users that voted in the poll

is_closed: bool
    True, if the poll is closed

is_anonymous: bool
    True, if the poll is anonymous

type: str
    Poll type, currently can be „regular“ or „quiz“

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.

allows_multiple_answers: bool
    True, if the poll allows multiple answers

question_entities: List[MessageEntity] | None
    Optional. Special entities that appear in the question. Currently, only custom emoji entities are
    allowed in poll questions

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) → 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, text_entities:
                                           List[MessageEntity] / None = None, **extra_data:
                                           Any)
```

This object contains information about one answer option in a poll.

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

`text: str`

Option text, 1-100 characters

`voter_count: int`

Number of users that voted for this option

`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_entities: List[MessageEntity] | None`

Optional. Special entities that appear in the option *text*. Currently, only custom emoji entities are allowed in poll option texts

ProximityAlertTriggered

```
class aiogram.types.proximity_alert_triggered.ProximityAlertTriggered(*, traveler: User,
                                                                    watcher: User,
                                                                    distance: int,
                                                                    **extra_data: Any)
```

This object represents the content of a service message, sent whenever a user in the chat triggers a proximity alert set by another user.

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

`traveler: User`

User that triggered the alert

`watcher: User`

User that set the alert

`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.

`distance: int`

The distance between the users

ReactionCount

```
class aiogram.types.reaction_count.ReactionCount(*, type: ReactionTypeEmoji /
                                                  ReactionTypeCustomEmoji, total_count: int,
                                                  **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) → 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: Li-  
teral[ReactionTypeType.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.

$$\text{model_post_init}(_ModelMetaclass_ \text{context}: Any) \rightarrow \text{None}$$

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

```
emoji: str
```

[illegible]

ReplyKeyboardMarkup

```
class aiogram.types.reply_keyboard.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). Not supported in channels and for messages sent on behalf of a Telegram Business account.

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

```
class aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove(*, remove_keyboard:
    Literal[True] = True, selective:
    bool | None = None,
    **extra_data: Any)
```

Upon receiving a message with this object, Telegram clients will remove the current custom keyboard and display the default letter-keyboard. By default, custom keyboards are displayed until a new keyboard is sent by a bot. An exception is made for one-time keyboards that are hidden immediately after the user presses a button (see `aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup`). Not supported in channels and for messages sent on behalf of a Telegram Business account.

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.

`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 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


```
class aiogram.types.reply_parameters.ReplyParameters(*, message_id: int, chat_id: int | str |
                                                    None = None,
                                                    allow_sending_without_reply: bool |
                                                    ~aiogram.client.default.Default | None =
                                                    <Default('allow_sending_without_reply')>,
                                                    quote: str | None = None,
                                                    quote_parse_mode: str |
                                                    ~aiogram.client.default.Default | None =
                                                    <Default('parse_mode')>, quote_entities:
                                                    ~typing.
                                                    List[~aiogram.types.message_entity.MessageEntity]
                                                    | None = None, quote_position: int | None
                                                    = None, **extra_data: ~typing.Any)
```

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

```
class aiogram.types.response_parameters.ResponseParameters(*, migrate_to_chat_id: int | None
                                                         = None, retry_after: int | None =
                                                         None, **extra_data: Any)
```

Describes why a request was unsuccessful.

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

migrate_to_chat_id: int | None

Optional. The group has been migrated to a supergroup with the specified identifier. This number may have more than 32 significant bits and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a signed 64-bit integer or double-precision float type are safe for storing this identifier.

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.

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

```
class aiogram.types.shared_user.SharedUser(*, user_id: int, first_name: str | None = None,
                                           last_name: str | None = None, username: str | None
                                           = None, photo: List[PhotoSize] | None = None,
                                           **extra_data: Any)
```

This object contains information about a user that was shared with the bot using a *aiogram.types.keyboard_button_request_users.KeyboardButtonRequestUsers* 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,
    allow_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) → 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

```
class aiogram.types.user.User(*, id: int, is_bot: bool, first_name: str, last_name: str | None =
    None, username: str | None = None, language_code: str | None =
    None, is_premium: bool | None = None,
    added_to_attachment_menu: bool | None = None, can_join_groups:
    bool | None = None, can_read_all_group_messages: bool | None =
    None, supports_inline_queries: bool | None = None,
    can_connect_to_business: bool | None = None, **extra_data: Any)
```

This object represents a Telegram user or bot.

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

id: int

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

is_bot: bool

True, if this user is a bot

first_name: str

User's or bot's first name

last_name: str | None

Optional. User's or bot's last name

username: str | None

Optional. User's or bot's username

language_code: str | None

Optional. IETF language tag of the user's language

is_premium: bool | None

Optional. True, if this user is a Telegram Premium user

added_to_attachment_menu: bool | None

Optional. True, if this user added the bot to the attachment menu

can_join_groups: bool | None

Optional. True, if the bot can be invited to groups. Returned only in *aiogram.methods.get_me.GetMe*.

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) → str`

`mention_html(name: str | None = None) → str`

`get_profile_photos(offset: int | None = None, limit: int | None = None, **kwargs: Any) → GetUserProfilePhotos`

Shortcut for method *aiogram.methods.get_user_profile_photos.GetUserProfilePhotos* will automatically fill method attributes:

- `user_id`

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

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

Параметри

- `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.

Повертає

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

```
class aiogram.types.user_profile_photos.UserProfilePhotos(*, total_count: int, photos:
    List[List[PhotoSize]], **extra_data:
    Any)
```

This object represent a user's profile pictures.

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

total_count: int

Total number of profile pictures the target user has

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.

Застаріло починаючи з версії 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

```
class aiogram.types.users_shared.UsersShared(*, request_id: int, users: List[SharedUser], user_ids: List[int] | None = None, **extra_data: Any)
```

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) → 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.

Застаріло починаючи з версії API:7.2: <https://core.telegram.org/bots/api-changelog#march-31-2024>

Venue

```
class aiogram.types.venue.Venue(*, location: Location, title: str, address: str, foursquare_id: str | None = None, foursquare_type: str | None = None, google_place_id: str | None = None, google_place_type: str | None = None, **extra_data: Any)
```

This object represents a venue.

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

`location: Location`

Venue location. Can't be a live location

`title: str`

Name of the venue

`address: str`

Address of the venue

`foursquare_id: str | None`

Optional. Foursquare identifier of the venue

`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

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

This object represents a video file.

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

`file_id: str`

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

`file_unique_id: str`

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

`width: int`

Video width as defined by sender

`height: int`

Video height as defined by sender

`duration: int`

Duration of the video in seconds as defined by sender

`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: 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

```
class aiogram.types.video_chat_participants_invited.VideoChatParticipantsInvited(*, users:
                                                                                     List[User],
                                                                                     **extra_data:
                                                                                     Any)
```

This object represents a service message about new members invited to a video chat.

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

users: List[User]

New members that were invited to the video chat

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.

VideoChatScheduled

```
class aiogram.types.video_chat_scheduled.VideoChatScheduled(*, start_date: datetime,
                                                             **extra_data: Any)
```

This object represents a service message about a video chat scheduled in the chat.

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

start_date: DateTime

Point in time (Unix timestamp) when the video chat is supposed to be started by a chat administrator

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

```
class aiogram.types.video_note.VideoNote(*, file_id: str, file_unique_id: str, length: int, duration:
    int, thumbnail: PhotoSize | None = None, file_size: int |
    None = None, **extra_data: Any)
```

This object represents a [video message](#) (available in Telegram apps as of [v.4.0](#)).

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

```
file_id: str
```

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

```
file_unique_id: str
```

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

```
length: int
```

Video width and height (diameter of the video message) as defined by sender

```
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
```

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) → 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) → 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

```
class aiogram.types.write_access_allowed.WriteAccessAllowed(*, from_request: bool | None =
    None, web_app_name: str | None = None, from_attachment_menu:
    bool | None = None, **extra_data: Any)
```

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

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

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

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

id: `str`

Unique identifier for this query

from_user: `User`

Sender

query: `str`

Text of the query (up to 256 characters)

`offset: str`

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

`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) → 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>

Параметри

- **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`

Повертає

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) → 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: Literal[InlineQueryResultType.ARTICLE],
                               id: str, title: str,
                               input_message_content: Union[InputTextMessageContent, InputLocationMessageContent,
                                                             InputVenueMessageContent, InputContactMessageContent,
                                                             InputInvoiceMessageContent],
                               reply_markup: Union[InlineKeyboardMarkup, None],
                               url: str | None = None,
                               hide_url: bool | None = None,
                               description: str | None = None,
                               thumbnail_url: str | None = None,
                               thumbnail_width: int | None = None,
                               thumbnail_height: int | None = None,
                               **extra_data: 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: `Union[InputTextMessageContent, InputLocationMessageContent, InputVenueMessageContent, InputContactMessageContent, InputInvoiceMessageContent]`

Content of the message to be sent

reply_markup: `Union[InlineKeyboardMarkup, None]`

Optional. Inline keyboard attached to the message

url: `str | None`

Optional. URL 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.

```
hide_url: bool | None
```

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

```
description: str | None
```

Optional. Short description of the result

```
thumbnail_url: str | None
```

Optional. Url of the thumbnail for the result

```
thumbnail_width: int | None
```

Optional. Thumbnail width

```
thumbnail_height: int | None
```

Optional. Thumbnail height

InlineQueryResultAudio

```

class aiogram.types.inline_query_result_audio.InlineQueryResultAudio(*, type: ~typing.Literal[InlineQueryResultType.AUDIO]
    = InlineQueryResultType.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: ~typing.List[~aiogram.types.message_entity.MessageEntity]
    | None = None,
    performer: str | None =
    None, audio_duration:
    int | None = None,
    reply_markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup
    | None = None,
    input_message_content:
    ~aiogram.types.input_text_message_content.InputTextMessageContent
    | ~aiogram.types.input_location_message_content.InputLocationMessageContent
    | ~aiogram.types.input_venue_message_content.InputVenueMessageContent
    | ~aiogram.types.input_contact_message_content.InputContactMessageContent
    | ~aiogram.types.input_invoice_message_content.InputInvoiceMessageContent
    | None = None,
    **extra_data:
    ~typing.Any)

```

Represents a link to 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.AUDIO]`

Type of the result, must be *audio*

id: `str`

Unique identifier for this result, 1-64 bytes

audio_url: `str`

A valid URL for the audio file

title: `str`

Title

caption: `str | None`

Optional. Caption, 0-1024 characters after entities parsing

parse_mode: `str | 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[InlineQueryResultCachedAudioType.AUDIO],
    id: str,
    audio_file_id: str,
    caption: str | None = None,
    parse_mode: str | ~aiogram.client.default.DefaultParseMode | None = <Default('parse_mode')>,
    caption_entities: ~typing.List[~aiogram.types.message_entities.MessageEntity] | None = None,
    reply_markup: ~aiogram.types.inline_keyboard.InlineKeyboardMarkup | None = None,
    input_message_content: ~aiogram.types.input_text.TextInputMessageContent | ~aiogram.types.input_location.LocationInputMessageContent | ~aiogram.types.input_venue.VenueInputMessageContent | ~aiogram.types.input_contact.ContactInputMessageContent | ~aiogram.types.input_invoice.InvoiceInputMessageContent | 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.AUDIO]`

Type of the result, must be *audio*

id: `str`

Unique identifier for this result, 1-64 bytes

`audio_file_id: str`

A valid file identifier for the audio file

`caption: str | None`

Optional. Caption, 0-1024 characters after entities parsing

`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 audio caption. See [formatting options](#) for more details.

`caption_entities: List[MessageEntity] | None`

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

`reply_markup: InlineKeyboardMarkup | None`

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

`input_message_content: InputTextMessageContent | InputLocationMessageContent |
InputVenueMessageContent | InputContactMessageContent | InputInvoiceMessageContent
| None`

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

InlineQueryResultCachedDocument

```

class aiogram.types.inline_query_result_cached_document.InlineQueryResultCachedDocument(*,
                                                                                         type:
                                                                                         ~typi-
                                                                                         ng.Literal[InlineQ
                                                                                         =
                                                                                         Inli-
                                                                                         neQueryResultTyp
                                                                                         id:
                                                                                         str,
                                                                                         ti-
                                                                                         tle:
                                                                                         str,
                                                                                         document_file_id:
                                                                                         str,
                                                                                         descri-
                                                                                         pti-
                                                                                         on:
                                                                                         str
                                                                                         /
                                                                                         None
                                                                                         =
                                                                                         None,
                                                                                         capti-
                                                                                         on:
                                                                                         str
                                                                                         /
                                                                                         None
                                                                                         =
                                                                                         None,
                                                                                         parse_mode:
                                                                                         str
                                                                                         /
                                                                                         ~ai-
                                                                                         ogram.client.defau
                                                                                         /
                                                                                         None
                                                                                         =
                                                                                         <Default('parse_r
                                                                                         capti-
                                                                                         on_entities:
                                                                                         ~typi-
                                                                                         ng.List[~aiogram.t
                                                                                         /
                                                                                         None
                                                                                         =
                                                                                         None,
                                                                                         reply_markup:
                                                                                         ~ai-
                                                                                         ogram.types.inline_
                                                                                         /
                                                                                         None
                                                                                         =
                                                                                         None,
                                                                                         input_message_co
                                                                                         ~ai-
                                                                                         ogram.types.input_
                                                                                         ~ai-
                                                                                         ogram.types.input_
                                                                                         /

```

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


```

class aiogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif(*, type:
    ~typing.Literal[InlineQueryResultType.GIF,
    = InlineQueryResultType.GIF,
    id: str,
    gif_file_id:
    str, title: str /
    None =
    None,
    caption: str /
    None =
    None,
    parse_mode:
    str / ~aiogram.client.default.Default
    / None =
    <Default('parse_mode')>,
    caption_entities:
    ~typing.List[~aiogram.types.message
    / None =
    None,
    reply_markup:
    ~aiogram.types.inline_keyboard_markup
    / None =
    None,
    input_message_content:
    ~aiogram.types.input_text_message_content
    / ~aiogram.types.input_location_message_content
    / ~aiogram.types.input_venue_message_content
    / ~aiogram.types.input_contact_message_content
    / ~aiogram.types.input_invoice_message_content
    / None =
    None,
    **extra_data:
    ~typing.Any)

```

Represents a link to 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) → 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_query_result_cached_mpeg4_gif.InlineQueryResultCachedMpeg4Gif( *,
                                                    type:
                                                    ~typi-
                                                    ng.Literal[InlineQ
                                                    =
                                                    Inli-
                                                    neQueryResultTy
                                                    id:
                                                    str,
                                                    mpeg4_file_id:
                                                    str,
                                                    ti-
                                                    tle:
                                                    str
                                                    /
                                                    None
                                                    =
                                                    None,
                                                    capti-
                                                    on:
                                                    str
                                                    /
                                                    None
                                                    =
                                                    None,
                                                    parse_mode:
                                                    str
                                                    /
                                                    ~ai-
                                                    ogram.client.defa
                                                    /
                                                    None
                                                    =
                                                    <Default('parse_
                                                    capti-
                                                    on_entities:
                                                    ~typi-
                                                    ng.List[~aiogram.
                                                    /
                                                    None
                                                    =
                                                    None,
                                                    reply_markup:
                                                    ~ai-
                                                    ogram.types.inlin
                                                    /
                                                    None
                                                    =
                                                    None,
                                                    input_message_c
                                                    ~ai-
                                                    ogram.types.inpu
                                                    /
                                                    ~ai-
                                                    ogram.types.inpu
                                                    /
                                                    ~ai-
                                                    ogram.types.inpu

```

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:
    ~typing.Literal[InlineQueryResultCachedPhoto] = InlineQueryResultType.PHOTO,
    id: str,
    photo_file_id: str, title: str / None = None,
    description: str / None = None,
    caption: str / None = None,
    parse_mode: str | ~aiogram.client.default.DefaultParseMode | None = <Default('parse_mode')>,
    caption_entities: ~typing.List[~aiogram.types.message_entity] / None = None,
    reply_markup: ~aiogram.types.inline_keyboard_markup / None = None,
    input_message_content: ~aiogram.types.input_text_message_content | ~aiogram.types.input_location_message_content | ~aiogram.types.input_venue_message_content | ~aiogram.types.input_contact_message_content | ~aiogram.types.input_invoice_message_content / None = None,
    **extra_data: ~typing.Any)

```

Represents a link to a photo stored on the Telegram servers. By default, this photo will be sent by the user with an optional caption. Alternatively, you can use `input_message_content` to send a message

with the specified content instead of the photo.

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

type: `Literal[InlineQueryResultType.PHOTO]`

Type of the result, must be *photo*

id: `str`

Unique identifier for this result, 1-64 bytes

photo_file_id: `str`

A valid file identifier of the photo

title: `str` | `None`

Optional. Title for the result

description: `str` | `None`

Optional. Short description of the result

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

```
class aiogram.types.inline_query_result_cached_sticker.InlineQueryResultCachedSticker(*,
                                                                                       type:
                                                                                       Li-
                                                                                       teral[InlineQueryRes
                                                                                       =
                                                                                       Inli-
                                                                                       neQueryResultType.STICKER]
                                                                                       id:
                                                                                       str,
                                                                                       sti-
                                                                                       cker_file_id:
                                                                                       str,
                                                                                       reply_markup:
                                                                                       Inli-
                                                                                       neKeyboardMarkup
                                                                                       /
                                                                                       None
                                                                                       =
                                                                                       None,
                                                                                       input_message_content:
                                                                                       InputTextMessageContent
                                                                                       /
                                                                                       InputLocationMessageContent
                                                                                       /
                                                                                       InputVenueMessageContent
                                                                                       /
                                                                                       InputContactMessageContent
                                                                                       /
                                                                                       InputInvoiceMessageContent
                                                                                       /
                                                                                       None
                                                                                       =
                                                                                       None,
                                                                                       **extra_data:
                                                                                       Any)
    """
```

Represents a link to a sticker stored on the Telegram servers. By default, this sticker will be sent by the user. Alternatively, you can use *input_message_content* to send a message with the specified content instead of the sticker.

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:
    ~typing.Literal[InlineQueryResultCachedVideo] = InlineQueryResultType.VIDEO,
    id: str,
    video_file_id: str, title: str,
    description: str | None = None,
    caption: str | None = None,
    parse_mode: str | ~aiogram.client.default.DefaultParseMode = <Default('parse_mode')>,
    caption_entities: ~typing.List[~aiogram.types.message_entity] | None = None,
    reply_markup: ~aiogram.types.inline_keyboard_markup | None = None,
    input_message_content: ~aiogram.types.input_message_content | ~aiogram.types.input_text_message_content | ~aiogram.types.input_location_message_content | ~aiogram.types.input_venue_message_content | ~aiogram.types.input_contact_message_content | ~aiogram.types.input_invoice_message_content | None = None,
    **extra_data: ~typing.Any)

```

Represents a link to a video file stored on the Telegram servers. By default, this video file will be sent by the user with an optional caption. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the video.

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

type: `Literal[InlineQueryResultType.VIDEO]`

Type of the result, must be *video*

id: `str`

Unique identifier for this result, 1-64 bytes

video_file_id: `str`

A valid file identifier for the video file

title: `str`

Title for the result

description: `str | None`

Optional. Short description of the result

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 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[InlineQueryResultCachedVoiceType.VOICE],
    id: str,
    voice_file_id: str, title: str,
    caption: str | None = None,
    parse_mode: str | ~aiogram.client.default.DefaultParseMode | None = <Default('parse_mode')>,
    caption_entities: ~typing.List[~aiogram.types.message_entities.MessageEntity] | None = None,
    reply_markup: ~aiogram.types.inline_keyboard.InlineKeyboardMarkup | None = None,
    input_message_content: ~aiogram.types.input_text.TextInputMessageContent | ~aiogram.types.input_location.LocationInputMessageContent | ~aiogram.types.input_venue.VenueInputMessageContent | ~aiogram.types.input_contact.ContactInputMessageContent | ~aiogram.types.input_invoice.InvoiceInputMessageContent | 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) → 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: Literal[InlineQueryResultType.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: InputTextMessageContent | InputLocationMessageContent | InputVenueMessageContent | InputContactMessageContent | InputInvoiceMessageContent | None = None,
                                thumbnail_url: str | None = None,
                                thumbnail_width: int | None = None,
                                thumbnail_height: int | 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.DOCUMENT] = InlineQueryResultType.DOCUMENT, id: str, title: str, document_url: str, mime_type: str, 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, description: str | None = None, reply_markup: ~aiogram.types.inline_keyboard_markup | None = None, input_message_content: ~aiogram.types.input_text_message_content | ~aiogram.types.input_location_message_content | ~aiogram.types.input_venue_message_content | ~aiogram.types.input_contact_message_content | ~aiogram.types.input_invoice_message_content | None = None, thumbnail_url: str | None = None, thumbnail_width: int | None = None, thumbnail_height: int | None = None, **extra_data: ~typing.Any)

```

Represents a link to a file. By default, this file will be sent by the user with an optional caption. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the file. Currently, only **.PDF** and **.ZIP** files can be sent using this method.

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

type: `Literal[InlineQueryResultType.DOCUMENT]`

Type of the result, must be *document*

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

title: str
Title for the result

document_url: str
A valid URL for the file

mime_type: str
MIME type of the content of the file, either „application/pdf“ or „application/zip“

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

parse_mode: str | 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) → 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: Literal[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*) → `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

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

Represents a link to an animated GIF file. By default, this animated GIF file will be sent by the user with optional caption. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the animation.

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

type: `Literal[InlineQueryResultType.GIF]`

Type of the result, must be *gif*

id: `str`

Unique identifier for this result, 1-64 bytes

gif_url: `str`

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

thumbnail_url: `str`

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

gif_width: `int | None`

Optional. Width of the GIF

`gif_height: int | None`

Optional. Height of the GIF

`gif_duration: int | None`

Optional. Duration of the GIF in seconds

`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: Literal[InlineQueryResultType.LOCATION],
    = InlineQueryResultType.LOCATION,
    id: str, latitude: float, longitude: float, title: str,
    horizontal_accuracy: float | None = None,
    live_period: int | None = None,
    heading: int | None = None,
    proximity_alert_radius: int | None = None,
    reply_markup: InlineKeyboardMarkup | None = None,
    input_message_content: InputTextMessageContent | InputLocationMessageContent | InputVenueMessageContent | InputContactMessageContent | InputInvoiceMessageContent | None = None,
    thumbnail_url: str | None = None,
    thumbnail_width: int | None = None,
    thumbnail_height: int | 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 during which the location can be updated, should be between 60 and 86400, or 0x7FFFFFFF for live locations that can be edited indefinitely.

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

Represents a link to a video animation (H.264/MPEG-4 AVC video without sound). By default, this animated MPEG-4 file will be sent by the user with optional caption. Alternatively, you can use *input_message_content* to send a message with the specified content instead of the animation.

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

type: `Literal[InlineQueryResultType.MPEG4_GIF]`

Type of the result, must be *mpeg4_gif*

id: `str`

Unique identifier for this result, 1-64 bytes

mpeg4_url: `str`

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

thumbnail_url: `str`

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

mpeg4_width: `int | None`

Optional. Video width

mpeg4_height: `int | None`

Optional. Video height

mpeg4_duration: `int | None`

Optional. Video duration in seconds

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

Represents a link to a photo. By default, this photo will be sent by the user with optional caption. Alternatively, you can use *input_message_content* to send a message with the specified content instead of the photo.

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

type: `Literal[InlineQueryResultType.PHOTO]`

Type of the result, must be *photo*

id: `str`

Unique identifier for this result, 1-64 bytes

photo_url: `str`

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

`thumbnail_url: str`
 URL of the thumbnail for the photo

`photo_width: int | None`
Optional. Width of the photo

`photo_height: int | None`
Optional. Height of the photo

`title: str | None`
Optional. Title for the result

`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 | 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: Literal[InlineQueryResultType.VENUE]
                                                                    = InlineQueryResultType.VENUE,
                                                                    id: str, latitude: float,
                                                                    longitude: float, title:
                                                                    str, address: str,
                                                                    foursquare_id: str /
                                                                    None = None,
                                                                    foursquare_type: str /
                                                                    None = None,
                                                                    google_place_id: str /
                                                                    None = None,
                                                                    google_place_type: str /
                                                                    None = None,
                                                                    reply_markup:
                                                                    InlineKeyboardMarkup
                                                                    / None = None,
                                                                    input_message_content:
                                                                    InputTextMessageContent
                                                                    / InputLocationMessageContent
                                                                    / InputVenueMessageContent
                                                                    /
                                                                    InputContactMessageContent
                                                                    / InputInvoiceMessageContent /
                                                                    None = None,
                                                                    thumbnail_url: str /
                                                                    None = None,
                                                                    thumbnail_width: int /
                                                                    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 `input_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) → 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: ~typing.Literal[InlineQueryResultType.VIDEO] = InlineQueryResultType.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.MessageEntity] | None = None, video_width: int | None = None, video_height: int | None = None, video_duration: int | None = None, description: str | None = None, reply_markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup | None = None, input_message_content: ~aiogram.types.input_text_message_content.InputTextMessageContent | ~aiogram.types.input_location_message_content.InputLocationMessageContent | ~aiogram.types.input_venue_message_content.InputVenueMessageContent | ~aiogram.types.input_contact_message_content.InputContactMessageContent | ~aiogram.types.input_invoice_message_content.InputInvoiceMessageContent | None = None, **extra_data: ~typing.Any)
```

Represents a link to a page containing an embedded video player or a video file. By default, this video file will be sent by the user with an optional caption. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the video.

If an `InlineQueryResultVideo` message contains an embedded video (e.g., YouTube), you **must** replace its content using `input_message_content`.

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

`type: Literal[InlineQueryResultType.VIDEO]`

Type of the result, must be *video*

`id: str`

Unique identifier for this result, 1-64 bytes

`video_url: str`
 A valid URL for the embedded video player or video file

`mime_type: str`
 MIME type of the content of the video URL, „text/html“ or „video/mp4“

`thumbnail_url: str`
 URL of the thumbnail (JPEG only) for the video

`title: str`
 Title for the result

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

`parse_mode: str | 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) → 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_width: int | None`
Optional. Video width

`video_height: int | None`
Optional. Video height

`video_duration: int | None`
Optional. Video duration in seconds

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

`reply_markup: InlineKeyboardMarkup | None`
Optional. [Inline keyboard](#) attached to the message

`input_message_content: InputTextMessageContent | InputLocationMessageContent | InputVenueMessageContent | InputContactMessageContent | InputInvoiceMessageContent | None`
Optional. Content of the message to be sent instead of the video. This field is **required** if `InlineQueryResultVideo` is used to send an HTML-page as a result (e.g., a YouTube video).

InlineQueryResultVoice

```
class aiogram.types.inline_query_result_voice.InlineQueryResultVoice(*, type: ~typing.Literal[InlineQueryResultType.VOICE],
                             = InlineQueryResultType.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: ~typing.List[~aiogram.types.message_entity.MessageEntity] | None = None,
                             voice_duration: int | None = None,
                             reply_markup: ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup | None = None,
                             input_message_content: ~aiogram.types.input_text_message_content.InputTextMessageContent
                             | ~aiogram.types.input_location_message_content.InputLocationMessageContent
                             | ~aiogram.types.input_venue_message_content.InputVenueMessageContent
                             | ~aiogram.types.input_contact_message_content.InputContactMessageContent
                             | ~aiogram.types.input_invoice_message_content.InputInvoiceMessageContent
                             | None = None,
                             **extra_data: ~typing.Any)
```

Represents a link to a voice recording in an .OGG container encoded with OPUS. By default, this voice recording will be sent by the user. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the the voice message.

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

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

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

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

```
text: str
```

Label text on the button

```
web_app: WebAppInfo | None
```

Optional. Description of the [Web App](#) that will be launched when the user presses the button. The Web App will be able to switch back to the inline mode using the method [switchInlineQuery](#) inside the Web App.

```
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.

`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, provi-
                                                                            der_token:
                                                                            str, currency:
                                                                            str, prices: Li-
                                                                            st[LabeledPrice],
                                                                            max_tip_amount:
                                                                            int / None =
                                                                            None,
                                                                            suggested_tip_amounts:
                                                                            List[int] /
                                                                            None = None,
                                                                            provider_data:
                                                                            str / None =
                                                                            None,
                                                                            photo_url: str
                                                                            / None =
                                                                            None,
                                                                            photo_size:
                                                                            int / None =
                                                                            None,
                                                                            photo_width:
                                                                            int / None =
                                                                            None,
                                                                            photo_height:
                                                                            int / None =
                                                                            None,
                                                                            need_name:
                                                                            bool / None =
                                                                            None,
                                                                            need_phone_number:
                                                                            bool / None =
                                                                            None,
                                                                            need_email:
                                                                            bool / None =
                                                                            None,
                                                                            need_shipping_address:
                                                                            bool / None =
                                                                            None,
                                                                            send_phone_number_to_provider:
                                                                            bool / None =
                                                                            None,
                                                                            send_email_to_provider:
                                                                            bool / None =
                                                                            None,
                                                                            is_flexible:
                                                                            bool / None =
                                                                            None,
                                                                            **extra_data:
                                                                            Any)

```

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

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

title: `str`

Product name, 1-32 characters

description: `str`

Product description, 1-255 characters

payload: `str`

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

provider_token: `str`

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

currency: `str`

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

prices: `List[LabeledPrice]`

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

max_tip_amount: `int | None`

Optional. The maximum accepted amount for tips in the *smallest units* of the currency (integer, **not** float/double). For example, for a maximum tip of US\$ 1.45 pass `max_tip_amount = 145`. See the *exp* parameter in [currencies.json](#), it shows the number of digits past the decimal point for each currency (2 for the majority of currencies). Defaults to 0

suggested_tip_amounts: `List[int] | None`

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

provider_data: `str | None`

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

photo_url: `str | None`

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

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_size: `int | None`

Optional. Photo size in bytes

photo_width: `int | None`

Optional. Photo width

photo_height: `int | None`

Optional. Photo height

`need_name: bool | None`

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

`need_phone_number: bool | None`

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

`need_email: bool | None`

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

`need_shipping_address: bool | None`

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

`send_phone_number_to_provider: bool | None`

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

`send_email_to_provider: bool | None`

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

`is_flexible: bool | None`

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

InputLocationMessageContent

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

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

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

`latitude: float`

Latitude of the location in degrees

`longitude: float`

Longitude of the location in degrees

`horizontal_accuracy: float | None`

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

`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 during which the location can be updated, should be between 60 and 86400, or 0x7FFFFFFF for live locations that can be edited indefinitely.

`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.

`model_post_init(_ModelMetaclass__context: Any) → None`

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 =
                                                                    <Default('parse_mode')>,
                                                                    entities: ~typing.List[~aiogram.types.message_entity
                                                                    / None = None, link_preview_options:
                                                                    ~aiogram.types.link_preview_options.LinkPreviewOptions
                                                                    / None = None, disable_web_page_preview:
                                                                    bool | ~aiogram.client.default.Default
                                                                    / None =
                                                                    <Default('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

Застаріло починаючи з версії 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) → 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

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

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

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

inline_message_id: str | None

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

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.

Stickers

InputSticker

```
class aiogram.types.input_sticker.InputSticker(*, sticker: InputFile | str, format: str, emoji_list:
                                              List[str], mask_position: MaskPosition | None =
                                              None, keywords: List[str] | None = None,
                                              **extra_data: Any)
```

This object describes a sticker to be added to a sticker set.

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

sticker: *InputFile* | str

The added sticker. Pass a *file_id* as a String to send a file that already exists on the Telegram servers, pass an HTTP URL as a String for Telegram to get a file from the Internet, upload a new one using multipart/form-data, or pass „attach://<file_attach_name>“ to upload a new one using multipart/form-data under <file_attach_name> name. Animated and video stickers can't be uploaded via HTTP URL. [More information on Sending Files](#) »

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

```
class aiogram.types.mask_position.MaskPosition(*, point: str, x_shift: float, y_shift: float, scale: float, **extra_data: Any)
```

This object describes the position on faces where a mask should be placed by default.

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

point: str

The part of the face relative to which the mask should be placed. One of „forehead“, „eyes“, „mouth“, or „chin“.

x_shift: float

Shift by X-axis measured in widths of the mask scaled to the face size, from left to right. For example, choosing -1.0 will place mask just to the left of the default mask position.

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.

y_shift: float

Shift by Y-axis measured in heights of the mask scaled to the face size, from top to bottom. For example, 1.0 will place the mask just below the default mask position.

scale: float

Mask scaling coefficient. For example, 2.0 means double size.

Sticker

```
class aiogram.types.sticker.Sticker(*, file_id: str, file_unique_id: str, type: str, width: int, height: int, is_animated: bool, is_video: bool, thumbnail: PhotoSize | None = None, emoji: str | None = None, set_name: str | None = None, premium_animation: File | None = None, mask_position: MaskPosition | None = None, custom_emoji_id: str | None = None, needs_repainting: bool | None = None, file_size: int | None = None, **extra_data: Any)
```

This object represents a sticker.

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

file_id: str

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

file_unique_id: str

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

type: str

Type of the sticker, currently one of „regular“, „mask“, „custom_emoji“. The type of the sticker is independent from its format, which is determined by the fields *is_animated* and *is_video*.

`width: int`
 Sticker width

`height: int`
 Sticker height

`is_animated: bool`
 True, if the sticker is *animated*

`is_video: bool`
 True, if the sticker is a *video sticker*

`thumbnail: PhotoSize | None`
Optional. Sticker thumbnail in the .WEBP or .JPG format

`emoji: str | None`
Optional. Emoji associated with the sticker

`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) → SetStickerPositionInSet`
 Shortcut for method *aiogram.methods.set_sticker_position_in_set*.
SetStickerPositionInSet will automatically fill method attributes:

- `sticker`

 Use this method to move a sticker in a set created by the bot to a specific position. Returns `True` on success.

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

Параметри
`position` – New sticker position in the set, zero-based

Повертає

instance of method `aiogram.methods.set_sticker_position_in_set.
SetStickerPositionInSet`

`delete_from_set(**kwargs: Any) → DeleteStickerFromSet`

Shortcut for method `aiogram.methods.delete_sticker_from_set.DeleteStickerFromSet` will automatically fill method attributes:

- `sticker`

Use this method to delete a sticker from a set created by the bot. Returns `True` on success.

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

Повертає

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) → 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

Застаріло починаючи з версії 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

Застаріло починаючи з версії API:7.2: <https://core.telegram.org/bots/api-changelog#march-31-2024>

Telegram Passport

EncryptedCredentials

```
class aiogram.types.encrypted_credentials.EncryptedCredentials(*, data: str, hash: str, secret: str, **extra_data: Any)
```

Describes data required for decrypting and authenticating `aiogram.types.encrypted_passport_element.EncryptedPassportElement`. See the Telegram Passport Documentation for a complete description of the data decryption and authentication processes.

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

data: str

Base64-encoded encrypted JSON-serialized data with unique user's payload, data hashes and secrets required for `aiogram.types.encrypted_passport_element.EncryptedPassportElement` decryption and authentication

hash: str

Base64-encoded data hash for data authentication

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.

secret: str

Base64-encoded secret, encrypted with the bot's public RSA key, required for data decryption

EncryptedPassportElement

```
class aiogram.types.encrypted_passport_element.EncryptedPassportElement(*, type: str, hash:
    str, data: str | None
    = None,
    phone_number: str |
    None = None,
    email: str | None =
    None, files:
    List[PassportFile] |
    None = None,
    front_side:
    PassportFile | None
    = None,
    reverse_side:
    PassportFile | None
    = None, selfie:
    PassportFile | None
    = None, translation:
    List[PassportFile] |
    None = None,
    **extra_data: Any)
```

Describes documents or other Telegram Passport elements shared with the bot by the user.

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

type: str

Element type. One of „personal_details“, „passport“, „driver_license“, „identity_card“, „internal_passport“, „address“, „utility_bill“, „bank_statement“, „rental_agreement“, „passport_registration“, „temporary_registration“, „phone_number“, „email“.

hash: str

Base64-encoded element hash for using in *aiogram.types.encrypted_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) → 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

```
class aiogram.types.passport_data.PassportData(*, data: List[EncryptedPassportElement],
                                              credentials: EncryptedCredentials, **extra_data:
                                              Any)
```

Describes Telegram Passport data shared with the bot by the user.

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

data: List[*EncryptedPassportElement*]

Array with information about documents and other Telegram Passport elements that was shared with the bot

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.

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

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:
                                                                                      Li-
                                                                                      teral[PassportElementE
                                                                                      =
                                                                                      PassportElementErrorT
                                                                                      type:
                                                                                      str, fi-
                                                                                      eld_name:
                                                                                      str,
                                                                                      data_hash:
                                                                                      str,
                                                                                      message:
                                                                                      str,
                                                                                      **extra_data:
                                                                                      Any)
```

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

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

```
source: Literal[PassportElementType.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: Li-
                                                                    teral[PassportElementType.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*) → 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: Li-
                                                                    teral[PassportElementType.FILES]
                                                                    =
                                                                    PassportElementType.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[PassportElementType.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*) → 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

```
class aiogram.types.passport_element_error_front_side.PassportElementErrorFrontSide(*,
                                                                    source:
                                                                    Li-
                                                                    teral[PassportElementE
                                                                    =
                                                                    PassportElementErrorT
                                                                    type:
                                                                    str, fi-
                                                                    le_hash:
                                                                    str,
                                                                    message:
                                                                    str,
                                                                    **extra_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[PassportElementType.FRONT_SIDE]`

Error source, must be *front_side*

`type: str`

The section of the user's Telegram Passport which has the issue, one of „passport“, „driver_license“, „identity_card“, „internal_passport“

`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 front side of the document

`message: str`

Error message

PassportElementErrorReverseSide

```
class aiogram.types.passport_element_error_reverse_side.PassportElementErrorReverseSide(*,
                                                                                       source:
                                                                                       Li-
                                                                                       teral[PassportElem
                                                                                       =
                                                                                       PassportElementE
                                                                                       type:
                                                                                       str,
                                                                                       fi-
                                                                                       le_hash:
                                                                                       str,
                                                                                       message:
                                                                                       str,
                                                                                       **extra_data:
                                                                                       Any)
```

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

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

`source: Literal[PassportElementType.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) → 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: Literal[PassportElementType.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[PassportElementType.SELFIE]`
Error source, must be *selfie*

`type: str`
The section of the user's Telegram Passport which has the issue, one of „passport“, „driver_license“, „identity_card“, „internal_passport“

`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:
                                                                                               Li-
                                                                                               teral[Pas-
                                                                                               =
                                                                                               Passport
                                                                                               type:
                                                                                               str,
                                                                                               fi-
                                                                                               le_hash:
                                                                                               str,
                                                                                               message:
                                                                                               str,
                                                                                               **extra_
                                                                                               Any)
```

Represents an issue with one of the files that constitute the translation of a document. The error is considered resolved when the file changes.

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

source: `Literal[PassportElementType.TRANSLATION_FILE]`

Error source, must be *translation_file*

type: `str`

Type of element of the user's Telegram Passport which has the issue, one of „passport“, „driver_license“, „identity_card“, „internal_passport“, „utility_bill“, „bank_statement“, „rental_agreement“, „passport_registration“, „temporary_registration“

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(*,
                                                                                               source: Literal[PassportElementType.TRANSLATION_FILES] =
                                                                                               None,
                                                                                               type: str,
                                                                                               file_hash: str,
                                                                                               list_of_files: List[str],
                                                                                               message: str,
                                                                                               **kwargs: Any)
    """
```

Represents an issue with the translated version of a document. The error is considered resolved when a file with the document translation change.

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

source: `Literal[PassportElementType.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*) → 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:
                                                                                      Li-
                                                                                      teral[PassportElementErrorType] =
                                                                                      PassportElementErrorUnspecified,
                                                                                      type:
                                                                                      str,
                                                                                      element_hash:
                                                                                      str,
                                                                                      message:
                                                                                      str,
                                                                                      **extra_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

```
class aiogram.types.passport_file.PassportFile(*, file_id: str, file_unique_id: str, file_size: int,
                                                file_date: datetime, **extra_data: Any)
```

This object represents a file uploaded to Telegram Passport. Currently all Telegram Passport files are in JPEG format when decrypted and don't exceed 10MB.

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

file_id: `str`

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

file_unique_id: `str`

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

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_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) → 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) → 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) → 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

```
class aiogram.types.pre_checkout_query.PreCheckoutQuery(*, id: str, from_user: User, currency: str, total_amount: int, invoice_payload: str, shipping_option_id: str | None = None, order_info: OrderInfo | None = None, **extra_data: Any)
```

This object contains information about an incoming pre-checkout query.

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

```
id: str
```

Unique query identifier

```
from_user: User
```

User who sent the query

`currency: str`

Three-letter ISO 4217 [currency](#) code

`total_amount: int`

Total price in the *smallest units* of the currency (integer, **not** float/double). For example, for a price of US\$ 1.45 pass `amount = 145`. See the *exp* parameter in [currencies.json](#), it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).

`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.

`invoice_payload: str`

Bot specified invoice payload

`shipping_option_id: str | None`

Optional. Identifier of the shipping option chosen by the user

`order_info: OrderInfo | None`

Optional. Order information provided by the user

`answer(ok: bool, error_message: str | None = None, **kwargs: Any) → AnswerPreCheckoutQuery`

Shortcut for method [aiogram.methods.answer_pre_checkout_query](#). [AnswerPreCheckoutQuery](#) will automatically fill method attributes:

- `pre_checkout_query_id`

Once the user has confirmed their payment and shipping details, the Bot API sends the final confirmation in the form of an [aiogram.types.update.Update](#) with the field `pre_checkout_query`. Use this method to respond to such pre-checkout queries. On success, **True** is returned. **Note:** The Bot API must receive an answer within 10 seconds after the pre-checkout query was sent.

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

Параметри

- `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.

Повертає

instance of method [aiogram.methods.answer_pre_checkout_query](#). [AnswerPreCheckoutQuery](#)

ShippingAddress

```
class aiogram.types.shipping_address.ShippingAddress(*, country_code: str, state: str, city: str,
                                                    street_line1: str, street_line2: str,
                                                    post_code: str, **extra_data: Any)
```

This object represents a shipping address.

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

country_code: str

Two-letter ISO 3166-1 alpha-2 country code

state: str

State, if applicable

city: str

City

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.

street_line1: str

First line for the address

street_line2: str

Second line for the address

post_code: str

Address post code

ShippingOption

```
class aiogram.types.shipping_option.ShippingOption(*, id: str, title: str, prices:
                                                    List[LabeledPrice], **extra_data: Any)
```

This object represents one shipping option.

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

id: str

Shipping option identifier

title: str

Option title

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.

prices: List[LabeledPrice]

List of price portions

ShippingQuery

```
class aiogram.types.shipping_query.ShippingQuery(*, id: str, from_user: User, invoice_payload: str, shipping_address: ShippingAddress, **extra_data: Any)
```

This object contains information about an incoming shipping query.

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

id: `str`

Unique query identifier

from_user: `User`

User who sent the query

invoice_payload: `str`

Bot specified invoice payload

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>

Параметри

- **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.

Повертає

instance of method `aiogram.methods.answer_shipping_query.AnswerShippingQuery`

SuccessfulPayment

```
class aiogram.types.successful_payment.SuccessfulPayment(*, currency: str, total_amount: int,
                                                         invoice_payload: str,
                                                         telegram_payment_charge_id: str,
                                                         provider_payment_charge_id: str,
                                                         shipping_option_id: str | None =
                                                         None, order_info: OrderInfo | None =
                                                         None, **extra_data: Any)
```

This object contains basic information about a successful payment.

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

currency: str

Three-letter ISO 4217 [currency](#) code

total_amount: int

Total price in the *smallest units* of the currency (integer, **not** float/double). For example, for a price of US\$ 1.45 pass `amount = 145`. See the *exp* parameter in [currencies.json](#), it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).

invoice_payload: str

Bot specified invoice payload

telegram_payment_charge_id: str

Telegram payment identifier

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) → 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`

Поведінка

`property event: TelegramObject`

`exception aiogram.types.update.UpdateTypeLookupError`

Update does not contain any known event type.

WebhookInfo

```
class aiogram.types.webhook_info.WebhookInfo(*, url: str, has_custom_certificate: bool,
                                              pending_update_count: int, ip_address: str | None
                                              = None, last_error_date: datetime | None = None,
                                              last_error_message: str | None = None,
                                              last_synchronization_error_date: datetime | None
                                              = None, max_connections: int | None = None,
                                              allowed_updates: List[str] | None = None,
                                              **extra_data: Any)
```

Describes the current status of a webhook.

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

`url: str`

Webhook URL, may be empty if webhook is not set up

`has_custom_certificate: bool`

True, if a custom certificate was provided for webhook certificate checks

`pending_update_count: int`

Number of updates awaiting delivery

`ip_address: str | None`

Optional. Currently used webhook IP address

`last_error_date: DateTime | None`

Optional. Unix time for the most recent error that happened when trying to deliver an update via webhook

`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_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

```
class aiogram.types.game.Game(*, title: str, description: str, photo: List[PhotoSize], text: str | None = None, text_entities: List[MessageEntity] | None = None, animation: Animation | None = None, **extra_data: Any)
```

This object represents a game. Use `BotFather` to create and edit games, their short names will act as unique identifiers.

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

```
title: str
```

Title of the game

```
description: str
```

Description of the game

```
photo: List[PhotoSize]
```

Photo that will be displayed in the game message in chats.

```
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) → 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

```
class aiogram.methods.add_sticker_to_set.AddStickerToSet(*, user_id: int, name: str, sticker:
                                                         InputSticker, **extra_data: Any)
```

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

```
class aiogram.methods.create_new_sticker_set.CreateNewStickerSet(*, user_id: int, name: str,
                                                                title: str, stickers:
                                                                List[InputSticker],
                                                                sticker_type: str | None =
                                                                None, needs_repainting: bool
                                                                / None = None,
                                                                sticker_format: str | None =
                                                                None, **extra_data: Any)
```

Use this method to create a new sticker set owned by a user. The bot will be able to edit the sticker set thus created. Returns **True** on success.

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

user_id: int

User identifier of created sticker set owner

name: str

Short name of sticker set, to be used in `t.me/addstickers/` URLs (e.g., *animals*). Can contain only English letters, digits and underscores. Must begin with a letter, can't contain consecutive underscores and must end in `_"by_<bot_username>"`. `<bot_username>` is case insensitive. 1-64 characters.

title: str

Sticker set title, 1-64 characters

stickers: List[InputSticker]

A JSON-serialized list of 1-50 initial stickers to be added to the sticker set

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_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“

Застаріло починаючи з версії 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(...)
```

Method as object

Imports:

- from aiogram.methods.create_new_sticker_set import CreateNewStickerSet
- alias: from aiogram.methods import CreateNewStickerSet

With specific bot

```
result: bool = await bot(CreateNewStickerSet(...))
```

As reply into Webhook in handler

```
return CreateNewStickerSet(...)
```

deleteStickerFromSet

Returns: bool

```
class aiogram.methods.delete_sticker_from_set.DeleteStickerFromSet(*, sticker: str,
                                                                    **extra_data: Any)
```

Use this method to delete a sticker from a set created by the bot. Returns **True** on success.

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

sticker: str

File identifier of the sticker

```
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_from_set(...)
```

Method as object

Imports:

- `from aiogram.methods.delete_sticker_from_set import DeleteStickerFromSet`
- `alias: from aiogram.methods import DeleteStickerFromSet`

With specific bot

```
result: bool = await bot(DeleteStickerFromSet(...))
```

As reply into Webhook in handler

```
return DeleteStickerFromSet(...)
```

As shortcut from received object

- `aiogram.types.sticker.Sticker.delete_from_set()`

deleteStickerSet

Returns: bool

```
class aiogram.methods.delete_sticker_set.DeleteStickerSet(*, name: str, **extra_data: Any)
```

Use this method to delete a sticker set that was created by the bot. Returns `True` on success.

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

name: str

Sticker set name

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(...)
```

Method as object

Imports:

- from aiogram.methods.delete_sticker_set import DeleteStickerSet
- alias: from aiogram.methods import DeleteStickerSet

With specific bot

```
result: bool = await bot(DeleteStickerSet(...))
```

As reply into Webhook in handler

```
return DeleteStickerSet(...)
```

getCustomEmojiStickers

Returns: List[Sticker]

```
class aiogram.methods.get_custom_emoji_stickers.GetCustomEmojiStickers(*,
                                                                    custom_emoji_ids:
                                                                    List[str],
                                                                    **extra_data: Any)
```

Use this method to get information about custom emoji stickers by their identifiers. Returns an Array of *aiogram.types.sticker.Sticker* objects.

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

custom_emoji_ids: List[str]

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(...)
```

Method as object

Imports:

- from aiogram.methods.get_custom_emoji_stickers import GetCustomEmojiStickers
- alias: from aiogram.methods import GetCustomEmojiStickers

With specific bot

```
result: List[Sticker] = await bot(GetCustomEmojiStickers(...))
```

getStickerSet

Returns: StickerSet

```
class aiogram.methods.get_sticker_set.GetStickerSet(*, name: str, **extra_data: Any)
```

Use this method to get a sticker set. On success, a *aiogram.types.sticker_set.StickerSet* object is returned.

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

name: str

Name of the sticker set

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: StickerSet = await bot.get_sticker_set(...)
```

Method as object

Imports:

- `from aiogram.methods.get_sticker_set import GetStickerSet`
- `alias: from aiogram.methods import GetStickerSet`

With specific bot

```
result: StickerSet = await bot(GetStickerSet(...))
```

replaceStickerInSet

Returns: bool

```
class aiogram.methods.replace_sticker_in_set.ReplaceStickerInSet(*, user_id: int, name: str,
                                                                old_sticker: str, sticker:
                                                                InputSticker, **extra_data:
                                                                Any)
```

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(...)
```

Method as object

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: ~ai-
    ogram.types.inline_keyboard_markup.InlineKeyboardMarkup
    | ~ai-
    ogram.types.reply_keyboard_markup.ReplyKeyboardMarkup
    | ~ai-
    ogram.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) → 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 a 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

Застаріло починаючи з версії 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

Застаріло починаючи з версії API:7.0: <https://core.telegram.org/bots/api-changelog#december-29-2023>

Usage

As bot method

```
result: Message = await bot.send_sticker(...)
```

Method as object

Imports:

- `from aiogram.methods.send_sticker import SendSticker`
- `alias: from aiogram.methods import SendSticker`

With specific bot

```
result: Message = await bot(SendSticker(...))
```

As reply into Webhook in handler

```
return SendSticker(...)
```

As shortcut from received object

- `aiogram.types.message.Message.answer_sticker()`
- `aiogram.types.message.Message.reply_sticker()`
- `aiogram.types.chat_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,
                                                                                             custom_e
                                                                                             str
                                                                                             /
                                                                                             None
                                                                                             =
                                                                                             None,
                                                                                             **extra_a
                                                                                             Any)
```

Use this method to set the thumbnail of a custom emoji sticker set. Returns `True` on success.

Source: <https://core.telegram.org/bots/api#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) → 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

```
class aiogram.methods.set_sticker_emoji_list.SetStickerEmojiList(*, sticker: str, emoji_list:
                                                                    List[str], **extra_data:
                                                                    Any)
```

Use this method to change the list of emoji assigned to a regular or custom emoji sticker. The sticker must belong to a sticker set created by the bot. Returns **True** on success.

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

sticker: str

File identifier of the sticker

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_list: List[str]

A JSON-serialized list of 1-20 emoji associated with the sticker

Usage

As bot method

```
result: bool = await bot.set_sticker_emoji_list(...)
```

Method as object

Imports:

- `from aiogram.methods.set_sticker_emoji_list import SetStickerEmojiList`
- `alias: from aiogram.methods import SetStickerEmojiList`

With specific bot

```
result: bool = await bot(SetStickerEmojiList(...))
```

As reply into Webhook in handler

```
return SetStickerEmojiList(...)
```

setStickerKeywords

Returns: bool

```
class aiogram.methods.set_sticker_keywords.SetStickerKeywords(*, sticker: str, keywords:
    List[str] | None = None,
    **extra_data: Any)
```

Use this method to change search keywords assigned to a regular or custom emoji sticker. The sticker must belong to a sticker set created by the bot. Returns **True** on success.

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

sticker: str

File identifier of the sticker

```
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

```
class aiogram.methods.set_sticker_position_in_set.SetStickerPositionInSet(*, sticker: str,
                                                                           position: int,
                                                                           **extra_data:
                                                                           Any)
```

Use this method to move a sticker in a set created by the bot to a specific position. Returns `True` on success.

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

sticker: str

File identifier of the sticker

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.

position: int

New sticker position in the set, zero-based

Usage

As bot method

```
result: bool = await bot.set_sticker_position_in_set(...)
```

Method as object

Imports:

- `from aiogram.methods.set_sticker_position_in_set import SetStickerPositionInSet`
- `alias: from aiogram.methods import SetStickerPositionInSet`

With specific bot

```
result: bool = await bot(SetStickerPositionInSet(...))
```

As reply into Webhook in handler

```
return SetStickerPositionInSet(...)
```

As shortcut from received object

- `aiogram.types.sticker.Sticker.set_position_in_set()`

setStickerSetThumbnail

Returns: bool

```
class aiogram.methods.set_sticker_set_thumbnail.SetStickerSetThumbnail(*, name: str,
                                                                    user_id: int, format:
                                                                    str, thumbnail:
                                                                    InputFile | str | None
                                                                    = None,
                                                                    **extra_data: Any)
```

Use this method to set the thumbnail of a regular or mask sticker set. The format of the thumbnail file must match the format of the stickers in the set. Returns **True** on success.

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

name: str

Sticker set name

user_id: int

User identifier of the sticker set owner

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#animated-sticker-requirements>), or a **WEBM** video with the thumbnail up to 32 kilobytes in size; see <https://core.telegram.org/stickers#video-sticker-requirements> for animated sticker technical requirements. Pass a *file_id* as a String to send a file that already exists on the Telegram servers, pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data. *More information on Sending Files »*. Animated and video sticker set thumbnails can't be uploaded via HTTP URL. If omitted, then the thumbnail is dropped and the first sticker is used as the thumbnail.

Usage

As bot method

```
result: bool = await bot.set_sticker_set_thumbnail(...)
```

Method as object

Imports:

- from aiogram.methods.set_sticker_set_thumbnail import SetStickerSetThumbnail
- alias: from aiogram.methods import SetStickerSetThumbnail

With specific bot

```
result: bool = await bot(SetStickerSetThumbnail(...))
```

As reply into Webhook in handler

```
return SetStickerSetThumbnail(...)
```

setStickerSetTitle

Returns: bool

```
class aiogram.methods.set_sticker_set_title.SetStickerSetTitle(*, name: str, title: str,
**extra_data: Any)
```

Use this method to set the title of a created sticker set. Returns True on success.

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

```
name: str
    Sticker set name

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
    Sticker set title, 1-64 characters
```

Usage

As bot method

```
result: bool = await bot.set_sticker_set_title(...)
```

Method as object

Imports:

- `from aiogram.methods.set_sticker_set_title import SetStickerSetTitle`
- `alias: from aiogram.methods import SetStickerSetTitle`

With specific bot

```
result: bool = await bot(SetStickerSetTitle(...))
```

As reply into Webhook in handler

```
return SetStickerSetTitle(...)
```

uploadStickerFile

Returns: `File`

```
class aiogram.methods.upload_sticker_file.UploadStickerFile(*, user_id: int, sticker: InputFile,
                                                            sticker_format: str, **extra_data:
                                                            Any)
```

Use this method to upload a file with a sticker for later use in the `aiogram.methods.create_new_sticker_set.CreateNewStickerSet`, `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*) → 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

Imports:

- `from aiogram.methods.upload_sticker_file import UploadStickerFile`
- `alias: from aiogram.methods import UploadStickerFile`

With specific bot

```
result: File = await bot(UploadStickerFile(...))
```

Available methods

answerCallbackQuery

Returns: bool

```
class aiogram.methods.answer_callback_query.AnswerCallbackQuery(*, callback_query_id: str,
                                                                text: str | None = None,
                                                                show_alert: bool | None =
                                                                None, url: str | None = None,
                                                                cache_time: int | None =
                                                                None, **extra_data: Any)
```

Use this method to send answers to callback queries sent from [inline keyboards](#). The answer will be displayed to the user as a notification at the top of the chat screen or as an alert. On success, `True` is returned.

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

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

`callback_query_id: str`

Unique identifier for the query to be answered

`text: str | None`

Text of the notification. If not specified, nothing will be shown to the user, 0-200 characters

`show_alert: bool | None`

If `True`, an alert will be shown by the client instead of a notification at the top of the chat screen. Defaults to `false`.

`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.

`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) → 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

```
class aiogram.methods.ban_chat_member.BanChatMember(*, chat_id: int | str, user_id: int,
                                                    until_date: datetime | timedelta | int | None
                                                    = None, revoke_messages: bool | None =
                                                    None, **extra_data: Any)
```

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

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

`chat_id: int | str`

Unique identifier for the target group or username of the target supergroup or channel (in the format `@channelusername`)

`user_id: int`

Unique identifier of the target user

`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`

```
class aiogram.methods.ban_chat_sender_chat.BanChatSenderChat(*, chat_id: int | str,
                                                             sender_chat_id: int,
                                                             **extra_data: Any)
```

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

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

`chat_id: int | str`

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

`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.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) → 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

Imports:

- `from aiogram.methods.close import Close`
- `alias: from aiogram.methods import Close`

With specific bot

```
result: bool = await bot(Close(...))
```

As reply into Webhook in handler

```
return Close(...)
```

closeForumTopic

Returns: bool

```
class aiogram.methods.close_forum_topic.CloseForumTopic(*, chat_id: int | str,
                                                         message_thread_id: int, **extra_data:
                                                         Any)
```

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

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

`chat_id: int | str`

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

`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.close_forum_topic(...)
```

Method as object

Imports:

- `from aiogram.methods.close_forum_topic import CloseForumTopic`
- `alias: from aiogram.methods import CloseForumTopic`

With specific bot

```
result: bool = await bot(CloseForumTopic(...))
```

As reply into Webhook in handler

```
return CloseForumTopic(...)
```

closeGeneralForumTopic

Returns: bool

```
class aiogram.methods.close_general_forum_topic.CloseGeneralForumTopic(*, chat_id: int | str,
                                                                    **extra_data: Any)
```

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

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

`chat_id: int | str`

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

`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

Imports:

- `from aiogram.methods.close_general_forum_topic import CloseGeneralForumTopic`
- `alias: from aiogram.methods import CloseGeneralForumTopic`

With specific bot

```
result: bool = await bot(CloseGeneralForumTopic(...))
```

As reply into Webhook in handler

```
return CloseGeneralForumTopic(...)
```

copyMessageReturns: `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 a 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

Застаріло починаючи з версії 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

Застаріло починаючи з версії API:7.0: <https://core.telegram.org/bots/api-changelog#december-29-2023>

Usage

As bot method

```
result: MessageId = await bot.copy_message(...)
```

Method as object

Imports:

- `from aiogram.methods.copy_message import CopyMessage`
- `alias: from aiogram.methods import CopyMessage`

With specific bot

```
result: MessageId = await bot(CopyMessage(...))
```

As reply into Webhook in handler

```
return CopyMessage(...)
```

As shortcut from received object

- `aiogram.types.message.Message.copy_to()`

copyMessages

Returns: `List[MessageId]`

```
class aiogram.methods.copy_messages.CopyMessages(*, chat_id: int | str, from_chat_id: int | str,
                                                  message_ids: List[int], message_thread_id: int
                                                  / None = None, disable_notification: bool /
                                                  None = None, protect_content: bool / None =
                                                  None, remove_caption: bool / None = None,
                                                  **extra_data: Any)
```

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) → 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`

```
class aiogram.methods.create_forum_topic.CreateForumTopic(*, chat_id: int | str, name: str,
                                                         icon_color: int | None = None,
                                                         icon_custom_emoji_id: str | None
                                                         = None, **extra_data: Any)
```

Use this method to create a topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have the `can_manage_topics` administrator rights. Returns information about the created topic as a `aiogram.types.forum_topic.ForumTopic` object.

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

chat_id: int | str

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

name: str

Topic name, 1-128 characters

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`

```
class aiogram.methods.decline_chat_join_request.DeclineChatJoinRequest(*, chat_id: int | str,
                                                                        user_id: int,
                                                                        **extra_data: Any)
```

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

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

`chat_id: int | str`

Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)

`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

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
```

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_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
```

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_chat_sticker_set(...)
```

Method as object

Imports:

- `from aiogram.methods.delete_chat_sticker_set import DeleteChatStickerSet`
- `alias: from aiogram.methods import DeleteChatStickerSet`

With specific bot

```
result: bool = await bot(DeleteChatStickerSet(...))
```

As reply into Webhook in handler

```
return DeleteChatStickerSet(...)
```

As shortcut from received object

- `aiogram.types.chat.Chat.delete_sticker_set()`

deleteForumTopic

Returns: bool

```
class aiogram.methods.delete_forum_topic.DeleteForumTopic(*, chat_id: int / str,
                                                           message_thread_id: int,
                                                           **extra_data: Any)
```

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

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

`chat_id: int | str`

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

`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.delete_forum_topic(...)
```

Method as object

Imports:

- `from aiogram.methods.delete_forum_topic import DeleteForumTopic`
- `alias: from aiogram.methods import DeleteForumTopic`

With specific bot

```
result: bool = await bot(DeleteForumTopic(...))
```

As reply into Webhook in handler

```
return DeleteForumTopic(...)
```

deleteMyCommands

Returns: `bool`

```
class aiogram.methods.delete_my_commands.DeleteMyCommands(*, scope: BotCommandScopeDefault / BotCommandScopeAllPrivateChats / BotCommandScopeAllGroupChats / BotCommandScopeAllChatAdministrators / BotCommandScopeChat / BotCommandScopeChatAdministrators / BotCommandScopeChatMember / None = None, language_code: str / None = None, **extra_data: Any)
```

Use this method to delete the list of the bot's commands for the given scope and user language. After deletion, [higher level commands](#) will be shown to affected users. Returns `True` on success.

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

`scope`: `BotCommandScopeDefault` | `BotCommandScopeAllPrivateChats` | `BotCommandScopeAllGroupChats` | `BotCommandScopeAllChatAdministrators` | `BotCommandScopeChat` | `BotCommandScopeChatAdministrators` | `BotCommandScopeChatMember` | `None`

A JSON-serialized object, describing scope of users for which the commands are relevant. Defaults to `aiogram.types.bot_command_scope_default.BotCommandScopeDefault`.

`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`

```
class aiogram.methods.edit_chat_invite_link.EditChatInviteLink(*, chat_id: int | str,
                                                                invite_link: str, name: str |
                                                                None = None, expire_date:
                                                                datetime | timedelta | int |
                                                                None = None, member_limit:
                                                                int | None = None,
                                                                creates_join_request: bool |
                                                                None = None, **extra_data:
                                                                Any)
```

Use this method to edit a non-primary invite link created by the bot. The bot must be an administrator in the chat for this to work and must have the appropriate administrator rights. Returns the edited invite link as a `aiogram.types.chat_invite_link.ChatInviteLink` object.

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

`chat_id: int | str`

Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)

`invite_link: str`

The invite link to edit

`name: str | None`

Invite link name; 0-32 characters

`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

```
class aiogram.methods.edit_forum_topic.EditForumTopic(*, chat_id: int | str, message_thread_id:
    int, name: str | None = None,
    icon_custom_emoji_id: str | None =
    None, **extra_data: Any)
```

Use this method to edit name and icon of a topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have `can_manage_topics` administrator rights, unless it is the creator of the topic. Returns `True` on success.

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

chat_id: int | str

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

message_thread_id: int

Unique identifier for the target message thread of the forum topic

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.

name: str | None

New topic name, 0-128 characters. If not specified or empty, the current name of the topic will be kept

`icon_custom_emoji_id: str | None`

New unique identifier of the custom emoji shown as the topic icon. Use `aiogram.methods.get_forum_topic_icon_stickers.GetForumTopicIconStickers` to get all allowed custom emoji identifiers. Pass an empty string to remove the icon. If not specified, the current icon will be kept

Usage

As bot method

```
result: bool = await bot.edit_forum_topic(...)
```

Method as object

Imports:

- `from aiogram.methods.edit_forum_topic import EditForumTopic`
- `alias: from aiogram.methods import EditForumTopic`

With specific bot

```
result: bool = await bot(EditForumTopic(...))
```

As reply into Webhook in handler

```
return EditForumTopic(...)
```

editGeneralForumTopic

Returns: `bool`

```
class aiogram.methods.edit_general_forum_topic.EditGeneralForumTopic(*, chat_id: int | str,
                                                                       name: str,
                                                                       **extra_data: Any)
```

Use this method to edit the name of the „General“ topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have `can_manage_topics` administrator rights. Returns `True` on success.

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

`chat_id: int | str`

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

`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.

```
name: str
```

New topic name, 1-128 characters

Usage

As bot method

```
result: bool = await bot.edit_general_forum_topic(...)
```

Method as object

Imports:

- `from aiogram.methods.edit_general_forum_topic import EditGeneralForumTopic`
- `alias: from aiogram.methods import EditGeneralForumTopic`

With specific bot

```
result: bool = await bot(EditGeneralForumTopic(...))
```

As reply into Webhook in handler

```
return EditGeneralForumTopic(...)
```

exportChatInviteLink

Returns: `str`

```
class aiogram.methods.export_chat_invite_link.ExportChatInviteLink(*, chat_id: int / str,
                                                                    **extra_data: Any)
```

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

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

Source: <https://core.telegram.org/bots/api#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) → None`

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

Usage

As bot method

```
result: str = await bot.export_chat_invite_link(...)
```

Method as object

Imports:

- `from aiogram.methods.export_chat_invite_link import ExportChatInviteLink`
- `alias: from aiogram.methods import ExportChatInviteLink`

With specific bot

```
result: str = await bot(ExportChatInviteLink(...))
```

As reply into Webhook in handler

```
return ExportChatInviteLink(...)
```

As shortcut from received object

- `aiogram.types.chat.Chat.export_invite_link()`

forwardMessage

Returns: `Message`

```
class aiogram.methods.forward_message.ForwardMessage(*, chat_id: int | str, from_chat_id: int | str, message_id: int, message_thread_id: int | None = None, disable_notification: bool | None = None, protect_content: bool | ~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]`

```
class aiogram.methods.forward_messages.ForwardMessages(*, chat_id: int | str, from_chat_id: int | str, message_ids: List[int], message_thread_id: int | None = None, disable_notification: bool | None = None, protect_content: bool | None = None, **extra_data: Any)
```

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

Usage

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`

```
class aiogram.methods.get_business_connection.GetBusinessConnection(*,
                                                                    business_connection_id:
                                                                    str, **extra_data: Any)
```

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) → None`

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

Usage

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: ChatFullInfo

```
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_full_info.ChatFullInfo* 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) → None

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

Usage

As bot method

```
result: ChatFullInfo = 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: ChatFullInfo = await bot(GetChat(...))
```

getChatAdministrators

Returns: `List[Union[ChatMemberOwner, ChatMemberAdministrator, ChatMemberMember, ChatMemberRestricted, ChatMemberLeft, ChatMemberBanned]]`

```
class aiogram.methods.get_chat_administrators.GetChatAdministrators(*, chat_id: int | str,
                                                                    **extra_data: Any)
```

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

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

`chat_id: int | str`

Unique identifier for the target chat or username of the target supergroup or channel (in the format @channelusername)

`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[Union[ChatMemberOwner, ChatMemberAdministrator, ChatMemberMember,
↳ ChatMemberRestricted, ChatMemberLeft, ChatMemberBanned]] = await bot.get_chat_
↳ administrators(...)
```

Method as object

Imports:

- `from aiogram.methods.get_chat_administrators import GetChatAdministrators`
- `alias: from aiogram.methods import GetChatAdministrators`

With specific bot

```
result: List[Union[ChatMemberOwner, ChatMemberAdministrator, ChatMemberMember,  
↳ ChatMemberRestricted, ChatMemberLeft, ChatMemberBanned]] = await  
↳ bot(GetChatAdministrators(...))
```

As shortcut from received object

- `aiogram.types.chat.Chat.get_administrators()`

getChatMember

Returns: `Union[ChatMemberOwner, ChatMemberAdministrator, ChatMemberMember, ChatMemberRestricted, ChatMemberLeft, ChatMemberBanned]`

```
class aiogram.methods.get_chat_member.GetChatMember(*, chat_id: int | str, user_id: int,  
                                                    **extra_data: Any)
```

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

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

`chat_id: int | str`

Unique identifier for the target chat or username of the target supergroup or channel (in the format @channelusername)

`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

Usage

As bot method

```
result: Union[ChatMemberOwner, ChatMemberAdministrator, ChatMemberMember,
↳ ChatMemberRestricted, ChatMemberLeft, ChatMemberBanned] = await bot.get_chat_member(...
↳ )
```

Method as object

Imports:

- `from aiogram.methods.get_chat_member import GetChatMember`
- `alias: from aiogram.methods import GetChatMember`

With specific bot

```
result: Union[ChatMemberOwner, ChatMemberAdministrator, ChatMemberMember,
↳ ChatMemberRestricted, ChatMemberLeft, ChatMemberBanned] = await bot(GetChatMember(...))
```

As shortcut from received object

- `aiogram.types.chat.Chat.get_member()`

getChatMemberCount

Returns: `int`

```
class aiogram.methods.get_chat_member_count.GetChatMemberCount(*, chat_id: int | str,
                                                                **extra_data: Any)
```

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

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

`chat_id: int | str`

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

`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: int = await bot.get_chat_member_count(...)
```

Method as object

Imports:

- `from aiogram.methods.get_chat_member_count import GetChatMemberCount`
- `alias: from aiogram.methods import GetChatMemberCount`

With specific bot

```
result: int = await bot(GetChatMemberCount(...))
```

As shortcut from received object

- `aiogram.types.chat.Chat.get_member_count()`

getChatMenuButton

Returns: `Union[MenuButtonDefault, MenuButtonWebApp, MenuButtonCommands]`

```
class aiogram.methods.get_chat_menu_button.GetChatMenuButton(*, chat_id: int | None = None,
                                                             **extra_data: Any)
```

Use this method to get the current value of the bot's menu button in a private chat, or the default menu button. Returns *aiogram.types.menu_button.MenuButton* on success.

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

`chat_id: int | None`

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

`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: Union[MenuButtonDefault, MenuButtonWebApp, MenuButtonCommands] = await bot.get_
↳ chat_menu_button(...)
```

Method as object

Imports:

- from aiogram.methods.get_chat_menu_button import GetChatMenuButton
- alias: from aiogram.methods import GetChatMenuButton

With specific bot

```
result: Union[MenuButtonDefault, MenuButtonWebApp, MenuButtonCommands] = await
↳ bot(GetChatMenuButton(...))
```

getFile

Returns: File

```
class aiogram.methods.get_file.GetFile(*, file_id: str, **extra_data: Any)
```

Use this method to get basic information about a file and prepare it for downloading. For the moment, bots can download files of up to 20MB in size. On success, a *aiogram.types.file.File* object is returned. The file can then be downloaded via the link https://api.telegram.org/file/bot<token>/<file_path>, 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) → None
```

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

Usage

As bot method

```
result: File = await bot.get_file(...)
```

Method as object

Imports:

- from aiogram.methods.get_file import GetFile
- alias: from aiogram.methods import GetFile

With specific bot

```
result: File = await bot(GetFile(...))
```

getForumTopicIconStickers

Returns: List[Sticker]

```
class aiogram.methods.get_forum_topic_icon_stickers.GetForumTopicIconStickers(**extra_data: Any)
```

Use this method to get custom emoji stickers, which can be used as a forum topic icon by any user. Requires no parameters. Returns an Array of *aiogram.types.sticker.Sticker* objects.

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

```
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_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.

```
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: User = await bot.get_me(...)
```

Method as object

Imports:

- `from aiogram.methods.get_me import GetMe`
- `alias: from aiogram.methods import GetMe`

With specific bot

```
result: User = await bot(GetMe(...))
```

getMyCommands

Returns: List[BotCommand]

```
class aiogram.methods.get_my_commands.GetMyCommands(*, scope: BotCommandScopeDefault /  
BotCommandScopeAllPrivateChats /  
BotCommandScopeAllGroupChats /  
BotCommandScopeAllChatAdministrators /  
BotCommandScopeChat /  
BotCommandScopeChatAdministrators /  
BotCommandScopeChatMember / None =  
None, language_code: str / None = None,  
**extra_data: Any)
```

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

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

*scope: BotCommandScopeDefault | BotCommandScopeAllPrivateChats |
BotCommandScopeAllGroupChats | BotCommandScopeAllChatAdministrators |
BotCommandScopeChat | BotCommandScopeChatAdministrators | BotCommandScopeChatMember
| None*

A JSON-serialized object, describing scope of users. Defaults to *aiogram.types.bot_command_scope_default.BotCommandScopeDefault*.

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 or an empty string

Usage

As bot method

```
result: List[BotCommand] = await bot.get_my_commands(...)
```


Method as object

Imports:

- `from aiogram.methods.get_my_commands import GetMyCommands`
- `alias: from aiogram.methods import GetMyCommands`

With specific bot

```
result: List[BotCommand] = await bot(GetMyCommands(...))
```

getMyDefaultAdministratorRights

Returns: `ChatAdministratorRights`

```
class aiogram.methods.get_my_default_administrator_rights.GetMyDefaultAdministratorRights(*,
                                                                                          for_channels:
                                                                                          bool
                                                                                          /
                                                                                          None
                                                                                          =
                                                                                          None,
                                                                                          **extra_data:
                                                                                          Any)
```

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

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

`for_channels: bool | None`

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

`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`

```
class aiogram.methods.get_my_description.GetMyDescription(*, language_code: str | None = None, **extra_data: Any)
```

Use this method to get the current bot description for the given user language. Returns *aiogram.types.bot_description.BotDescription* on success.

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

`language_code: str | None`

A two-letter ISO 639-1 language code or an empty string

`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: 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) → 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

```
class aiogram.methods.get_my_short_description.GetMyShortDescription(*, language_code: str |  
                                                                    None = None,  
                                                                    **extra_data: Any)
```

Use this method to get the current bot short description for the given user language. Returns *aiogram.types.bot_short_description.BotShortDescription* on success.

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

`language_code: str | None`

A two-letter ISO 639-1 language code or an empty string

`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

```
class aiogram.methods.get_user_chat_boosts.GetUserChatBoosts(*, chat_id: int | str, user_id:  
                                                                int, **extra_data: Any)
```

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) → 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`

```
class aiogram.methods.get_user_profile_photos.GetUserProfilePhotos(*, user_id: int, offset: int
    / None = None, limit: int
    / None = None,
    **extra_data: Any)
```

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

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

`user_id: int`

Unique identifier of the target user

`offset: int | None`

Sequential number of the first photo to be returned. By default, all photos are returned.

`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.

`limit: int | None`

Limits the number of photos to be retrieved. Values between 1-100 are accepted. Defaults to 100.

Usage

As bot method

```
result: UserProfilePhotos = await bot.get_user_profile_photos(...)
```

Method as object

Imports:

- `from aiogram.methods.get_user_profile_photos import GetUserProfilePhotos`
- `alias: from aiogram.methods import GetUserProfilePhotos`

With specific bot

```
result: UserProfilePhotos = await bot(GetUserProfilePhotos(...))
```

As shortcut from received object

- `aiogram.types.user.User.get_profile_photos()`

hideGeneralForumTopic

Returns: bool

```
class aiogram.methods.hide_general_forum_topic.HideGeneralForumTopic(*, chat_id: int | str,
                                                                    **extra_data: Any)
```

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

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

`chat_id: int | str`

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

`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.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*) → 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.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
```

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

Usage

As bot method

```
result: bool = await bot.log_out(...)
```

Method as object

Imports:

- `from aiogram.methods.log_out import Logout`
- `alias: from aiogram.methods import Logout`

With specific bot

```
result: bool = await bot(Logout(...))
```

As reply into Webhook in handler

```
return Logout(...)
```

pinChatMessage

Returns: bool

```
class aiogram.methods.pin_chat_message.PinChatMessage(*, chat_id: int | str, message_id: int,
                                                         disable_notification: bool | None = None,
                                                         **extra_data: Any)
```

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

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

chat_id: int | str

Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)

message_id: int

Identifier of a message to pin

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`

Pass `True` if it is not necessary to send a notification to all chat members about the new pinned message. Notifications are always disabled in channels and private chats.

Usage

As bot method

```
result: bool = await bot.pin_chat_message(...)
```

Method as object

Imports:

- `from aiogram.methods.pin_chat_message import PinChatMessage`
- `alias: from aiogram.methods import PinChatMessage`

With specific bot

```
result: bool = await bot(PinChatMessage(...))
```

As reply into Webhook in handler

```
return PinChatMessage(...)
```

As shortcut from received object

- `aiogram.types.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 | 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 | 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(_ModelMetaclass__context: Any) → 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, post stories to the chat page, pin chat stories, and access the chat's story archive

`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

```
class aiogram.methods.reopen_forum_topic.ReopenForumTopic(*, chat_id: int | str,
                                                            message_thread_id: int,
                                                            **extra_data: Any)
```

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

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

`chat_id: int | str`

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

`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

```
class aiogram.methods.reopen_general_forum_topic.ReopenGeneralForumTopic(*, chat_id: int | str, **extra_data: Any)
```

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

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

chat_id: int | str

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

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

```
class aiogram.methods.restrict_chat_member.RestrictChatMember(*, chat_id: int | str, user_id:
                                                                int, permissions:
                                                                ChatPermissions,
                                                                use_independent_chat_permissions:
                                                                bool | None = None, until_date:
                                                                datetime | timedelta | int | None
                                                                = None, **extra_data: Any)
```

Use this method to restrict a user in a supergroup. The bot must be an administrator in the supergroup for this to work and must have the appropriate administrator rights. Pass **True** for all permissions to lift restrictions from a user. Returns **True** on success.

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

chat_id: int | str

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

user_id: int

Unique identifier of the target user

permissions: *ChatPermissions*

A JSON-serialized object for new user permissions

```
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.

```
until_date: datetime.datetime | datetime.timedelta | int | None
```

Date when restrictions will be lifted for the user; Unix time. If user is restricted for more than 366 days or less than 30 seconds from the current time, they are considered to be restricted forever

Usage

As bot method

```
result: bool = await bot.restrict_chat_member(...)
```

Method as object

Imports:

- `from aiogram.methods.restrict_chat_member import RestrictChatMember`
- `alias: from aiogram.methods import RestrictChatMember`

With specific bot

```
result: bool = await bot(RestrictChatMember(...))
```

As reply into Webhook in handler

```
return RestrictChatMember(...)
```

As shortcut from received object

- `aiogram.types.chat.Chat.restrict()`

revokeChatInviteLink

Returns: `ChatInviteLink`

```
class aiogram.methods.revoke_chat_invite_link.RevokeChatInviteLink(*, chat_id: int | str,
                                                                    invite_link: str,
                                                                    **extra_data: Any)
```

Use this method to revoke an invite link created by the bot. If the primary link is revoked, a new link is automatically generated. The bot must be an administrator in the chat for this to work and must have the appropriate administrator rights. Returns the revoked invite link as *aiogram.types.chat_invite_link.ChatInviteLink* object.

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

`chat_id: int | str`

Unique identifier of the target chat or username of the target channel (in the format `@channelusername`)

`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(*, chat_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

`allow_sending_without_reply: bool | None`

Pass `True` if the message should be sent even if the specified replied-to message is not found

Застаріло починаючи з версії 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

Застаріло починаючи з версії 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 | 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,
    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

`allow_sending_without_reply: bool | None`

Pass `True` if the message should be sent even if the specified replied-to message is not found

Застаріло починаючи з версії 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

Застаріло починаючи з версії 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: ~ai-
        ogram.types.inline_keyboard_markup.InlineKeyboardMarkup
        | ~ai-
        ogram.types.reply_keyboard_markup.ReplyKeyboardMarkup
        | ~ai-
        ogram.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) → 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

`allow_sending_without_reply: bool | None`
Pass `True` if the message should be sent even if the specified replied-to message is not found
Застаріло починаючи з версії 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
Застаріло починаючи з версії 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`

```
class aiogram.methods.send_dice.SendDice(*, chat_id: int | 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: ~ai-
                                         ogram.types.inline_keyboard_markup.InlineKeyboardMarkup
                                         / ~ai-
                                         ogram.types.reply_keyboard_markup.ReplyKeyboardMarkup
                                         / ~ai-
                                         ogram.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

`allow_sending_without_reply: bool | None`

Pass `True` if the message should be sent even if the specified replied-to message is not found

Застаріло починаючи з версії 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

Застаріло починаючи з версії 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 | 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
    | ~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

`allow_sending_without_reply: bool | None`

Pass True if the message should be sent even if the specified replied-to message is not found

Застапіло починаючи з версії 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

Застапіло починаючи з версії 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, 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: ~ai-
ogram.types.reply_parameters.ReplyParameters
| None = None, reply_markup: ~ai-
ogram.types.inline_keyboard_markup.InlineKeyboardMarkup
| ~ai-
ogram.types.reply_keyboard_markup.ReplyKeyboardMarkup
| ~ai-
ogram.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 during which the location will be updated (see [Live Locations](#), should be between 60 and 86400, or 0x7FFFFFFF for live locations that can be edited indefinitely.

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

`allow_sending_without_reply: bool | None`

Pass `True` if the message should be sent even if the specified replied-to message is not found

Застаріло починаючи з версії 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

Застаріло починаючи з версії 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: ~typing.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')>, 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

Застаріло починаючи з версії 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

Застаріло починаючи з версії 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: ~typing.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
                                              | ~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

`allow_sending_without_reply`: `bool` | None

Pass `True` if the message should be sent even if the specified replied-to message is not found

Застапіло починаючи з версії 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

Застапіло починаючи з версії 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

Застапіло починаючи з версії 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 | 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, 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

`allow_sending_without_reply: bool | None`

Pass `True` if the message should be sent even if the specified replied-to message is not found

Застаріло починаючи з версії 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

Застаріло починаючи з версії 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[~aiogram.types.input_poll_option.InputPollOption | str], business_connection_id: str | None = None, message_thread_id: int | None = None, question_parse_mode: str | ~aiogram.client.default.Default | None = <Default('parse_mode')>, question_entities: ~typing.List[~aiogram.types.message_entity.MessageEntity] | 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 | ~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[*InputPollOption* | str]

A JSON-serialized list of 2-10 answer options

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

`question_parse_mode: str | Default | None`

Mode for parsing entities in the question. See [formatting options](#) for more details. Currently, only custom emoji entities are allowed

`question_entities: List[MessageEntity] | None`

A JSON-serialized list of special entities that appear in the poll question. It can be specified instead of *question_parse_mode*

`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

`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_parse_mode: str | Default | None`

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

`explanation_entities: List[MessageEntity] | None`

A JSON-serialized list of special entities that appear in the poll explanation. It can be specified instead of *explanation_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

`allow_sending_without_reply`: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Застаріло починаючи з версії 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

Застаріло починаючи з версії 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: ~ai-
                                             ogram.types.inline_keyboard_markup.InlineKeyboardMarkup
                                             | ~ai-
                                             ogram.types.reply_keyboard_markup.ReplyKeyboardMarkup
                                             | ~ai-
                                             ogram.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.

`model_post_init(_ModelMetaclass__context: Any) → None`

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

`allow_sending_without_reply: bool | None`

Pass `True` if the message should be sent even if the specified replied-to message is not found

Застаріло починаючи з версії 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

Застаріло починаючи з версії 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 | ~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

`allow_sending_without_reply: bool | None`

Pass True if the message should be sent even if the specified replied-to message is not found

Застаріло починаючи з версії 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

Застаріло починаючи з версії API:7.0: <https://core.telegram.org/bots/api-changelog#december-29-2023>

Usage

As bot method

```
result: Message = await bot.send_video(...)
```

Method as object

Imports:

- `from aiogram.methods.send_video import SendVideo`
- `alias: from aiogram.methods import SendVideo`

With specific bot

```
result: Message = await bot(SendVideo(...))
```

As reply into Webhook in handler

```
return SendVideo(...)
```

As shortcut from received object

- `aiogram.types.message.Message.answer_video()`
- `aiogram.types.message.Message.reply_video()`
- `aiogram.types.chat_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 | 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

allow_sending_without_reply: bool | None

Pass True if the message should be sent even if the specified replied-to message is not found

Застаріло починаючи з версії 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

Застаріло починаючи з версії 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: ~ai-
    ogram.types.inline_keyboard_markup.InlineKeyboardMarkup
    | ~ai-
    ogram.types.reply_keyboard_markup.ReplyKeyboardMarkup
    | ~ai-
    ogram.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, or in .MP3 format, or in .M4A format (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

`allow_sending_without_reply: bool | None`

Pass `True` if the message should be sent even if the specified replied-to message is not found

Застаріло починаючи з версії 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

Застаріло починаючи з версії API:7.0: <https://core.telegram.org/bots/api-changelog#december-29-2023>

Usage

As bot method

```
result: Message = await bot.send_voice(...)
```

Method as object

Imports:

- `from aiogram.methods.send_voice import SendVoice`
- `alias: from aiogram.methods import SendVoice`

With specific bot

```
result: Message = await bot(SendVoice(...))
```

As reply into Webhook in handler

```
return SendVoice(...)
```

As shortcut from received object

- *aiogram.types.message.Message.answer_voice()*
- *aiogram.types.message.Message.reply_voice()*
- *aiogram.types.chat_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

```
class aiogram.methods.set_chat_description.SetChatDescription(*, chat_id: int | str,
                                                             description: str | None = None,
                                                             **extra_data: Any)
```

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

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

`chat_id: int | str`

Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)

`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 | 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) → 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.

Usage

As bot method

```
result: bool = await bot.set_chat_permissions(...)
```

Method as object

Imports:

- `from aiogram.methods.set_chat_permissions import SetChatPermissions`
- `alias: from aiogram.methods import SetChatPermissions`

With specific bot

```
result: bool = await bot(SetChatPermissions(...))
```

As reply into Webhook in handler

```
return SetChatPermissions(...)
```

As shortcut from received object

- `aiogram.types.chat.Chat.set_permissions()`

setChatPhoto

Returns: bool

```
class aiogram.methods.set_chat_photo.SetChatPhoto(*, chat_id: int | str, photo: InputFile,
**extra_data: Any)
```

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

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

`chat_id: int | str`

Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)

`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: InputFile`

New chat photo, uploaded using multipart/form-data

Usage

As bot method

```
result: bool = await bot.set_chat_photo(...)
```

Method as object

Imports:

- `from aiogram.methods.set_chat_photo import SetChatPhoto`
- `alias: from aiogram.methods import SetChatPhoto`

With specific bot

```
result: bool = await bot(SetChatPhoto(...))
```

As shortcut from received object

- `aiogram.types.chat.Chat.set_photo()`

setChatStickerSet

Returns: bool

```
class aiogram.methods.set_chat_sticker_set.SetChatStickerSet(*, chat_id: int | str,
                                                             sticker_set_name: str,
                                                             **extra_data: Any)
```

Use this method to set a new group sticker set for a supergroup. The bot must be an administrator in the chat for this to work and must have the appropriate administrator rights. Use the field `can_set_sticker_set` optionally returned in `aiogram.methods.get_chat.GetChat` requests to check if the bot can use this method. Returns `True` on success.

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

`chat_id: int | str`

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

`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_set_name: str`

Name of the sticker set to be set as the group sticker set

Usage

As bot method

```
result: bool = await bot.set_chat_sticker_set(...)
```

Method as object

Imports:

- `from aiogram.methods.set_chat_sticker_set import SetChatStickerSet`
- `alias: from aiogram.methods import SetChatStickerSet`

With specific bot

```
result: bool = await bot(SetChatStickerSet(...))
```

As reply into Webhook in handler

```
return SetChatStickerSet(...)
```

As shortcut from received object

- `aiogram.types.chat.Chat.set_sticker_set()`

setChatTitle

Returns: bool

```
class aiogram.methods.set_chat_title.SetChatTitle(*, chat_id: int | str, title: str, **extra_data: Any)
```

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

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

`chat_id: int | str`

Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)

`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

Usage

As bot method

```
result: bool = await bot.set_chat_title(...)
```

Method as object

Imports:

- `from aiogram.methods.set_chat_title import SetChatTitle`
- `alias: from aiogram.methods import SetChatTitle`

With specific bot

```
result: bool = await bot(SetChatTitle(...))
```

As reply into Webhook in handler

```
return SetChatTitle(...)
```

As shortcut from received object

- `aiogram.types.chat.Chat.set_title()`

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) → 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*) → 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(*,
                                                                                          ri-
                                                                                          ghts:
                                                                                          ChatAdmini-
                                                                                          stratorRi-
                                                                                          ghts
                                                                                          /
                                                                                          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`

```
class aiogram.methods.set_my_description.SetMyDescription(*, description: str | None = None,
                                                         language_code: str | None = None,
                                                         **extra_data: Any)
```

Use this method to change the bot's description, which is shown in the chat with the bot if the chat is empty. Returns `True` on success.

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

`description: str | None`

New bot description; 0-512 characters. Pass an empty string to remove the dedicated description for the given language.

`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 description will be applied to all users for whose language there is no dedicated description.

Usage

As bot method

```
result: bool = await bot.set_my_description(...)
```

Method as object

Imports:

- `from aiogram.methods.set_my_description import SetMyDescription`
- `alias: from aiogram.methods import SetMyDescription`

With specific bot

```
result: bool = await bot(SetMyDescription(...))
```

As reply into Webhook in handler

```
return SetMyDescription(...)
```

setMyName

Returns: `bool`

```
class aiogram.methods.set_my_name.SetMyName(*, name: str | None = None, language_code: str | None = None, **extra_data: Any)
```

Use this method to change the bot's name. Returns **True** on success.

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

`name: str | None`

New bot name; 0-64 characters. Pass an empty string to remove the dedicated name for the given language.

`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`

```
class aiogram.methods.set_my_short_description.SetMyShortDescription(*, short_description: str | None = None, language_code: str | None = None, **extra_data: Any)
```

Use this method to change the bot's short description, which is shown on the bot's profile page and is sent together with the link when users share the bot. Returns `True` on success.

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

`short_description: str | None`

New short description for the bot; 0-120 characters. Pass an empty string to remove the dedicated short description for the given language.

`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 short description will be applied to all users for whose language there is no dedicated short description.

Usage

As bot method

```
result: bool = await bot.set_my_short_description(...)
```

Method as object

Imports:

- `from aiogram.methods.set_my_short_description import SetMyShortDescription`
- `alias: from aiogram.methods import SetMyShortDescription`

With specific bot

```
result: bool = await bot(SetMyShortDescription(...))
```

As reply into Webhook in handler

```
return SetMyShortDescription(...)
```

unbanChatMember

Returns: `bool`

```
class aiogram.methods.unban_chat_member.UnbanChatMember(*, chat_id: int | str, user_id: int,
                                                         only_if_banned: bool | None = None,
                                                         **extra_data: Any)
```

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

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

`chat_id: int | str`

Unique identifier for the target group or username of the target supergroup or channel (in the format `@channelusername`)

```
user_id: int
    Unique identifier of the target user

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.

only_if_banned: bool | None
    Do nothing if the user is not banned
```

Usage

As bot method

```
result: bool = await bot.unban_chat_member(...)
```

Method as object

Imports:

- `from aiogram.methods.unban_chat_member import UnbanChatMember`
- `alias: from aiogram.methods import UnbanChatMember`

With specific bot

```
result: bool = await bot(UnbanChatMember(...))
```

As reply into Webhook in handler

```
return UnbanChatMember(...)
```

As shortcut from received object

- `aiogram.types.chat.Chat.unban()`

unbanChatSenderChat

Returns: bool

```
class aiogram.methods.unban_chat_sender_chat.UnbanChatSenderChat(*, chat_id: int | str,
                                                                sender_chat_id: int,
                                                                **extra_data: Any)
```

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

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

`chat_id: int | str`

Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)

`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

```
class aiogram.methods.unhide_general_forum_topic.UnhideGeneralForumTopic(*, chat_id: int /  
                                                                           str, **extra_data:  
                                                                           Any)
```

Use this method to unhide the „General“ topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have the `can_manage_topics` administrator rights. Returns True on success.

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

`chat_id: int | str`

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

`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

```
class aiogram.methods.unpin_all_chat_messages.UnpinAllChatMessages(*, chat_id: int | str,
                                                                    **extra_data: Any)
```

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

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

chat_id: int | str

Unique identifier for the target chat or username of the target channel (in the format @channelusername)

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.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

Usage

As bot method

```
result: bool = await bot.unpin_all_forum_topic_messages(...)
```

Method as object

Imports:

- from aiogram.methods.unpin_all_forum_topic_messages import UnpinAllForumTopicMessages
- alias: from aiogram.methods import UnpinAllForumTopicMessages

With specific bot

```
result: bool = await bot(UnpinAllForumTopicMessages(...))
```

As reply into Webhook in handler

```
return UnpinAllForumTopicMessages(...)
```

unpinAllGeneralForumTopicMessages

Returns: bool

```
class aiogram.methods.unpin_all_general_forum_topic_messages.UnpinAllGeneralForumTopicMessages(*,
                                                                                               chat_id:
                                                                                               int
                                                                                               /
                                                                                               str,
                                                                                               **extra_a
                                                                                               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.

Usage

As bot method

```
result: bool = await bot.unpin_all_general_forum_topic_messages(...)
```

Method as object

Imports:

- `from aiogram.methods.unpin_all_general_forum_topic_messages import UnpinAllGeneralForumTopicMessages`
- `alias: from aiogram.methods import UnpinAllGeneralForumTopicMessages`

With specific bot

```
result: bool = await bot(UnpinAllGeneralForumTopicMessages(...))
```

As reply into Webhook in handler

```
return UnpinAllGeneralForumTopicMessages(...)
```

As shortcut from received object

- `aiogram.types.chat.Chat.unpin_all_general_forum_topic_messages()`

unpinChatMessage

Returns: bool

```
class aiogram.methods.unpin_chat_message.UnpinChatMessage(*, chat_id: int | str, message_id: int  
                                                         / None = None, **extra_data: Any)
```

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

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

`chat_id: int | str`

Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)

`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

```
class aiogram.methods.delete_message.DeleteMessage(*, chat_id: int | str, message_id: int,
**extra_data: Any)
```

Use this method to delete a message, including service messages, with the following limitations:

- A message can only be deleted if it was sent less than 48 hours ago.
- Service messages about a supergroup, channel, or forum topic creation can't be deleted.

- A dice message in a private chat can only be deleted if it was sent more than 24 hours ago.
- Bots can delete outgoing messages in private chats, groups, and supergroups.
- Bots can delete incoming messages in private chats.
- Bots granted `can_post_messages` permissions can delete outgoing messages in channels.
- If the bot is an administrator of a group, it can delete any message there.
- If the bot has `can_delete_messages` permission in a supergroup or a channel, it can delete any message there.

Returns `True` on success.

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

`chat_id: int | str`

Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)

`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

```
class aiogram.methods.delete_messages.DeleteMessages(*, chat_id: int / str, message_ids:
                                                    List[int], **extra_data: Any)
```

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) → 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]

```
class aiogram.methods.edit_message_caption.EditMessageCaption(*, chat_id: int | str | None =
    None, message_id: int | None =
    None, inline_message_id: str |
    None = None, caption: str |
    None = None, parse_mode: str |
    ~aiogram.client.default.Default |
    None =
    <Default('parse_mode')>,
    caption_entities: ~typing.
    List[~aiogram.types.message_entity.MessageEntity]
    | None = None, reply_markup:
    ~aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup
    | None = None, **extra_data:
    ~typing.Any)
```

Use this method to edit captions of messages. On success, if the edited message is not an inline message, the edited *aiogram.types.message.Message* is returned, otherwise **True** is returned.

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

chat_id: int | str | None

Required if *inline_message_id* is not specified. Unique identifier for the target chat or username of the target channel (in the format @channelusername)

message_id: int | None

Required if *inline_message_id* is not specified. Identifier of the message to edit

inline_message_id: str | None

Required if *chat_id* and *message_id* are not specified. Identifier of the inline message

caption: str | None

New caption of the message, 0-1024 characters after entities parsing

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 | None = None,
                                                                    live_period: int |
                                                                    None = None,
                                                                    hori-
                                                                    zontal_accuracy:
                                                                    float | None =
                                                                    None, heading: int
                                                                    | None = None,
                                                                    proximi-
                                                                    ty_alert_radius:
                                                                    int | None = None,
                                                                    reply_markup: Inli-
                                                                    neKeyboardMarkup
                                                                    | 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) → None

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

live_period: int | None

New period in seconds during which the location can be updated, starting from the message send date. If 0x7FFFFFFF is specified, then the location can be updated forever. Otherwise, the new value must not exceed the current *live_period* by more than a day, and the live location expiration date must remain within the next 90 days. If not specified, then *live_period* remains unchanged

`horizontal_accuracy: float | None`

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

`heading: int | None`

Direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.

`proximity_alert_radius: int | None`

The maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and 100000 if specified.

`reply_markup: InlineKeyboardMarkup | None`

A JSON-serialized object for a new inline keyboard.

Usage

As bot method

```
result: Union[Message, bool] = await bot.edit_message_live_location(...)
```

Method as object

Imports:

- `from aiogram.methods.edit_message_live_location import EditMessageLiveLocation`
- `alias: from aiogram.methods import EditMessageLiveLocation`

With specific bot

```
result: Union[Message, bool] = await bot(EditMessageLiveLocation(...))
```

As reply into Webhook in handler

```
return EditMessageLiveLocation(...)
```

As shortcut from received object

- `aiogram.types.message.Message.edit_live_location()`

editMessageMedia

Returns: Union[Message, bool]

```
class aiogram.methods.edit_message_media.EditMessageMedia(*, media: InputMediaAnimation /
    InputMediaDocument /
    InputMediaAudio / InputMediaPhoto
    / InputMediaVideo, chat_id: int / str
    / None = None, message_id: int /
    None = None, inline_message_id:
    str / None = None, reply_markup:
    InlineKeyboardMarkup / None =
    None, **extra_data: Any)
```

Use this method to edit animation, audio, document, photo, or video messages. If a message is part of a message album, then it can be edited only to an audio for audio albums, only to a document for document albums and to a photo or a video otherwise. When an inline message is edited, a new file can't be uploaded; use a previously uploaded file via its file_id or specify a URL. On success, if the edited message is not an inline message, the edited `aiogram.types.message.Message` is returned, otherwise `True` is returned.

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

`media: InputMediaAnimation | InputMediaDocument | InputMediaAudio | InputMediaPhoto | InputMediaVideo`

A JSON-serialized object for a new media content of the message

`chat_id: int | str | None`

Required if `inline_message_id` is not specified. Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)

`message_id: int | None`

Required if `inline_message_id` is not specified. Identifier of the message to edit

`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:
~typi-
ng.List[~aiogram.types.message_entity.MessageEntity]
| None = None, link_preview_options:
~ai-
ogram.types.link_preview_options.LinkPreviewOptions
| None = None, reply_markup: ~ai-
ogram.types.inline_keyboard_markup.InlineKeyboardMar
| 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) → 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

Застаріло починаючи з версії 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]`

```
class aiogram.methods.stop_message_live_location.StopMessageLiveLocation(*, chat_id: int | str
    / None = None,
    message_id: int |
    None = None,
    inline_message_id:
    str | None = None,
    reply_markup: Inli-
    neKeyboardMarkup
    / 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*) → 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) → 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
    / InlineQueryResultCachedSticker
    / InlineQueryResultCachedVideo
    / InlineQueryResultCachedVoice
    / InlineQueryResultArticle
    / InlineQueryResultAudio
    / InlineQueryResultContact
    / InlineQueryResultGame
    / InlineQueryResultDocument
    / InlineQueryResultGif
    / InlineQueryResultLocation
    / InlineQueryResultMpeg4Gif
    / InlineQueryResultPhoto
    / InlineQueryResultVenue
    / InlineQueryResultVideo
    / InlineQueryResultVoice],
    cache_time: int / None = None,
    is_personal: bool / None = None,
    next_offset: str / None = None,
    button: InlineQueryResultsButton /
    None = None,
    switch_pm_parameter: str / None
    = None, switch_pm_text: str /
    None = None, **extra_data: Any)
```

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

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

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

inline_query_id: str

Unique identifier for the answered query

results: List[InlineQueryResultCachedAudio | InlineQueryResultCachedDocument | InlineQueryResultCachedGif | InlineQueryResultCachedMpeg4Gif | InlineQueryResultCachedPhoto | InlineQueryResultCachedSticker | InlineQueryResultCachedVideo | InlineQueryResultCachedVoice | InlineQueryResultArticle | InlineQueryResultAudio | InlineQueryResultContact | InlineQueryResultGame | InlineQueryResultDocument | InlineQueryResultGif | InlineQueryResultLocation | InlineQueryResultMpeg4Gif | InlineQueryResultPhoto | InlineQueryResultVenue | InlineQueryResultVideo | InlineQueryResultVoice]

A JSON-serialized array of results for the inline query

cache_time: int | None

The maximum amount of time in seconds that the result of the inline query may be cached on the server. Defaults to 300.

is_personal: bool | None

Pass **True** if results may be cached on the server side only for the user that sent the query. By

default, results may be returned to any user who sends the same query.

`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.

Застаріло починаючи з версії 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*

Застаріло починаючи з версії 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 /
                                                             Inli-
                                                             neQueryResultCachedDocument /
                                                             InlineQueryResultCachedGif /
                                                             Inli-
                                                             neQueryResultCachedMpeg4Gif /
                                                             InlineQueryResultCachedPhoto /
                                                             InlineQueryResultCachedSticker /
                                                             InlineQueryResultCachedVideo /
                                                             InlineQueryResultCachedVoice /
                                                             InlineQueryResultArticle /
                                                             InlineQueryResultAudio /
                                                             InlineQueryResultContact /
                                                             InlineQueryResultGame /
                                                             InlineQueryResultDocument /
                                                             InlineQueryResultGif /
                                                             InlineQueryResultLocation /
                                                             InlineQueryResultMpeg4Gif /
                                                             InlineQueryResultPhoto /
                                                             InlineQueryResultVenue /
                                                             InlineQueryResultVideo /
                                                             InlineQueryResultVoice,
                                                             **extra_data: Any)
```

Use this method to set the result of an interaction with a [Web App](#) and send a corresponding message on behalf of the user to the chat from which the query originated. On success, a `aiogram.types.sent_web_app_message.SentWebAppMessage` object is returned.

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

web_app_query_id: str

Unique identifier for the query to be answered

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]

```

class aiogram.methods.get_game_high_scores.GetGameHighScores(*, user_id: int, chat_id: int /
    None = None, message_id: int /
    None = None,
    inline_message_id: str / None =
    None, **extra_data: Any)

```

Use this method to get data for high score tables. Will return the score of the specified user and several of their neighbors in a game. Returns an Array of *aiogram.types.game_high_score.GameHighScore* objects.

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

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

`user_id: int`

Target user id

`chat_id: int | None`

Required if `inline_message_id` is not specified. Unique identifier for the target chat

`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`

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.

`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 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.

`allow_sending_without_reply: bool | None`

Pass True if the message should be sent even if the specified replied-to message is not found

Застаріло починаючи з версії 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

Застаріло починаючи з версії 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]`

```
class aiogram.methods.set_game_score.SetGameScore(*, user_id: int, score: int, force: bool | None =
    None, disable_edit_message: bool | None =
    None, chat_id: int | None = None,
    message_id: int | None = None,
    inline_message_id: str | None = None,
    **extra_data: Any)
```

Use this method to set the score of the specified user in a game message. On success, if the message is not an inline message, the `aiogram.types.message.Message` is returned, otherwise `True` is returned. Returns an error, if the new score is not greater than the user's current score in the chat and `force` is `False`.

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

```

user_id: int
    User identifier

score: int
    New score, must be non-negative

force: bool | None
    Pass True if the high score is allowed to decrease. This can be useful when fixing mistakes or
    banning cheaters

disable_edit_message: bool | None
    Pass True if the game message should not be automatically edited to include the current
    scoreboard

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 | 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

```
class aiogram.methods.answer_shipping_query.AnswerShippingQuery(*, shipping_query_id: str, ok:
                                                                    bool, shipping_options:
                                                                    List[ShippingOption] | None
                                                                    = None, error_message: str |
                                                                    None = None, **extra_data:
                                                                    Any)
```

If you sent an invoice requesting a shipping address and the parameter *is_flexible* was specified, the Bot API will send an `aiogram.types.update.Update` with a *shipping_query* field to the bot. Use this method to reply to shipping queries. On success, `True` is returned.

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

shipping_query_id: str

Unique identifier for the query to be answered

ok: bool

Pass `True` if delivery to the specified address is possible and `False` if there are any problems (for example, if delivery to the specified address is not possible)

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 | None = None,
                                                           photo_height: int | None = None,
                                                           need_name: bool | None = None,
                                                           need_phone_number: bool | None
                                                           = None, need_email: bool | None
                                                           = None, need_shipping_address:
                                                           bool | None = None,
                                                           send_phone_number_to_provider:
                                                           bool | None = None,
                                                           send_email_to_provider: bool |
                                                           None = None, is_flexible: bool |
                                                           None = None, **extra_data: Any)
```

Use this method to create a link for an invoice. Returns the created invoice link as *String* on success.

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

title: str

Product name, 1-32 characters

description: str

Product description, 1-255 characters

payload: str

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

provider_token: str

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

currency: str

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

prices: List[LabeledPrice]

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

max_tip_amount: int | None

The maximum accepted amount for tips in the *smallest units* of the currency (integer, **not** float/double). For example, for a maximum tip of US\$ 1.45 pass `max_tip_amount = 145`. See the *exp* parameter in [currencies.json](#), it shows the number of digits past the decimal point for each currency (2 for the majority of currencies). Defaults to 0

suggested_tip_amounts: List[int] | None

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

`provider_data: str | None`

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

`photo_url: str | None`

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

`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`

```
class aiogram.methods.send_invoice.SendInvoice(*, chat_id: int | str, title: str, description: str,
        payload: str, provider_token: str, currency: str,
        prices: ~typing.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,
        send_email_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: ~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 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) → 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

Застаріло починаючи з версії 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

Застаріло починаючи з версії 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

```
class aiogram.methods.delete_webhook.DeleteWebhook(*, drop_pending_updates: bool | None = None, **extra_data: Any)
```

Use this method to remove webhook integration if you decide to switch back to `aiogram.methods.get_updates.GetUpdates`. Returns `True` on success.

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

`drop_pending_updates: bool | None`

Pass `True` to drop all pending updates


```
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_webhook(...)
```

Method as object

Imports:

- `from aiogram.methods.delete_webhook import DeleteWebhook`
- `alias: from aiogram.methods import DeleteWebhook`

With specific bot

```
result: bool = await bot(DeleteWebhook(...))
```

As reply into Webhook in handler

```
return DeleteWebhook(...)
```

getUpdates

Returns: `List[Update]`

```
class aiogram.methods.get_updates.GetUpdates(*, offset: int | None = None, limit: int | None =
                                             None, timeout: int | None = None, allowed_updates:
                                             List[str] | None = None, **extra_data: Any)
```

Use this method to receive incoming updates using long polling ([wiki](#)). Returns an Array of *aiogram.types.update.Update* objects.

Notes

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

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

`offset: int | None`

Identifier of the first update to be returned. Must be greater by one than the highest among the identifiers of previously received updates. By default, updates starting with the earliest unconfirmed update are returned. An update is considered confirmed as soon as [aiogram.methods.get_updates.GetUpdates](#) is called with an *offset* higher than its *update_id*. The negative offset can be specified to retrieve updates starting from *-offset* update from the end of the updates queue. All previous updates will be forgotten.

`limit: int | None`

Limits the number of updates to be retrieved. Values between 1-100 are accepted. Defaults to 100.

`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.

`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) → 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) → 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 /
    PassportElementErrorFrontSide /
    PassportElementErrorReverseSide /
    PassportElementErrorSelfie /
    PassportElementErrorFile /
    PassportElementErrorFiles /
    PassportElementErrorTranslationFile /
    PassportElementErrorTranslationFiles /
    PassportElementErrorUnspecified], **extra_data: Any)
```

Inform 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

Ось список усіх доступних переліків:

BotCommandScopeType

```
class aiogram.enums.bot_command_scope_type.BotCommandScopeType(value, names=None, *,
                                                                module=None,
                                                                qualname=None, type=None,
                                                                start=1, boundary=None)
```

Цей об'єкт представляє область, до якої застосовуються команди бота.

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

```
DEFAULT = 'default'
```

```
ALL_PRIVATE_CHATS = 'all_private_chats'
```

```
ALL_GROUP_CHATS = 'all_group_chats'
```

```
ALL_CHAT_ADMINISTRATORS = 'all_chat_administrators'
```

```
CHAT = 'chat'
```

```
CHAT_ADMINISTRATORS = 'chat_administrators'
```

```
CHAT_MEMBER = 'chat_member'
```

ChatAction

```
class aiogram.enums.chat_action.ChatAction(value, names=None, *, module=None,
                                            qualname=None, type=None, start=1,
                                            boundary=None)
```

Цей об'єкт представляє дії бота.

Виберіть один залежно від того, що користувач збирається отримати:

- `typing` для текстових повідомлень,
- `upload_photo` для фотографій,
- `record_video` або `upload_video` для відео,
- `record_voice` або `upload_voice` для голосових повідомлень,
- `upload_document` для загальних файлів,
- `choose_sticker` для наклею,
- `find_location` для даних про місце знаходження,
- `record_video_note` або `upload_video_note` для відео кружків.

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

```
TYPING = 'typing'
UPLOAD_PHOTO = 'upload_photo'
RECORD_VIDEO = 'record_video'
UPLOAD_VIDEO = 'upload_video'
RECORD_VOICE = 'record_voice'
UPLOAD_VOICE = 'upload_voice'
UPLOAD_DOCUMENT = 'upload_document'
CHOOSE_STICKER = 'choose_sticker'
FIND_LOCATION = 'find_location'
RECORD_VIDEO_NOTE = 'record_video_note'
UPLOAD_VIDEO_NOTE = 'upload_video_note'
```

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

```
class aiogram.enums.chat_member_status.ChatMemberStatus(value, names=None, *, module=None,
                                                         qualname=None, type=None, start=1,
                                                         boundary=None)
```

Цей об'єкт представляє статус учасника чату.

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

```
CREATOR = 'creator'
ADMINISTRATOR = 'administrator'
MEMBER = 'member'
RESTRICTED = 'restricted'
```



```
LEFT = 'left'

KICKED = 'kicked'
```

ChatType

```
class aiogram.enums.chat_type.ChatType(value, names=None, *, module=None, qualname=None,
                                       type=None, start=1, boundary=None)
```

Цей об'єкт представляє тип чату.

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

```
SENDER = 'sender'

PRIVATE = 'private'

GROUP = 'group'

SUPERGROUP = 'supergroup'

CHANNEL = 'channel'
```

ContentType

```
class aiogram.enums.content_type.ContentType(value, names=None, *, module=None,
                                             qualname=None, type=None, start=1,
                                             boundary=None)
```

Цей об'єкт представляє тип вмісту в повідомленні.

```
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'

CHAT_BACKGROUND_SET = 'chat_background_set'

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

```
class aiogram.enums.dice_emoji.DiceEmoji(value, names=None, *, module=None, qualname=None,
                                          type=None, start=1, boundary=None)
```

Емоджі, на яких базується анімація кидка кубика.

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

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

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'

MPG4_GIF = 'mpeg4_gif'

PHOTO = 'photo'

STICKER = 'sticker'

VIDEO = 'video'

VOICE = 'voice'

ARTICLE = 'article'

CONTACT = 'contact'

GAME = 'game'

LOCATION = 'location'

VENUE = 'venue'

```

InputMediaType

```
class aiogram.enums.input_media_type.InputMediaType(value, names=None, *, module=None,
                                                    qualname=None, type=None, start=1,
                                                    boundary=None)
```

Цей об'єкт представляє тип вхідного медіа.

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

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

```
class aiogram.enums.mask_position_point.MaskPositionPoint(value, names=None, *,
                                                          module=None, qualname=None,
                                                          type=None, start=1,
                                                          boundary=None)
```

Частина обличчя, щодо якої слід розмістити маску.

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

FOREHEAD = 'forehead'

EYES = 'eyes'

MOUTH = 'mouth'

CHIN = 'chin'

MenuButtonType

```
class aiogram.enums.menu_button_type.MenuButtonType(value, names=None, *, module=None,
                                                    qualname=None, type=None, start=1,
                                                    boundary=None)
```

Цей об'єкт представляє тип кнопки меню.

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

DEFAULT = 'default'

COMMANDS = 'commands'

WEB_APP = 'web_app'

MessageEntityType

```
class aiogram.enums.message_entity_type.MessageEntityType(value, names=None, *,
                                                         module=None, qualname=None,
                                                         type=None, start=1,
                                                         boundary=None)
```

Цей об'єкт представляє тип сутності повідомлення.

Джерело: <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

```
class aiogram.enums.message_origin_type.MessageOriginType(value, names=None, *,
                                                           module=None, qualname=None,
                                                           type=None, start=1,
                                                           boundary=None)
```

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

```
class aiogram.enums.parse_mode.ParseMode(value, names=None, *, module=None, qualname=None,
                                           type=None, start=1, boundary=None)
```

Параметри форматування.

Джерело: <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

```
class aiogram.enums.poll_type.PollType(value, names=None, *, module=None, qualname=None,
                                       type=None, start=1, boundary=None)
```

Цей об'єкт представляє тип опитування.

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

```
REGULAR = 'regular'

QUIZ = 'quiz'
```

ReactionTypeType

```
class aiogram.enums.reaction_type_type.ReactionTypeType(value, names=None, *, module=None,
                                                         qualname=None, type=None, start=1,
                                                         boundary=None)
```

This object represents reaction type.

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

```
EMOJI = 'emoji'

CUSTOM_EMOJI = 'custom_emoji'
```

StickerFormat

```
class aiogram.enums.sticker_format.StickerFormat(value, names=None, *, module=None,
                                                  qualname=None, type=None, start=1,
                                                  boundary=None)
```

Format of the sticker

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

```
STATIC = 'static'

ANIMATED = 'animated'

VIDEO = 'video'
```

StickerType

```
class aiogram.enums.sticker_type.StickerType(value, names=None, *, module=None,
                                              qualname=None, type=None, start=1,
                                              boundary=None)
```

Частина обличчя, щодо якої слід розмістити маску.

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

REGULAR = 'regular'

MASK = 'mask'

CUSTOM_EMOJI = 'custom_emoji'

TopicIconColor

```
class aiogram.enums.topic_icon_color.TopicIconColor(value, names=None, *, module=None,
                                                      qualname=None, type=None, start=1,
                                                      boundary=None)
```

Колір значка теми у форматі RGB.

Джерело: 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)
```

Цей об'єкт представляє повний список дозволених типів оновлення.

Джерело: <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 Як завантажити файл?

Завантаження файлу вручну

По-перше, ви повинні отримати *file_id* файлу, який ви хочете завантажити. Інформація про файли, надіслані боту, міститься в [Message](#).

Наприклад, завантажте документ, який прийшов боту.

```
file_id = message.document.file_id
```

Потім скористайтеся методом [getFile](#), щоб отримати *file_path*.

```
file = await bot.get_file(file_id)
file_path = file.file_path
```

Після цього скористайтеся методом [download_file](#) з об'єкта бота.

download_file(...)

Завантажує файл за *file_path* у вказане місце.

Якщо ви хочете автоматично створити місце призначення (*io.BytesIO*), використовуйте значення призначення за замовчуванням і обробіть результат цього методу.

```
async Bot.download_file(file_path: str, destination: BinaryIO | Path | str | None = None, timeout: int = 30, chunk_size: int = 65536, seek: bool = True) → BinaryIO | None
```

Завантажує файл з *file_path* у вказане місце.

Якщо ви хочете автоматично створити місце призначення (*io.BytesIO*), використовуйте значення призначення за замовчуванням і обробіть результат цього методу.

Параметри

- *file_path* – Шлях до файлу на сервері Telegram (Ви можете отримати його з *aiogram.types.File*)
- *destination* – Ім'я файлу, шлях до файлу або екземпляр *io.IOBase*. Для напр. *io.BytesIO*, за замовчуванням немає
- *timeout* – Загальний час очікування в секундах, за замовчуванням 30
- *chunk_size* – Розмір фрагментів файлу, за замовчуванням 64 Кб
- *seek* – Перейти до початку файлу, коли завантаження завершиться. Використовується лише для призначення з типом *typing.BinaryIO*, за замовчуванням значення *True*

Існує два варіанти завантаження файлу: на **disk** або на **binary I/O object**.

Завантаження файлу на диск

Щоб завантажити файл на диск, необхідно вказати ім'я файлу або шлях, куди його завантажити. У цьому випадку функція нічого не поверне.

```
await bot.download_file(file_path, "text.txt")
```

Завантаження файлу в оперативну пам'ять

Щоб завантажити файл до оперативної пам'яті, ви повинні вказати об'єкт із типом *typing.BinaryIO* або використати значення за замовчуванням (*None*).

У першому випадку функція поверне ваш об'єкт:

```
my_object = MyBinaryIO()
result: MyBinaryIO = await bot.download_file(file_path, my_object)
# print(result is my_object) # True
```

Якщо залишити значення за замовчуванням, буде створено та повернено об'єкт *io.BytesIO*.

```
result: io.BytesIO = await bot.download_file(file_path)
```

Завантаження файла коротким шляхом

Щоразу добувати `file_path` вручну нудно, тому вам слід використовувати метод `download`.

`download(...)`

Завантажує файл за `file_id` або `Downloadable` об'єктом у вказане місце.

Якщо ви хочете автоматично створити місце призначення (`io.BytesIO`), використовуйте значення призначення за замовчуванням і обробіть результат цього методу.

`async Bot.download(file: str | Downloadable, destination: BinaryIO | Path | str | None = None, timeout: int = 30, chunk_size: int = 65536, seek: bool = True) → BinaryIO | None`

Завантажує файл за `file_id` або `Downloadable` об'єктом у вказане місце.

Якщо ви хочете автоматично створити місце призначення (`io.BytesIO`), використовуйте значення призначення за замовчуванням і обробіть результат цього методу.

Параметри

- `file` – `file_id` або `Downloadable` об'єкт
- `destination` – Ім'я файлу, шлях до файлу або екземпляр `io.IOBase`. Для напр. `io.BytesIO`, за замовчуванням немає
- `timeout` – Загальний час очікування в секундах, за замовчуванням 30
- `chunk_size` – Розмір фрагментів файлу, за замовчуванням 64 Кб
- `seek` – Перейти до початку файлу, коли завантаження завершиться. Використовується лише для призначення з типом `typing.BinaryIO`, за замовчуванням значення `True`

Він відрізняється від `download_file` **лише** тим, що приймає `file_id` або `Downloadable` об'єкт (об'єкт, який містить атрибут `file_id`) замість `file_path`.

Ви можете завантажити файл на `disk` або в `binary I/O object` так само.

Приклад:

```
document = message.document
await bot.download(document)
```

2.3.7 Як відвантажити файл?

Як стверджує [official Telegram Bot API documentation](#) існує три способа надіслати файл (фото, наклейки, аудіо, медіа тощо):

Якщо файл уже зберігається десь на серверах Telegram або файл доступний за URL-адресою, вам не потрібно його повторно завантажувати.

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` - відвантажений з файлової системи
- `aiogram.types.input_file.BufferedInputFile` - відвантажений з буферу
- `aiogram.types.input_file.URLInputFile` - відвантажений з URL

Попередження: Поважайте 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.

Відвантаження з файлової системи

Перш за все, вам потрібно буде імпортувати обгортку *InputFile*:

```
from aiogram.types import FSInputFile
```

Тепер ви можете використовувати її:

```
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)
```

Об'єкт для відвантаження файлів із файлової системи

Параметри

- *path* – Шлях до файлу
- *filename* – Ім'я файлу, яке буде передано в telegram. За замовчуванням, буде взято зі шляху
- *chunk_size* – Розмір фрагмента відвантаження

Відвантаження з буферу

Files can be also passed from buffer (For example you generate image using *Pillow* and you want to send it to Telegram):

Імпорт обгортки:

```
from aiogram.types import BufferedInputFile
```

Тепер ви можете використовувати її:

```
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)
```

Об'єкт для відвантаження файлів із файлової системи

Параметри

- *file* – Байти
- *filename* – Ім'я файлу, яке буде передано в telegram.

- `chunk_size` – Розмір фрагмента відвантаження

Відвантаження з URL

Якщо вам потрібно відвантажити файл з іншого сервера, але пряме посилання прив'язано до IP-адреси вашого сервера, або ви хочете обійти власні обмеження на завантаження `<https://core.telegram.org/bots/api#sending-files>` – за URL-адресою, ви можете використовувати об'єкт `aiogram.types.input_file.URLInputFile`.

Імпорт обгортки:

```
from aiogram.types import URLInputFile
```

Тепер ви можете використовувати її:

```
image = URLInputFile(
    "https://www.python.org/static/community_logos/python-powered-h-140x182.png",
    filename="python-logo.png"
)
```

```
class aiogram.types.input_file.URLInputFile(url: str, headers: Dict[str, Any] | None = None,
                                             filename: str | None = None, chunk_size: int =
                                             65536, timeout: int = 30, bot: 'Bot' | None = None)
```

2.4 Обробка подій

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 Маршрутизатор

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)
```

Базується на object

Маршрутизатор може маршрутизувати події, а також вкладені типи оновлень, такі як повідомлення, запит зворотного виклику, опитування та всі інші типи подій.

Обробники подій можуть бути зареєстровані в обсервері двома шляхами:

- За допомогою методу обсервера - `router.<event_type>.register(handler, <filters, ...>)`
- За допомогою декоратора - `@router.<event_type>(<filters, ...>)`

```
__init__(*, name: str / None = None) -> None
```

Параметри

`name` – Додаткова назва маршрутизатора, може бути корисною для відлагодження

```
include_router(router: Router) -> Router
```

Підключення маршрутизатора.

Параметри

`router` –

Повертає

```
include_routers(*routers: Router) -> None
```

Attach multiple routers.

Параметри

`routers` –

Повертає

```
resolve_used_update_types(skip_events: Set[str] / None = None) -> List[str]
```

Resolve registered event names

Is useful for getting updates only for registered event types.

Параметри

`skip_events` – skip specified event names

Повертає

set of registered names

Обсервери подій

Попередження: Усі обробники завжди мають бути асинхронними. Ім'я функції обробки не має значення. Назва аргументу події також не важлива, але рекомендується не накладати назву на контекстні дані, оскільки функція не може прийняти два аргументи з однаковою назвою.

Ось список доступних обсерверів і приклади того, як зареєструвати обробники

У цих прикладах використовуються лише обробники реєстрації у стилі декоратора, але якщо вам не подобаються `@decorators`, просто використовуйте `<event type>.register(...)` method instead.

Повідомлення

Увага: Будьте уважні при фільтруванні цієї події

Вам слід очікувати, що ця подія може мати різні набори атрибутів у різних випадках

(Наприклад, текст, стікер та документ завжди мають різні типи вмісту)

Рекомендований спосіб перевірити наявність полів перед використанням, наприклад за допомогою *magic filter*: `F.text` для обробки тексту, `F.sticker` для обробки лише стікерів і тощо.

```
@router.message()
async def message_handler(message: types.Message) -> Any: pass
```

Відредаговане повідомлення

```
@router.edited_message()
async def edited_message_handler(edited_message: types.Message) -> Any: pass
```

Пост на каналі

```
@router.channel_post()
async def channel_post_handler(channel_post: types.Message) -> Any: pass
```

Відредагований пост на каналі

```
@router.edited_channel_post()
async def edited_channel_post_handler(edited_channel_post: types.Message) -> Any: pass
```

Inline запит

```
@router.inline_query()
async def inline_query_handler(inline_query: types.InlineQuery) -> Any: pass
```

Вибраний результат inline запиту

```
@router.chosen_inline_result()
async def chosen_inline_result_handler(chosen_inline_result: types.ChosenInlineResult) ->
    Any: pass
```

Запит зворотної відповіді

```
@router.callback_query()
async def callback_query_handler(callback_query: types.CallbackQuery) -> Any: pass
```

Запит підтвердження доставки

```
@router.shipping_query()
async def shipping_query_handler(shipping_query: types.ShippingQuery) -> Any: pass
```

Запит перед оформленням замовлення

```
@router.pre_checkout_query()
async def pre_checkout_query_handler(pre_checkout_query: types.PreCheckoutQuery) -> Any:
    pass
```

Опитування

```
@router.poll()
async def poll_handler(poll: types.Poll) -> Any: pass
```

Відповідь на опитування

```
@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

```
@router.chat_join_request()
async def chat_join_request_handler(chat_join_request: types.ChatJoinRequest) -> Any:
    ↪ pass
```

Message reaction

```
@router.message_reaction()
async def message_reaction_handler(message_reaction: types.MessageReactionUpdated) ->
    ↪ Any: pass
```

Message reaction count

```
@router.message_reaction_count()
async def message_reaction_count_handler(message_reaction_count: types.
    ↪ MessageReactionCountUpdated) -> Any: pass
```

Chat boost

```
@router.chat_boost()
async def chat_boost_handler(chat_boost: types.ChatBoostUpdated) -> Any: pass
```

Remove chat boost

```
@router.removed_chat_boost()
async def removed_chat_boost_handler(removed_chat_boost: types.ChatBoostRemoved) -> Any:
    ↪ pass
```

Помилки

```
@router.errors()  
async def error_handler(exception: types.ErrorEvent) -> Any: pass
```

Is useful for handling errors from other handlers, error event described [here](#)

Вкладені маршрутизатори

Попередження:

До речі, маршрутизатори можуть бути вкладеними в інші маршрутизатори з деякими обмеженнями:

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)

Приклад:

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)
```

Оновлення

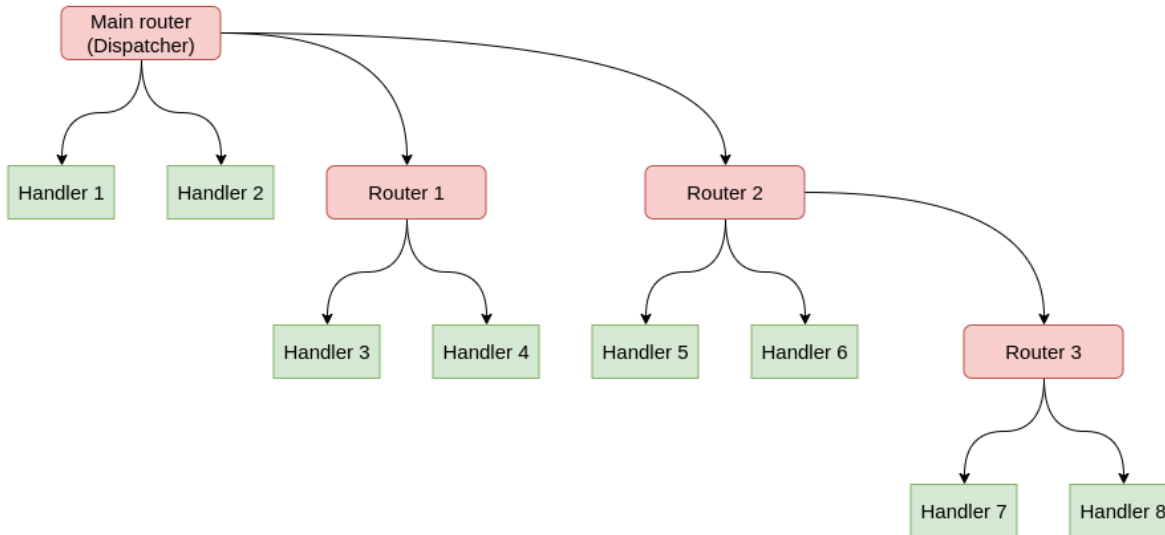
```
@dispatcher.update()  
async def message_handler(update: types.Update) -> Any: pass
```

Попередження: The only root Router (Dispatcher) can handle this type of event.

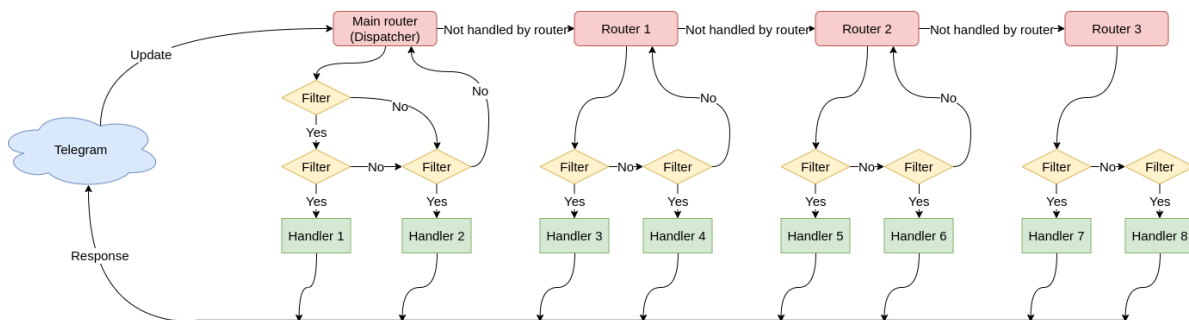
Примітка: Dispatcher already has default handler for this event type, so you can use it for handling all updates that are not handled by any other handlers.

Як це працює?

Наприклад, диспетчер має 2 маршрутизатори, останній маршрутизатор також має один вкладений маршрутизатор:



У цьому випадку потік розповсюдження оновлення матиме вигляд:



2.4.2 Диспетчер

Диспетчер - це кореневий маршрутизатор, і в коді диспетчер може використовуватися безпосередньо для маршрутизації подій або підключення інших маршрутизаторів до диспетчера.

Here is only listed base information about Dispatcher. All about writing handlers, filters and etc. you can find in next pages:

- *Router*
- *Фільтрування подій*

```
class aiogram.dispatcher.dispatcher.Dispatcher(*, storage: BaseStorage / None = None,
                                              fsm_strategy: FSMStrategy =
                                              FSMStrategy.USER_IN_CHAT, events_isolation:
                                              BaseEventIsolation / None = None, disable_fsm:
                                              bool = False, name: str / None = None, **kwargs:
                                              Any)
```

Кореневий маршрутизатор

```
__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
```

Кореневий маршрутизатор

Параметри

- `storage` – Сховище для кінцевого автомату (FSM)
- `fsm_strategy` – Стратегія кінцевого апарату
- `events_isolation` – Ізоляція подій
- `disable_fsm` – Відключення кінцевого апарату, зауважте що при вимкненому кінцевому апараті вам не слід використовувати сховище (кінцевого апарату) та ізоляцію подій
- `kwargs` – Інші аргументи будуть передані обробникам як іменовані аргументи

```
async feed_raw_update(bot: Bot, update: Dict[str, Any], **kwargs: Any) → Any
```

Основна точка входу для подій

Параметри

- `bot` –
- `update` –
- `kwargs` –

```
async feed_update(bot: Bot, update: Update, **kwargs: Any) → Any
```

Основна точка входу для подій. Відповідь цього метода може бути використана для відповіді у Webhook

Параметри

- `bot` –
- `update` –

```
run_polling(*bots: Bot, polling_timeout: int = 10, handle_as_tasks: bool = True, backoff_config:
            BackoffConfig = BackoffConfig(min_delay=1.0, max_delay=5.0, factor=1.3,
            jitter=0.1), allowed_updates: List[str] / _SentinelObject / None = sentinel.UNSET,
            handle_signals: bool = True, close_bot_session: bool = True, **kwargs: Any) → None
```

Запуск кількох ботів з опитуванням

Параметри

- `bots` – Bot instances (one or more)
- `polling_timeout` – Long-polling wait time
- `handle_as_tasks` – Запуск обробки без очікування результату
- `backoff_config` – backoff-retry config

- `allowed_updates` – Список типів подій, які має опрацьовувати ваш бот
- `handle_signals` – handle signals (SIGINT/SIGTERM)
- `close_bot_session` – close bot sessions on shutdown
- `kwargs` – контекстні дані

Повертає

```
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
```

Запуск кількох ботів з опитуванням (асинхронно)

Параметри

- `bots` – Bot instances (one or more)
- `polling_timeout` – Long-polling wait time
- `handle_as_tasks` – Запуск обробки без очікування результату
- `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` – контекстні дані

Повертає

```
async stop_polling() → None
```

Execute this method if you want to stop polling programmatically

Повертає**Просте застосування**

Наприклад:

```
dp = Dispatcher()

@dp.message()
async def message_handler(message: types.Message) -> None:
    await SendMessage(chat_id=message.from_user.id, text=message.text)
```

Включаючи маршрутизатори

Наприклад:

```
dp = Dispatcher()
router1 = Router()
dp.include_router(router1)
```

Обробка подій

Усі оновлення можна передати диспетчеру через `Dispatcher.feed_update(bot=..., update=...)` method:

```
bot = Bot(...)
dp = Dispatcher()

...

result = await dp.feed_update(bot=bot, update=incoming_update)
```

2.4.3 Dependency injection

Dependency injection is a programming technique that makes a class independent of its dependencies. It achieves that by decoupling the usage of an object from its creation. This helps you to follow [SOLID's](#) dependency inversion and single responsibility principles.

How it works in aiogram

For each update `aiogram.dispatcher.dispatcher.Dispatcher` passes handling context data. Filters and middleware can also make changes to the context.

To access contextual data you should specify corresponding keyword parameter in handler or filter. For example, to get `aiogram.fsm.context.FSMContext` we do it like that:

```
@router.message(ProfileCompletion.add_photo, F.photo)
async def add_photo(
    message: types.Message, bot: Bot, state: FSMContext
) -> Any:
    ... # do something with photo
```

Injecting own dependencies

Aiogram provides several ways to complement / modify contextual data.

The first and easiest way is to simply specify the named arguments in `aiogram.dispatcher.dispatcher.Dispatcher` initialization, polling start methods or `aiogram.webhook.aihttp_server.SimpleRequestHandler` initialization if you use webhooks.

```
async def main() -> None:
    dp = Dispatcher(..., foo=42)
    return await dp.start_polling(
        bot, bar="Bazz"
    )
```

Analogy for webhook:

```
async def main() -> None:
    dp = Dispatcher(..., foo=42)
    handler = SimpleRequestHandler(dispatcher=dp, bot=bot, bar="Bazz")
    ... # starting webhook
```

`aiogram.dispatcher.dispatcher.Dispatcher`'s workflow data also can be supplemented by setting values as in a dictionary:

```
dp = Dispatcher(...)
dp["eggs"] = Spam()
```

The middlewares updates the context quite often. You can read more about them on this page:

- *Middlewares*

The last way is to return a dictionary from the filter:

```
from typing import Any, Dict, Optional, Union

from aiogram import Router
from aiogram.filters import Filter
from aiogram.types import Message, User

router = Router(name=__name__)

class HelloFilter(Filter):
    def __init__(self, name: Optional[str] = None) -> None:
        self.name = name

    async def __call__(
        self,
        message: Message,
        event_from_user: User
        # Filters also can accept keyword parameters like in handlers
    ) -> Union[bool, Dict[str, Any]]:
        if message.text.casefold() == "hello":
            # Returning a dictionary that will update the context data
            return {"name": event_from_user.mention_html(name=self.name)}
        return False

@router.message(HelloFilter())
async def my_handler(
    message: Message, name: str # Now we can accept "name" as named parameter
) -> Any:
    return message.answer("Hello, {name}!".format(name=name))
```

...or using *MagicFilter* with `.as_()` method.

2.4.4 Фільтрування подій

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.

Вбудовані фільтри

Ось список вбудованих фільтрів:

Команди

Використання

1. Фільтр єдиного варіанту команд: `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. Обробка команди в публічних чатах, призначених для інших ботів: `Command("command", ignore_mention=True)`
5. Використання об'єкту `aiogram.types.bot_command.BotCommand` як посилання на команду `Command(BotCommand(command="command", description="My awesome command"))`

Попередження: Команда не може містити пробілів чи переносів рядків

```
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)
```

Цей фільтр може бути корисним для обробки команд із текстових повідомлень.

Працює лише з подіями `aiogram.types.message.Message`, що мають `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)
```

Перелік команд (рядки або скомпільовані шаблони регулярних виразів)

Параметри

- **prefix** – Префікс для команди. Префікс завжди складається з одного символу, але тут ви можете передати всі дозволені префікси, наприклад: `"/!"` працюватиме з командами з префіксом `"/"` або `:code:`»!»``.
- **ignore_case** – Ігнорувати регістр (не працює з регулярним виразом, замість цього використовуйте маркери)
- **ignore_mention** – Ігнорувати згадку про бота. За замовчуванням бот не може обробляти команди, призначені для інших ботів

- **magic** – Перевірка об'єкту команди за допомогою магічного фільтра після виконання всіх перевірок

Коли фільтр пройдено, `aiogram.filters.command.CommandObject` буде передано аргументу обробника `command`

```
class aiogram.filters.command.CommandObject(prefix: str = '/', command: str = '', mention: str | None = None, args: str | None = None, regex_match: Match[str] | None = None, magic_result: Any | None = None)
```

Екземпляр цього об'єкта завжди має команду та її префікс. Можна передати обробнику (handler) як аргумент ключового слова **command**

`prefix: str = '/'`

Префікс команди

`command: str = ''`

Команда без префікса та згадки

`mention: str | None = None`

Згадка (за наявності)

`args: str | None = None`

Аргумент команди

`regex_match: Match[str] | None = None`

Буде представлено результат відповідності, якщо команда представлена як регулярний вираз у фільтрі

`magic_result: Any | None = None`

`property mentioned: bool`

Ця команда згадується?

`property text: str`

Створення оригінального тексту з об'єкта

Дозволені обробники (handler)

Дозволені типи оновлень для цього фільтра:

- `message`
- `edited_message`

Зміна статусу користувача в чаті

Використання

Керуйте подіями, які залишають користувачів або приєднуються

```
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): ...
```

(continues on next page)

(continued from previous page)

```
@router.chat_member(ChatMemberUpdatedFilter(IS_NOT_MEMBER >> IS_MEMBER))
async def on_user_join(event: ChatMemberUpdated): ...
```

Або створіть власні умови, використовуючи попередньо визначений набір статусів і переходів.

Explanation

```
class aiogram.filters.chat_member_updated.ChatMemberUpdatedFilter(member_status_changed:
                                                                    _MemberStatusMarker |
                                                                    _MemberStatusGroupMarker
                                                                    /
                                                                    _MemberStatusTransition)

    member_status_changed
```

Ви можете імпортувати з `aiogram.filters` усі доступні варіанти *statuses*, *status group* або *transitions*:

Статуси

ім'я	Опис
CREATOR	Власник чату
ADMINISTRATOR	Адміністратор чату
MEMBER	Учасник чату
RESTRICTED	Обмежений користувач (може бути не учасником)
LEFT	Не є учасником чату
KICKED	Вигнаний адміністраторами учасник

Статуси можна розширити маркером *is_member*, додавши префікс + (для `is_member == True`) або - (для `is_member == False`), наприклад `+RESTRICTED` або `-RESTRICTED`

Групи статусів

Окремі статуси можна комбінувати за допомогою побітового оператора `or`, наприклад `CREATOR | ADMINISTRATOR`

ім'я	Опис
IS_MEMBER	Комбінація статусів (<code>CREATOR ADMINISTRATOR MEMBER +RESTRICTED</code>).
IS_ADMIN	Комбінація статусів (<code>CREATOR ADMINISTRATOR</code>).
IS_NOT_MEMBER	Комбінація статусів (<code>LEFT KICKED -RESTRICTED</code>).

Переходи

Переходи можна визначити за допомогою операторів порозрядного зсуву `>>` і `<<`. Старий статус учасника чату має бути визначений ліворуч для оператора `>>` (праворуч для `<<`), а новий статус має бути вказаний праворуч для `>>` оператор (ліворуч для `<<`)

Напрямок переходу можна змінити за допомогою оператора побітової інверсії: `~JOIN_TRANSITION` призведе до обміну старих і нових статусів.

ім'я	Опис
<code>JOIN_TRANSITION</code>	Означає, що статус змінено з <code>IS_NOT_MEMBER</code> на <code>IS_MEMBER</code> (<code>IS_NOT_MEMBER >> IS_MEMBER</code>)
<code>LEAVE_TRANSITION</code>	Означає, що статус змінено з <code>IS_MEMBER</code> на <code>IS_NOT_MEMBER</code> (<code>~JOIN_TRANSITION</code>)
<code>PROMOTED_TRANSITION</code>	Означає, що статус змінено з <code>(MEMBER RESTRICTED LEFT KICKED) >> ADMINISTRATOR</code> (<code>(MEMBER RESTRICTED LEFT KICKED) >> ADMINISTRATOR</code>)

Примітка: Зауважте, що якщо ви визначаєте об'єднання статусів (через `|`), вам потрібно буде додати дужки для оператора перед використанням оператора зсуву через пріоритети оператора.

Дозволені обробники

Дозволені типи оновлень для цього фільтра:

- `my_chat_member`
- `chat_member`

Магічні фільтри

Примітка: Ця сторінка все ще в розробці. Має багато неправильно сформульованих речень.

Це зовнішній пакет, який підтримується основною командою розробки *aiogram*.

За замовчуванням встановлюється разом з *aiogram* і, також, доступний в [PyPi - magic-filter](#). Це означає, що Ви можете встановити його та використовувати з будь-якими іншими бібліотеками та у власних проектах, незалежно від того встановлено *aiogram* чи ні.

Використання

Пакет **magic_filter** реалізує клас із короткою назвою `magic_filter.F`, тобто **F** можна імпортувати з *aiogram* або *magic_filter*. **F** є псевдонімом для `MagicFilter`.

Примітка: Зауважте, що *aiogram* має невелике розширення для *magic-filter*, і якщо Ви хочете використовувати це розширення, вам слід імпортувати магію з *aiogram* замість пакета *magic_filter*

Об'єкт класу `MagicFilter` можна викликати, підтримує *деякі дії* і запам'ятовує ланцюжок атрибутів і дію, яку слід перевіряти на вимогу.

Тож це означає, що ви можете ланцюжком отримати атрибути, описати прості перевірки даних, а потім викликати отриманий об'єкт, передаючи один об'єкт як аргумент, наприклад, створити ланцюжок атрибутів `F.foo.bar.baz`, а потім додати дію `„F.foo.bar.baz == 'spam'`, після чого викликати отриманий об'єкт - `(F.foo.bar.baz == 'spam').resolve(obj)`

Можливі дії

Об'єкт магічного фільтра підтримує деякі основні логічні операції над атрибутами об'єкта

Атрибут існує, або не «None»

Дії за замовчуванням.

```
F.photo # lambda message: message.photo
```

Перевірка на однаковість

```
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'
```

Перевірка на приналежність

Може використовуватися як метод із назвою `in_` або як оператор `matmul @` з будь-яким ітерованим

```
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'}
```

Перевірка на наявність

```
F.text.contains('foo') # lambda message: 'foo' in message.text
```

Рядок починається/закінчується на

Може застосовуватися лише для текстових атрибутів

```
F.text.startswith('foo') # lambda message: message.text.startswith('foo')
F.text.endswith('bar') # lambda message: message.text.endswith('bar')
```


Перевірка регулярними виразами

```
F.text.regex(r'Hello, .+') # lambda message: re.match(r'Hello, .+', message.text)
```

Власні функції

Приймає будь-яку функцію

```
F.chat.func(lambda chat: chat.id == -42) # lambda message: (lambda chat: chat.id == -42)(message.chat)
```

Інвертування результату

Будь-яка доступна операція може бути інвертована за допомогою побітової інверсії - ~

```
~F.text # lambda message: not message.text
~F.text.startswith('spam') # lambda message: not message.text.startswith('spam')
```

Комбінація

Усі операції можна комбінувати за допомогою побітових і/або операторів - &/|

```
(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')
```

Модифікатори атрибутів - маніпуляції з рядками

Робить текст верхнім або нижнім регістром

Можна використовувати лише з рядковими атрибутами.

```
F.text.lower() == 'test' # lambda message: message.text.lower() == 'test'
F.text.upper().in_({'FOO', 'BAR'}) # lambda message: message.text.upper() in {'FOO', 'BAR'}
↳ 'BAR'
F.text.len() == 5 # lambda message: len(message.text) == 5
```

Отримати результат фільтра як аргумент обробника

Ця частина недоступна безпосередньо в *magic-filter*, але її можна використовувати з *aiogram*

```
from aiogram import F
...
```

(continues on next page)

(continued from previous page)

```
@router.message(F.text.regex(r"^(\d+)$").as_("digits"))
async def any_digits_handler(message: Message, digits: Match[str]):
    await message.answer(html.quote(str(digits)))
```

Використання в aiogram

```
@router.message(F.text == 'hello')
@router.inline_query(F.data == 'button:1')
@router.message(F.text.startswith('foo'))
@router.message(F.content_type.in_({'text', 'sticker'}))
@router.message(F.text.regex(r'\d+'))

...

# Many others cases when you will need to check any of available event attribute
```

MagicData

Використання

1. `MagicData(F.event.from_user.id == F.config.admin_id)` (Зауважте, що `config` слід передати з проміжної програми)

Explanation

```
class aiogram.filters.magic_data.MagicData(magic_data: MagicFilter)
```

Цей фільтр допомагає фільтрувати події з контекстними даними

`magic_data`

Можна імпортувати:

- `from aiogram.filters import MagicData`

Дозволені типи обробників (handler)

Дозволені типи оновлень для цього фільтра:

- `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

Фабрика міток зворотного виклику та фільтрування

`class aiogram.filters.callback_data.CallbackData`

Базовий клас для обгортки мітки зворотного виклику

Цей клас слід використовувати як супер-клас зворотних викликів, визначених користувачем.

Ключове слово класу `prefix` потрібне для визначення префікса, а також аргумент `sep` можна передати для визначення роздільника (за замовчуванням це `:`).

`pack()` → `str`

Генерування рядок мітки зворотного виклику

Повертає

дійсна мітка зворотного виклику для Telegram Bot API

`classmethod unpack(value: str) → T`

Аналіз рядка мітки зворотного виклику

Параметри

`value` – значення з Telegram

Повертає

екземпляр мітки зворотного виклику

`classmethod filter(rule: MagicFilter / None = None) → CallbackQueryFilter`

Створює фільтр для запиту зворотного виклику з правилом

Параметри

`rule` – магічне правило

Повертає

екземпляр фільтру

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

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

Використання

Створення підкласу `CallbackData`:

```
class MyCallback(CallbackData, prefix="my"):
    foo: str
    bar: int
```

Після цього ви можете створити будь-який зворотній виклик на основі цього класу, наприклад:

```
cb1 = MyCallback(foo="demo", bar=42)
cb1.pack() # returns 'my:demo:42'
cb1.unpack('my:demo:42') # returns <MyCallback(foo="demo", bar=42)>
```

Отже... Тепер ви можете використовувати цей клас для створення будь-яких зворотних викликів із визначеною структурою

```
...
# Pass it into the markup
InlineKeyboardButton(
    text="demo",
    callback_data=MyCallback(foo="demo", bar="42").pack() # value should be packed to
↳ string
)
...
```

... і обробляти за певними правилами

```
# 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
)
```

Ще один абстрактний приклад:

```
class Action(str, Enum):
    ban = "ban"
    kick = "kick"
    warn = "warn"

class AdminAction(CallbackData, prefix="adm"):
    action: Action
    chat_id: int
```

(continues on next page)

(continued from previous page)

```

    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,
        ...
    )

```

Відомі обмеження

Дозволені типи та їх підкласи:

- str
- int
- bool
- float
- Decimal (from decimal import Decimal)
- Fraction (from fractions import Fraction)
- UUID (from uuid import UUID)
- Enum (from enum import Enum, лише для переліків рядків)
- IntEnum (from enum import IntEnum, тільки для переліків int)

Примітка: Зауважте, що ціле число Enum завжди має бути підкласом IntEnum через проблеми з синтаксичним аналізом.

Помилки

Ці фільтри можуть бути корисними для обробки помилок у текстових повідомленнях.

```
class aiogram.filters.exception.ExceptionTypeFilter(*exceptions: Type[Exception/])
```

Дозволяє зіставляти винятки за типом

`exceptions`

```
class aiogram.filters.exception.ExceptionMessageFilter(pattern: str | Pattern[str/])
```

Дозвол зіставляти винятки з повідомленням

`pattern`

Дозволені обробники

Дозволені типи оновлення для цього фільтра:

- `error`

Написання власних фільтрів

Фільтри бувають:

- Асинхронною функцією (`async def my_filter(*args, **kwargs): pass`)
- Синхронною функцією (`def my_filter(*args, **kwargs): pass`)
- Анонімною функцією (`lambda event: True`)
- Будь-яким очікуваним об'єктом (awaitable object, об'єкт, який може бути використаний в `await` виразі)
- Підкласом *`aiogram.filters.base.Filter`*
- Екземпляром *`MagicFilter`*

і має повертати `bool` або `dict`. Якщо словник передається як результат фільтра, отримані дані будуть передані до наступних фільтрів і обробника як аргументи ключових слів.

Базовий клас для власних фільтрів

```
class aiogram.filters.base.Filter
```

Якщо Ви хочете зареєструвати власні фільтри, як вбудовані фільтри, Вам потрібно буде написати підклас цього класу з заміною методу `__call__` і додаванням атрибутів фільтра.

```
abstract async __call__(*args: Any, **kwargs: Any) → bool | Dict[str, Any]
```

Цей метод слід перевизначити.

Приймає вхідну подію та має повертати логічне (`bool`) значення або `dict`.

Повертає

`bool` or `Dict[str, Any]`

`update_handler_flags(flags: Dict[str, Any]) → None`

Крім того, якщо ви хочете розширити маркери обробника (handler) за допомогою цього фільтра, вам слід реалізувати цей метод

Параметри

`flags` – існуючі маркери, можна оновити безпосередньо

Приклад власного фільтра

Наприклад, якщо Вам потрібно створити простий текстовий фільтр:

```
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

@router.message(MyFilter("hello"))
async def my_handler(message: Message):
    ...
```

Комбінування фільтрів

Взагалом, усі фільтри можна комбінувати двома способами

Рекомендований спосіб

Якщо Ви вкажете кілька фільтрів поспіль, це буде перевірено умовою «and» :

```
@<router>.message(F.text.startswith("show"), F.text.endswith("example"))
```

Крім того, якщо ви хочете використовувати два альтернативні способи запуску одного обробника (умова «or»), ви можете зареєструвати обробник двічі або більше разів, як вам подобається

```
@<router>.message(F.text == "hi")
@<router>.message(CommandStart())
```

Також іноді Вам потрібно буде інвертувати результат фільтра, наприклад, у вас є фільтр *IsAdmin* і ви хочете перевірити, чи користувач не є адміністратором

```
@<router>.message(~IsAdmin())
```

Інший можливий спосіб

Альтернативним способом є об'єднання за допомогою спеціальних функцій (`and_f()`, `or_f()`, `invert_f()`) з модуля `aiogram.filters`):

```
and_f(F.text.startswith("show"), F.text.endswith("example"))
or_f(F.text(text="hi"), CommandStart())
invert_f(IsAdmin())
and_f(<A>, or_f(<B>, <C>))
```

2.4.5 Long-polling

Long-polling is a technology that allows a Telegram server to send updates in case when you don't have dedicated IP address or port to receive webhooks for example on a developer machine.

To use long-polling mode you should use `aiogram.dispatcher.dispatcher.Dispatcher.start_polling()` or `aiogram.dispatcher.dispatcher.Dispatcher.run_polling()` methods.

Примітка: You can use polling from only one polling process per single Bot token, in other case Telegram server will return an error.

Примітка: If you will need to scale your bot, you should use webhooks instead of long-polling.

Примітка: 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()
```

(continues on next page)

(continued from previous page)

```

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

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.INFO, stream=sys.stdout)
    asyncio.run(main())

```

2.4.6 Webhook

Telegram Bot API supports webhook. If you set webhook for your bot, Telegram will send updates to the specified url. You can use `aiogram.methods.set_webhook.SetWebhook()` method to specify a url and receive incoming updates on it.

Примітка: 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.

Примітка: 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) → None
    Base handler that helps to handle incoming request from aiohttp and propagate it to the Dispatcher
```

Параметри

- `dispatcher` – instance of `aiogram.dispatcher.dispatcher.Dispatcher`
- `handle_in_background` – immediately responds to the Telegram instead of a waiting end of a handler process

```
register(app: None, /, path: str, **kwargs: Any) → None
    Register route and shutdown callback
```

Параметри

- `app` – instance of aiohttp Application
- `path` – route path

- `kwargs` –

`abstract async resolve_bot(request: Request) → Bot`

This method should be implemented in subclasses of this class.

Resolve Bot instance from request.

Параметри

`request` –

Повертає

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) → None
```

Handler for single Bot instance

Параметри

- `dispatcher` – instance of `aiogram.dispatcher.dispatcher.Dispatcher`
- `handle_in_background` – immediately responds to the Telegram instead of a waiting end of handler process
- `bot` – instance of `aiogram.client.bot.Bot`

`async close() → None`

Close bot session

`register(app: None, /, path: str, **kwargs: Any) → None`

Register route and shutdown callback

Параметри

- `app` – instance of aiohttp Application
- `path` – route path
- `kwargs` –

`async resolve_bot(request: Request) → Bot`

This method should be implemented in subclasses of this class.

Resolve Bot instance from request.

Параметри

`request` –

Повертає

Bot instance

```
class aiogram.webhook.aiohttp_server.TokenBasedRequestHandler(dispatcher: Dispatcher,
                                                             handle_in_background: bool =
                                                             True, bot_settings: Dict[str,
                                                             Any] | None = None, **data:
                                                             Any)
```

```
--init__(dispatcher: Dispatcher, handle_in_background: bool = True, bot_settings: Dict[str, Any] /  
None = None, **data: Any) → None
```

Handler that supports multiple bots the context will be resolved from path variable „bot_token“

Примітка: This handler is not recommended in due to token is available in URL and can be logged by reverse proxy server or other middleware.

Параметри

- `dispatcher` – instance of *aiogram.dispatcher.dispatcher.Dispatcher*
- `handle_in_background` – immediately responds to the Telegram instead of a waiting end of handler process
- `bot_settings` – kwargs that will be passed to new Bot instance

```
register(app: None, /, path: str, **kwargs: Any) → None
```

Validate path, register route and shutdown callback

Параметри

- `app` – instance of aiohttp Application
- `path` – route path
- `kwargs` –

```
async resolve_bot(request: Request) → Bot
```

Get bot token from a path and create or get from cache Bot instance

Параметри

`request` –

Повертає

Security

Telegram supports two methods to verify incoming requests that they are from Telegram:

Using a secret token

When you set webhook, you can specify a secret token and then use it to verify incoming requests.

Using IP filtering

You can specify a list of IP addresses from which you expect incoming requests, and then use it to verify incoming requests.

It can be acy using firewall rules or nginx configuration or middleware on application level.

So, aiogram has an implementation of the IP filtering middleware for aiohttp.

```
aiogram.webhook.aiohttp_server.ip_filter_middleware(ip_filter: IPFilter) → Callable[[Request,
                                                                                   Callable[[Request],
                                                                                   Awaitable[StreamResponse]]],
                                                                                   Awaitable[Any]]
```

Параметри

ip_filter –

Повертає

```
class aiogram.webhook.security.IPFilter(ips: Sequence[str | IPv4Network | IPv4Address] | None =
                                         None)
```

```
    __init__(ips: Sequence[str | IPv4Network | IPv4Address] | None = None)
```

Examples**Behind reverse proxy**

In this example we'll use aiohttp as web framework and nginx as reverse proxy.

```
"""
This example shows how to use webhook on behind of any reverse proxy (nginx, traefik,
↳ ingress etc.)
"""
import logging
import sys
from os import getenv

from aiohttp import web

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

# 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
```

(continues on next page)

(continued from previous page)

```

BASE_WEBHOOK_URL = "https://aiogram.dev/"

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

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

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

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

async def on_startup(bot: Bot) -> None:
    # If you have a self-signed SSL certificate, then you will need to send a public
    # certificate to Telegram
    await bot.set_webhook(f"{BASE_WEBHOOK_URL}{WEBHOOK_PATH}", secret_token=WEBHOOK_
    ↪ SECRET)

def main() -> None:
    # Dispatcher is a root router
    dp = Dispatcher()
    # ... and all other routers should be attached to Dispatcher
    dp.include_router(router)

    # Register startup hook to initialize webhook

```

(continues on next page)

(continued from previous page)

```

dp.startup.register(on_startup)

# Initialize Bot instance with a default parse mode which will be passed to all API
↳ calls
bot = Bot(TOKEN, parse_mode=ParseMode.HTML)

# Create aiohttp.web.Application instance
app = web.Application()

# Create an instance of request handler,
# aiogram has few implementations for different cases of usage
# In this example we use SimpleRequestHandler which is designed to handle simple
↳ cases
webhook_requests_handler = SimpleRequestHandler(
    dispatcher=dp,
    bot=bot,
    secret_token=WEBHOOK_SECRET,
)
# Register webhook handler on application
webhook_requests_handler.register(app, path=WEBHOOK_PATH)

# Mount dispatcher startup and shutdown hooks to aiohttp application
setup_application(app, dp, bot=bot)

# And finally start webserver
web.run_app(app, host=WEB_SERVER_HOST, port=WEB_SERVER_PORT)

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

```

When you use nginx as reverse proxy, you should set *proxy_pass* to your aiohttp server address.

```

location /webhook {
    proxy_set_header Host $http_host;
    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)
router = Router()

@router.message(CommandStart())
async def command_start_handler(message: Message) -> None:
    """
    This handler receives messages with `/start` command
    """
```

(continues on next page)

(continued from previous page)

```

    # Most event objects have aliases for API methods that can be called in events'
    context
    # For example if you want to answer to incoming message you can use `message.answer(.
    ..)` alias
    # and the target chat will be passed to :ref:`aiogram.methods.send_message`.
    SendMessage`
    # method automatically or call API method directly via
    # Bot instance: `bot.send_message(chat_id=message.chat.id, ...)`
    await message.answer(f"Hello, {hbold(message.from_user.full_name)}!")

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

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

async def on_startup(bot: Bot) -> None:
    # In case when you have a self-signed SSL certificate, you need to send the
    certificate
    # itself to Telegram servers for validation purposes
    # (see https://core.telegram.org/bots/self-signed)
    # But if you have a valid SSL certificate, you SHOULD NOT send it to Telegram
    servers.
    await bot.set_webhook(
        f"{BASE_WEBHOOK_URL}{WEBHOOK_PATH}",
        certificate=FSInputFile(WEBHOOK_SSL_CERT),
        secret_token=WEBHOOK_SECRET,
    )

def main() -> None:
    # Dispatcher is a root router
    dp = Dispatcher()
    # ... and all other routers should be attached to Dispatcher
    dp.include_router(router)

    # Register startup hook to initialize webhook
    dp.startup.register(on_startup)

    # Initialize Bot instance with a default parse mode which will be passed to all API
    calls

```

(continues on next page)

(continued from previous page)

```

bot = Bot(TOKEN, parse_mode=ParseMode.HTML)

# Create aiohttp.web.Application instance
app = web.Application()

# Create an instance of request handler,
# aiogram has few implementations for different cases of usage
# In this example we use SimpleRequestHandler which is designed to handle simple
↪ cases
webhook_requests_handler = SimpleRequestHandler(
    dispatcher=dp,
    bot=bot,
    secret_token=WEBHOOK_SECRET,
)
# Register webhook handler on application
webhook_requests_handler.register(app, path=WEBHOOK_PATH)

# Mount dispatcher startup and shutdown hooks to aiohttp application
setup_application(app, dp, bot=bot)

# Generate SSL context
context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
context.load_cert_chain(WEBHOOK_SSL_CERT, WEBHOOK_SSL_PRIV)

# And finally start webserver
web.run_app(app, host=WEB_SERVER_HOST, port=WEB_SERVER_PORT, ssl_context=context)

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

```

With using other web framework

You can pass incoming request to aiogram's webhook controller from any web framework you want.

Read more about it in `aiogram.dispatcher.dispatcher.Dispatcher.feed_webhook_update()` or `aiogram.dispatcher.dispatcher.Dispatcher.feed_update()` methods.

```

update = Update.model_validate(await request.json(), context={"bot": bot})
await dispatcher.feed_update(update)

```

Примітка: 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 Кінцевий автомат (FSM)

Кінцевий автомат, кінцевий автоматон, або машина станів (FSM, FSA, finite automaton, state machine) - це математична модель обчислень.

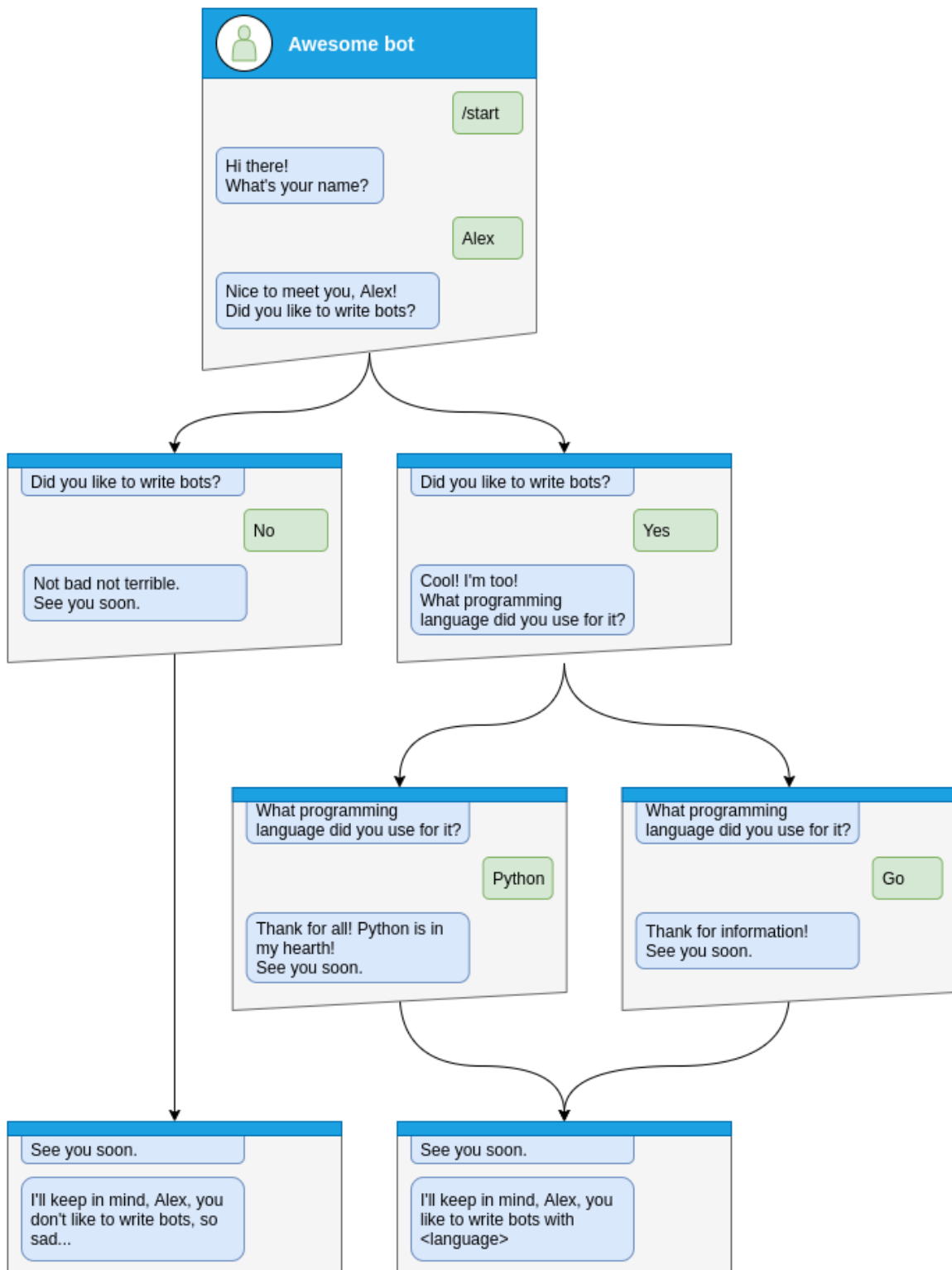
Це абстрактна машина, яка може перебувати в одному зі скінченної кількості станів у будь-який момент часу. Кінцевий автомат може переходити з одного стану в інший у відповідь на деякі вхідні дані; перехід з одного стану в інший називається переходом.

Кінцевий автомат визначається списком його станів, початковим станом і вхідними даними, які запускають кожен перехід.

Джерело: [WikiPedia](#)

Приклад використання

Не всі функції бота можна реалізувати як єдиний обробник (handler), наприклад, якщо Вам потрібно буде збирати деякі дані від користувача в окремих кроках, вам потрібно буде використовувати FSM.



Гайда, подивимось як реалізувати це крок за кроком

Крок за кроком

Перед обробкою будь-яких станів Вам потрібно буде вказати тип станів, які Ви хочете обробляти

```
class Form(StatesGroup):
    name = State()
    like_bots = State()
    language = State()
```

А потім напишіть обробник (handler) для кожного стану окремо від початку діалогу

Тут діалог можна почати лише за допомогою команди `/start`, тому давайте обробимо її та зробимо перехід користувача до стану `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(),
    )
```

Після цього Вам потрібно буде зберегти деякі дані в пам'яті та перейти до наступного кроку.

```
@form_router.message(Form.name)
async def process_name(message: Message, state: FSMContext) -> None:
    await state.update_data(name=message.text)
    await state.set_state(Form.like_bots)
    await message.answer(
        f"Nice to meet you, {html.quote(message.text)}!\nDid you like to write bots?",
        reply_markup=ReplyKeyboardMarkup(
            keyboard=[
                [
                    KeyboardButton(text="Yes"),
                    KeyboardButton(text="No"),
                ]
            ],
            resize_keyboard=True,
        ),
    )
```

На наступних кроках користувач може дати різні відповіді, це може бути «так», «ні» або будь-що інше

Обробка `yes` і скоро нам потрібно буде обробити стан `Form.language`

```
@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(),
    )
```

Обробка `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)
```

І обробка будь-яких інших відповідей

```
@form_router.message(Form.like_bots)
async def process_unknown_write_bots(message: Message) -> None:
    await message.reply("I don't understand you :(")
```

Всі можливі випадки кроку `like_bots` було розглянуто, нумо реалізуємо останній крок

```
@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())
```

І тепер Ви виконали всі кроки на зображенні, але ви можете зробити можливість скасувати діалог, давайте зробимо це за допомогою команди або тексту

```
@form_router.message(Command("cancel"))
@form_router.message(F.text.casefold() == "cancel")
async def cancel_handler(message: Message, state: FSMContext) -> None:
    """
    Allow user to cancel any action
    """
    current_state = await state.get_state()
    if current_state is None:
```

(continues on next page)

(continued from previous page)

```

        return

    logging.info("Cancelling state %r", current_state)
    await state.clear()
    await message.answer(
        "Cancelled.",
        reply_markup=ReplyKeyboardRemove(),
    )

```

Повний приклад

```

1  import asyncio
2  import logging
3  import sys
4  from os import getenv
5  from typing import Any, Dict
6
7  from aiogram import Bot, Dispatcher, F, Router, html
8  from aiogram.enums import ParseMode
9  from aiogram.filters import Command, CommandStart
10 from aiogram.fsm.context import FSMContext
11 from aiogram.fsm.state import State, StatesGroup
12 from aiogram.types import (
13     KeyboardButton,
14     Message,
15     ReplyKeyboardMarkup,
16     ReplyKeyboardRemove,
17 )
18
19 TOKEN = getenv("BOT_TOKEN")
20
21 form_router = Router()
22
23
24 class Form(StatesGroup):
25     name = State()
26     like_bots = State()
27     language = State()
28
29
30 @form_router.message(CommandStart())
31 async def command_start(message: Message, state: FSMContext) -> None:
32     await state.set_state(Form.name)
33     await message.answer(
34         "Hi there! What's your name?",
35         reply_markup=ReplyKeyboardRemove(),
36     )
37
38
39 @form_router.message(Command("cancel"))

```

(continues on next page)

(continued from previous page)

```

40 @form_router.message(F.text.casefold() == "cancel")
41 async def cancel_handler(message: Message, state: FSMContext) -> None:
42     """
43     Allow user to cancel any action
44     """
45     current_state = await state.get_state()
46     if current_state is None:
47         return
48
49     logging.info("Cancelling state %r", current_state)
50     await state.clear()
51     await message.answer(
52         "Cancelled.",
53         reply_markup=ReplyKeyboardRemove(),
54     )
55
56
57 @form_router.message(Form.name)
58 async def process_name(message: Message, state: FSMContext) -> None:
59     await state.update_data(name=message.text)
60     await state.set_state(Form.like_bots)
61     await message.answer(
62         f"Nice to meet you, {html.quote(message.text)}!\nDid you like to write bots?",
63         reply_markup=ReplyKeyboardMarkup(
64             keyboard=[
65                 [
66                     KeyboardButton(text="Yes"),
67                     KeyboardButton(text="No"),
68                 ]
69             ],
70             resize_keyboard=True,
71         ),
72     )
73
74
75 @form_router.message(Form.like_bots, F.text.casefold() == "no")
76 async def process_dont_like_write_bots(message: Message, state: FSMContext) -> None:
77     data = await state.get_data()
78     await state.clear()
79     await message.answer(
80         "Not bad not terrible.\nSee you soon.",
81         reply_markup=ReplyKeyboardRemove(),
82     )
83     await show_summary(message=message, data=data, positive=False)
84
85
86 @form_router.message(Form.like_bots, F.text.casefold() == "yes")
87 async def process_like_write_bots(message: Message, state: FSMContext) -> None:
88     await state.set_state(Form.language)
89
90     await message.reply(
91         "Cool! I'm too!\nWhat programming language did you use for it?",

```

(continues on next page)

(continued from previous page)

```

92     reply_markup=ReplyKeyboardRemove(),
93 )
94
95
96 @form_router.message(Form.like_bots)
97 async def process_unknown_write_bots(message: Message) -> None:
98     await message.reply("I don't understand you :(")
99
100
101 @form_router.message(Form.language)
102 async def process_language(message: Message, state: FSMContext) -> None:
103     data = await state.update_data(language=message.text)
104     await state.clear()
105
106     if message.text.casefold() == "python":
107         await message.reply(
108             "Python, you say? That's the language that makes my circuits light up! "
109         )
110
111     await show_summary(message=message, data=data)
112
113
114 async def show_summary(message: Message, data: Dict[str, Any], positive: bool = True) ->
115 ↪None:
116     name = data["name"]
117     language = data.get("language", "<something unexpected>")
118     text = f"I'll keep in mind that, {html.quote(name)}, "
119     text += (
120         f"you like to write bots with {html.quote(language)}."
121         if positive
122         else "you don't like to write bots, so sad..."
123     )
124     await message.answer(text=text, reply_markup=ReplyKeyboardRemove())
125
126
127 async def main():
128     bot = Bot(token=TOKEN, parse_mode=ParseMode.HTML)
129     dp = Dispatcher()
130     dp.include_router(form_router)
131
132     await dp.start_polling(bot)
133
134 if __name__ == "__main__":
135     logging.basicConfig(level=logging.INFO, stream=sys.stdout)
136     asyncio.run(main())

```

Читайте також

Сховища

Вбудоване сховище

MemoryStorage

```
class aiogram.fsm.storage.memory.MemoryStorage
```

Сховище кінцевого автомату за замовчуванням, зберігає всі дані в `dict` і забуває все під час вимкнення

Попередження: Не рекомендується використовувати на production, оскільки, Ви втратите всі дані під час перезапуску бота

```
__init__() → None
```

RedisStorage

```
class aiogram.fsm.storage.redis.RedisStorage(redis: ~redis.asyncio.client.Redis, key_builder:
~aiogram.fsm.storage.base.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.base.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
```

Параметри

- `redis` – Instance of Redis connection
- `key_builder` – builder that helps to convert contextual key to string
- `state_ttl` – TTL for state records
- `data_ttl` – TTL for data records

```
classmethod from_url(url: str, connection_kwargs: Dict[str, Any] | None = None, **kwargs: Any)
→ RedisStorage
```

Create an instance of `RedisStorage` with specifying the connection string

Параметри

- `url` – for example `redis://user:password@host:port/db`
- `connection_kwargs` – see `redis` docs
- `kwargs` – arguments to be passed to `RedisStorage`

Повертаєекземпляр класу *RedisStorage***MongoStorage****KeyBuilder**

Keys inside Redis and Mongo storages can be customized via key builders:

```
class aiogram.fsm.storage.base.KeyBuilder
```

Base class for key builder.

```
abstract build(key: StorageKey, part: Literal['data', 'state', 'lock'] / None = None) → str
```

Build key to be used in storage's db queries

Параметри

- **key** – contextual key
- **part** – part of the record

Повертає

key to be used in storage's db queries

```
class aiogram.fsm.storage.base.DefaultKeyBuilder(*, prefix: str = 'fsm', separator: str = ':',
                                                with_bot_id: bool = False,
                                                with_business_connection_id: bool = False,
                                                with_destiny: bool = False)
```

Simple key builder with default prefix.

Generates a colon-joined string with prefix, chat_id, user_id, optional bot_id, business_connection_id, destiny and field.

Format:

```
<prefix>:<bot_id?>:<business_connection_id?>:<chat_id>:<user_id>:<destiny?>:<field?>
```

```
build(key: StorageKey, part: Literal['data', 'state', 'lock'] / None = None) → str
```

Build key to be used in storage's db queries

Параметри

- **key** – contextual key
- **part** – part of the record

Повертає

key to be used in storage's db queries

Написання власних сховищ

```
class aiogram.fsm.storage.base.BaseStorage
```

Основний клас для всіх сховищ кінцевого автомату

```
abstract async set_state(key: StorageKey, state: str | State | None = None) → None
```

Установити стан для вказаного ключа

Параметри

- `key` – ключ сховища
- `state` – новий стан

```
abstract async get_state(key: StorageKey) → str | None
```

Отримання стану ключа

Параметри

`key` – ключ сховища

Повертає

поточний стан

```
abstract async set_data(key: StorageKey, data: Dict[str, Any]) → None
```

Запис даних (заміна)

Параметри

- `key` – ключ сховища
- `data` – нові дані

```
abstract async get_data(key: StorageKey) → Dict[str, Any]
```

Отримання поточних даних для ключа

Параметри

`key` – ключ сховища

Повертає

нинішні дані

```
async update_data(key: StorageKey, data: Dict[str, Any]) → Dict[str, Any]
```

Дата оновлення в сховищі для ключа (наприклад, `dict.update`)

Параметри

- `key` – ключ сховища
- `data` – неповні дані

Повертає

нові дані

```
abstract async close() → None
```

Закриття сховища (підключення до бази даних, файлу тощо)

Майстер сцен

Нове в версії 3.2.

Попередження: Дана фіча є експериментальною, тому у наступних оновленнях може змінюватись.

Базовий інтерфейс **aiogram**-у простий та потужний у використанні, що дозволяє реалізувати прості взаємодії, такі як обробка команд або повідомлень і відповідей. Однак деякі завдання вимагають поетапного діалогу між користувачем і ботом. Ось де сцени вступають у гру.

Що ж таке сцени?

Сцена в **aiogram** схожа на абстрактний, ізольований простір імен або кімнату, до якої користувач може потрапити за допомогою коду. Коли користувач перебуває в межах Сцени, більшість інших глобальних команд або обробників повідомлень пропускаються, якщо тільки вони не призначені для роботи поза Сценами. Сцени забезпечують структуру для більш складних взаємодій, ефективно ізолюючи та керуючи контекстами для різних етапів розмови. Вони дозволяють більш організовано контролювати та керувати розмовою.

Життєвий цикл

У кожен сцену можна «увійти», «покинути» або «вийти», що забезпечує чіткі переходи між різними етапами розмови. Наприклад, у багатоетапній взаємодії заповнення форми кожен крок може бути сценою - бот направляє користувача від однієї сцени до наступної, коли вони надають необхідну інформацію.

Слухачі подій

Сцени мають власні хуки, які є слухачами команд або повідомлень, які діють лише тоді, коли користувач знаходиться всередині сцени. Ці хуки реагують на дії користувача, коли користувач перебуває «всередині» Сцени, надаючи відповіді або дії, відповідні цьому контексту. Коли користувач переходить від однієї сцени до іншої, дії та відповіді відповідно змінюються, оскільки користувач тепер взаємодіє з групою слухачів у новій сцені. Ці «специфічні для сцени» хуки або слухачі, відірвані від глобального контексту прослуховування, дозволяють більш оптимізовану та організовану взаємодію бот-користувач.

Взаємодія

Кожна сцена схожа на самодостатній світ із взаємодіями, визначеними в межах цієї сцени. Таким чином, лише обробники, визначені в конкретній сцені, реагуватимуть на введення користувача протягом життєвого циклу цієї сцени.

Переваги

Сцени можуть допомогти керувати більш складними робочими процесами взаємодії та забезпечити більш інтерактивні та динамічні діалоги між користувачем і ботом. Це забезпечує велику гнучкість у обробці багатоетапних взаємодій або розмов з користувачами.

Як це використовувати?

Наприклад, у нас є тестовий бот, який задає користувачеві серію запитань, а потім відображає результати - назвемо його гра-вікторина.

Почнемо з моделей даних. У цьому прикладі прості моделі даних використовуються для представлення запитань і відповідей, у реальному житті ви, ймовірно, використовували б базу даних для зберігання даних.

Listing 3: Запитання

```
@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_
→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),
```

(continues on next page)

(continued from previous page)

```

        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),
    ],
),
]

```

Потім нам потрібно створити клас Scene, який представлятиме сцену вікторини:

Примітка: Іменованний аргумент, переданий у визначення класу, описує ім'я сцени - те саме, що стан сцени.

Listing 4: Сцена вікторини

```
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.
    """
```

Також нам потрібно визначити обробник, який допоможе запустити вікторину:

Listing 5: Обробник для запуску вікторини

```
quiz_router = Router(name=__name__)
# Add handler that initializes the scene
quiz_router.message.register(QuizScene.as_handler(), Command("quiz"))
```

Після визначення сцени нам потрібно зареєструвати її в SceneRegistry:

Listing 6: Реєстрація сцени

```
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
```

Отже, тепер ми можемо реалізувати логіку гри-вікторини, кожне запитання надсилається користувачеві одне за одним, а відповідь користувача перевіряється в кінці всіх запитань.

Тепер нам потрібно написати точку входу для обробника запитань:

Listing 7: Точка входу обробника запитань

```
@on.message.enter()
async def on_enter(self, message: Message, state: FSMContext, step: int | None = 0) -
-> Any:
    """
    Method triggered when the user enters the quiz scene.

    It displays the current question and answer options to the user.

    :param message:
```

(continues on next page)

(continued from previous page)

```

: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),
)

```

Після входу в сцену ми маємо очікувати відповіді користувача, тому нам потрібно написати для неї обробник, цей обробник має очікувати текстове повідомлення, зберегти відповідь і повторно виконати обробник запитання для наступного запитання:

Listing 8: Обробник відповідей

```

@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)

```

Коли користувач відповідає невідомим повідомленням, ми повинні знову очікувати текстове повідом-

лення:

Listing 9: Невідомий обробник повідомлень

```
@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.")
```

Після відповіді на всі запитання ми маємо показати результати користувачеві, як ви можете бачити в коді нижче, ми використовуємо `await self.wizard.exit()`, щоб вийти зі сцени, коли список запитань у `QuizScene»` закінчено `.on_enter` обробник.

Це означає, що нам потрібно написати обробник виходу, щоб показати результати користувачеві:

Listing 10: Обробник показу результатів

```
@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 == quiz.correct_answer
        if is_correct:
            correct += 1
            icon = ""
        else:
            incorrect += 1
            icon = ""
        if answer is None:
            answer = "no answer"
```

(continues on next page)

(continued from previous page)

```

        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({})

```

Також ми можемо виконати дії для виходу з вікторини або повернення до попереднього запитання:

Listing 11: Обробник виходу

```

@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: Обробник дії «повернутись»

```

@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"]

```

(continues on next page)

(continued from previous page)

```

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)

```

Тепер ми можемо запустити бота і протестувати гру-вікторину:

Listing 13: Запустіть бота

```

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`

```

Зберемо все разом

Listing 14: Приклад вікторини

```

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

```

(continues on next page)

(continued from previous page)

```

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_
↪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"),
        ],
    ),
    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=[

```

(continues on next page)

(continued from previous page)

```

        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) -
    ➔ 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!")

```

(continues on next page)

(continued from previous page)

```

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
    ↪ 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 == 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(

```

(continues on next page)

(continued from previous page)

```

        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:
    """

```

(continues on next page)

(continued from previous page)

```

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
    ↪ 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(
        "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

```

(continues on next page)

(continued from previous page)

```

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 quiz_scene:create_dispatcher --log-level info --token BOT_
    ↪TOKEN`

```

Компоненти

- *aiogram.fsm.scene.Scene* - представляє сцену, містить обробники
- *aiogram.fsm.scene.SceneRegistry* - контейнер для всіх сцен у боті, використовується для реєстрації сцен та їх вирішення за назвою
- *aiogram.fsm.scene.ScenesManager* - керує сценами для кожного користувача, використовується для входу, виходу та вирішення поточної сцени для користувача
- *aiogram.fsm.scene.SceneConfig* - конфігурація сцени, використовується для налаштування сцени
- *aiogram.fsm.scene.SceneWizard* - майстер сцени, який використовується для взаємодії з користувачем у сцені з активного обробника сцени
- Markers - маркер для обробників сцен, використовується для позначення обробників сцен

```
class aiogram.fsm.scene.Scene(wizard: SceneWizard)
```

Представляє крок в діалозі.

Сцена — це певний стан розмови, де можуть відбуватися певні дії.

Кожна сцена має набір фільтрів, які визначають, коли вона має бути запущена, і набір обробників, які визначають дії, які мають виконуватися, коли сцена активна.

Примітка: Цей клас не призначений для безпосереднього використання. Замість цього слід створити підкласи для визначення власних сцен.

```
classmethod add_to_router(router: Router) → None
```

Додає сцену до заданого маршрутизатора.

Параметри

router –

Повертає

```
classmethod as_handler(**kwargs: Any) → Callable[[...], Any]
```

Створить обробник точки входу для сцени, який можна використовувати для спрощення обробника, який запускає сцену.

```
>>> router.message.register(MyScene.as_handler(), Command("start"))
```

```
classmethod as_router(name: str / None = None) → Router
```

Returns the scene as a router.

Повертає

новий роутер

```
class aiogram.fsm.scene.SceneRegistry(router: Router, register_on_add: bool = True)
```

Клас, який представляє реєстр для сцен.

```
add(*scenes: Type[Scene], router: Router / None = None) → None
```

Цей метод додає вказані сцени до реєстру та додатково реєструє їх на маршрутизаторі.

Якщо сцена з таким самим станом уже існує в реєстрі, виникає `SceneException`.

Попередження: 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.

Параметри

- **scenes** – Параметр змінної довжини, який приймає один або кілька типів сцен. Ці сцени є екземплярами класу `Scene`.
- **router** – Додатковий параметр, який визначає маршрутизатор, до якого слід додати сцени.

Повертає

None

```
get(scene: Type[Scene] / str / None) → Type[Scene]
```

Цей метод повертає зареєстрований об'єкт `Scene` для вказаної сцени. Параметром сцени може бути або об'єкт `Scene`, або рядок, що представляє назву сцени. Якщо надається об'єкт `Scene`, атрибут стану об'єкта `SceneConfig`, пов'язаного з об'єктом `Scene`, використовуватиметься як ім'я сцени. Якщо вказано `None` або недійсний тип, буде викликано `SceneException`.

Якщо вказану сцену не зареєстровано в об'єкті `SceneRegistry`, буде породжено помилку `SceneException`.

Параметри

scene – Об'єкт `Scene` або рядок, що представляє назву сцени.

Повертає

Зареєстрований об'єкт `Scene`, що відповідає даному параметру сцени.

```
register(*scenes: Type[Scene]) → None
```

Реєструє одну або кілька сцен у `SceneRegistry`.

Параметри

`scenes` – Один або кілька класів сцен для реєстрації.

Повертає

`None`

```
class aiogram.fsm.scene.ScenesManager(registry: SceneRegistry, update_type: str, event: TelegramObject, state: FSMContext, data: Dict[str, Any])
```

Клас `ScenesManager` відповідає за керування сценами в програмі. Він надає методи входу та виходу зі сцен, а також відновлення активної сцени.

```
async close(**kwargs: Any) → None
```

Метод `Close` використовується для виходу з поточної активної сцени в `ScenesManager`.

Параметри

`kwargs` – Додаткові аргументи ключового слова, передані в метод виходу сцени.

Повертає

`None`

```
async enter(scene_type: Type[Scene] | str | None, _check_active: bool = True, **kwargs: Any) → None
```

Виходить на вказану сцену.

Параметри

- `scene_type` – Додатково `Type[Scene]` або `str`, що представляє тип сцени, який потрібно ввести.
- `_check_active` – Необов'язковий параметр, що вказує, чи перевіряти наявність активної сцени для виходу перед входом у нову сцену. За замовчуванням значення `True`.
- `kwargs` – Додаткові іменовані аргументи для передачі в метод `wizard.enter()` сцени.

Повертає

`None`

```
class aiogram.fsm.scene.SceneConfig(state: 'Optional[str]', handlers: 'List[HandlerContainer]', actions: 'Dict[SceneAction, Dict[str, CallableObject]]', reset_data_on_enter: 'Optional[bool]' = None, reset_history_on_enter: 'Optional[bool]' = None, callback_query_without_state: 'Optional[bool]' = None)
```

```
actions: Dict[SceneAction, Dict[str, CallableObject]]
```

Дії сцени

```
callback_query_without_state: bool | None = None
```

Створювати обробники кнопок без перевірки стану поточної сцени

```
handlers: List[HandlerContainer]
```

Обробники сцени

```
reset_data_on_enter: bool | None = None
```

Скинути дані сцени після входу

```
reset_history_on_enter: bool | None = None
```

Скинути історію сцени під час входу

state: str | None

Стан сцени

```
class aiogram.fsm.scene.SceneWizard(scene_config: SceneConfig, manager: ScenesManager, state:
    FSMContext, update_type: str, event: TelegramObject, data:
    Dict[str, Any])
```

Клас, який представляє майстер сцен.

Екземпляр цього класу передається кожній сцені як параметр. Отже, ви можете використовувати його для переходу між сценами, отримання та встановлення даних тощо.

Примітка: Цей клас не призначений для безпосереднього використання. Натомість його слід використовувати як параметр у конструкторі сцени.

async back(**kwargs: Any) → None

Цей метод використовується для повернення до попередньої сцени.

Параметри

kwargs – Аргументи ключових слів, які можна передати в метод.

Повертає

None

async clear_data() → None

Очищає дані.

Повертає

None

async enter(**kwargs: Any) → None

Метод Enter використовується для переходу в сцену в класі SceneWizard. Він встановлює стан, очищає дані та історію, якщо вказано, і запускає введення події сцени.

Параметри

kwargs – Додаткові іменовані аргументи.

Повертає

None

async exit(**kwargs: Any) → None

Вийти з поточної сцени та перейти до стандартного стану чи сцени.

Параметри

kwargs – Додаткові іменовані аргументи.

Повертає

None

async get_data() → Dict[str, Any]

Цей метод повертає дані, що зберігаються в поточному стані.

Повертає

Словник, що містить дані, що зберігаються в стані сцени.

async goto(scene: Type[Scene] / str, **kwargs: Any) → None

Метод goto переходить до нової сцени. Спочатку він викликає метод *leave*, щоб виконати будь-яке необхідне очищення в поточній сцені, а потім викликає подію *enter*, щоб увійти до вказаної сцени.

Параметри

- `scene` – Сцена для переходу. Може бути екземпляром *Scene* або рядком, що представляє сцену.
- `kwargs` – Додаткові іменовані аргументи для точки входу до *enter* менеджера сцен.

Повертає

None

```
async leave(_with_history: bool = True, **kwargs: Any) → None
```

Залишає поточну сцену. Цей метод використовується для виходу зі сцени та переходу до наступної сцени.

Параметри

- `_with_history` – Чи включати історію в знімок. За замовчуванням значення True.
- `kwargs` – Додаткові іменовані аргументи.

Повертає

None

```
async retake(**kwargs: Any) → None
```

Цей метод дозволяє повторно увійти до поточної сцени.

Параметри

`kwargs` – Додаткові іменовані аргументи для передачі до сцени.

Повертає

None

```
async set_data(data: Dict[str, Any]) → None
```

Встановлює налаштування дані в поточний стан.

Параметри

`data` – Словник, що містить налаштування дані, які потрібно встановити в поточному стані.

Повертає

None

```
async update_data(data: Dict[str, Any] | None = None, **kwargs: Any) → Dict[str, Any]
```

Цей метод оновлює дані, що зберігаються в поточному стані

Параметри

- `data` – Додатковий словник даних для оновлення.
- `kwargs` – Додаткові пари ключ-значення даних для оновлення.

Повертає

Словник оновлених даних

Маркери

Маркери подібні до механізму реєстрації подій Router, але вони використовуються для позначення обробників сцени в класі Scene.

Його можна імпортувати з `from aiogram.fsm.scene import on` і слід використовувати як декоратор.

Дозволені типи подій:

- 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

Кожен тип події можна відфільтрувати так само, як і в маршрутизаторі.

Також кожен тип події можна позначити як точку входу в сцену, точку виходу або точку переходу.

Якщо ви хочете позначити, що до сцени можна потрапити з повідомлення або ін-лайн кнопки, вам слід використовувати маркер `on.message` або `on.inline_query`:

```
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
```

Сцени мають три точки для переходів:

- Точка входу - коли користувач входить до сцени
- Точка переходу - коли користувач переходить до іншої сцени
- Точка виходу - коли користувач завершує сцену

2.4.8 Проміжні програми

aiogram надає потужний механізм для налаштування обробників(handler) подій через проміжні програми.

Проміжні програми у фреймворку для ботів виглядають як механізм проміжних програм у веб-фреймворках, таких як [aiohttp](#), [fastapi](#), [Django](#) тощо) з невеликою різницею – тут реалізовано два рівні проміжного програмних програм (до та після фільтрів).

Примітка: Проміжна програма — це функція, яка запускається під час кожної події, отриманої від Telegram Bot API у багатьох точках процесу обробки.

Основні поняття

Більшість книг та Інтернет-джерел стверджують:

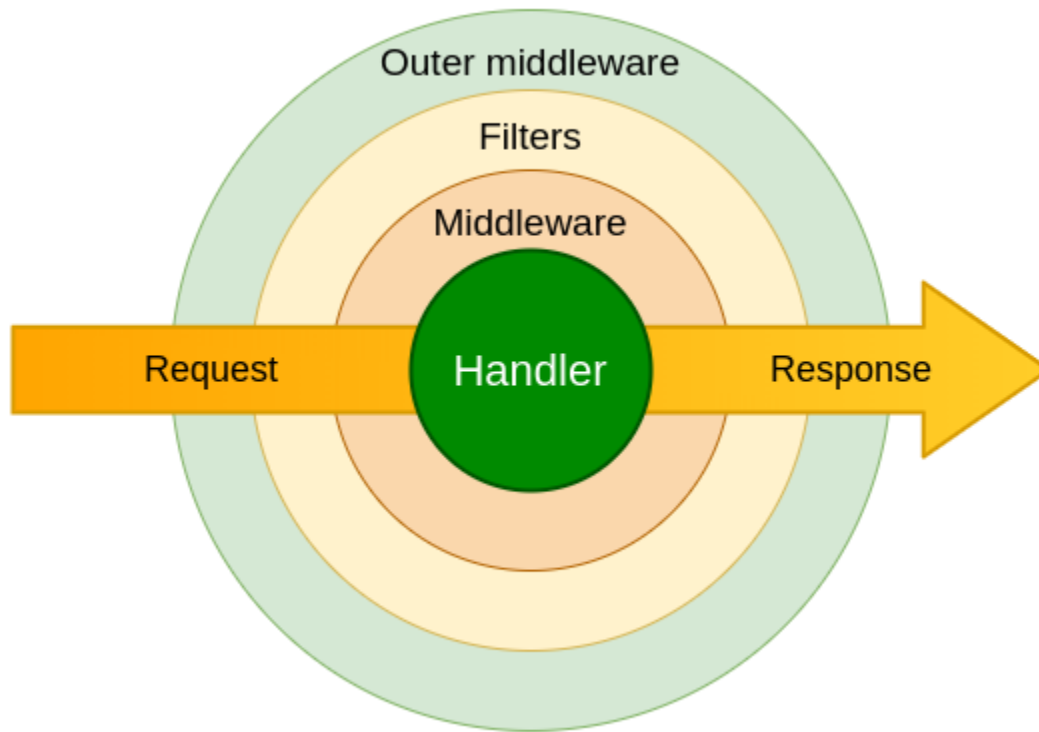
Проміжна програма — це програма, багаторазового використання, що використовує шаблони та фреймворки для ліквідування розриву між функціональними вимогами додатків і основними операційними системами, стеками мережевих протоколів і базами даних.

Проміжна програма може змінювати, розширювати або відхиляти подію обробки у багатьох точках процесу обробки.

Основи

Екземпляр проміжної програми можна застосувати для кожного типу події Telegram (оновлення, повідомлення тощо) у двох місцях

1. Зовнішня область - перед обробкою фільтрами (`<router>.<event>.outer_middleware(...)`)
2. Внутрішня область – після обробки фільтрами, але перед обробником (handler) (`<router>.<event>.middleware(...)`)



Увага: Проміжна програма має бути підкласом `BaseMiddleware` (`from aiogram import BaseMiddleware`) або будь-якою асинхронною функцією

Специфікація аргументів

```
class aiogram.dispatcher.middlewares.base.BaseMiddleware
```

Основа: ABC

Узагальнений клас проміжних програм

```
abstract async __call__(handler: Callable[[TelegramObject, Dict[str, Any]], Awaitable[Any]],
                        event: TelegramObject, data: Dict[str, Any]) → Any
```

Виконання проміжної програми

Параметри

- **handler** – Обробник (handler), обгорнутий у ланцюжок проміжних програм
- **event** – Вхідна подія (підклас `aiogram.types.base.TelegramObject`)
- **data** – Контекстні дані. Будуть зіставлені з аргументами обробника

Повертає

Any

Приклади

Небезпека: 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)`.

Класово орієнтований

```
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)
```

і тоді

```
router = Router()
router.message.middleware(CounterMiddleware())
```

Функціонально-орієнтований

```
@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)
```

Факти

1. Проміжні програми із зовнішньої області викликатимуться під час кожної вхідної події
2. Проміжні програми із внутрішньої області викликатимуться лише після проходження фільтрів
3. Внутрішні проміжні програми викликають тип події `aiogram.types.update.Update`, через те, що всі вхідні оновлення надходять до обробника (handler) певного типу подій через вбудований обробник (handler) оновлень

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*) → 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.

`exception aiogram.exceptions.DetailedAiogramError(message: str)`

Base exception for all aiogram errors with detailed message.

`exception aiogram.exceptions.CallbackAnswerException`

Exception for callback answer.

`exception aiogram.exceptions.SceneException`

Exception for scenes.

`exception aiogram.exceptions.UnsupportedKeywordArgument(message: str)`

Exception raised when a keyword argument is passed as filter.

`exception aiogram.exceptions.TelegramAPIError(method: TelegramMethod, message: str)`

Base exception for all Telegram API errors.

`exception aiogram.exceptions.TelegramNetworkError(method: TelegramMethod, message: str)`

Base exception for all Telegram network errors.

`exception aiogram.exceptions.TelegramRetryAfter(method: TelegramMethod, message: str,
retry_after: int)`

Exception raised when flood control exceeds.

`exception aiogram.exceptions.TelegramMigrateToChat(method: TelegramMethod, message: str,
migrate_to_chat_id: int)`

Exception raised when chat has been migrated to a supergroup.

`exception aiogram.exceptions.TelegramBadRequest(method: TelegramMethod, message: str)`

Exception raised when request is malformed.

`exception aiogram.exceptions.TelegramNotFound(method: TelegramMethod, message: str)`

Exception raised when chat, message, user, etc. not found.

`exception aiogram.exceptions.TelegramConflictError(method: TelegramMethod, message: str)`

Exception raised when bot token is already used by another application in polling mode.

`exception aiogram.exceptions.TelegramUnauthorizedError(method: TelegramMethod, message: str)`

Exception raised when bot token is invalid.

`exception aiogram.exceptions.TelegramForbiddenError(method: TelegramMethod, message: str)`

Exception raised when bot is kicked from chat or etc.

`exception aiogram.exceptions.TelegramServerError(method: TelegramMethod, message: str)`

Exception raised when Telegram server returns 5xx error.

`exception aiogram.exceptions.RestartingTelegram(method: TelegramMethod, message: str)`

Exception raised when Telegram server is restarting.

It seems like this error is not used by Telegram anymore, but it's still here for backward compatibility.

Currently, you should expect that Telegram can raise `RetryAfter` (with timeout 5 seconds)

error instead of this one.

```
exception aiogram.exceptions.TelegramEntityTooLarge(method: TelegramMethod, message: str)
```

Exception raised when you are trying to send a file that is too large.

```
exception aiogram.exceptions.ClientDecodeError(message: str, original: Exception, data: Any)
```

Exception raised when client can't decode response. (Malformed response, etc.)

2.4.10 Маркери

Маркери це, так звані мітки для обробників, які можна використовувати *міddlварax* або в спеціальних *утилітах* щоб провести класифікацію обробників.

Маркери можна додати до обробника через *декоратори*, *реєстрацію обробників* або *фільтри*.

Через декоратори

Наприклад, відмітимо обробник маркером *chat_action*

```
from aiogram import flags

@flags.chat_action
async def my_handler(message: Message)
```

Або просто для рейт-ліміту чи чогось іншого

```
from aiogram import flags

@flags.rate_limit(rate=2, key="something")
async def my_handler(message: Message)
```

Через метод реєстрації обробника

```
@router.message(..., flags={'chat_action': 'typing', 'rate_limit': {'rate': 5}})
```

Через фільтри

```
class Command(Filter):
    ...

    def update_handler_flags(self, flags: Dict[str, Any]) -> None:
        commands = flags.setdefault("commands", [])
        commands.append(self)
```

Використовувати в міddlварах

```
aiogram.dispatcher.flags.check_flags(handler: HandlerObject | Dict[str, Any], magic: MagicFilter)
    → Any
```

Check flags via magic filter

Параметри

- **handler** – handler object or data
- **magic** – instance of the magic

Повертає

the result of magic filter check

```
aiogram.dispatcher.flags.extract_flags(handler: HandlerObject | Dict[str, Any]) → Dict[str, Any]
```

Extract flags from handler or middleware context data

Параметри

handler – handler object or data

Повертає

dictionary with all handler flags

```
aiogram.dispatcher.flags.get_flag(handler: HandlerObject | Dict[str, Any], name: str, *, default:
    Any | None = None) → Any
```

Get flag by name

Параметри

- **handler** – handler object or data
- **name** – name of the flag
- **default** – default value (None)

Повертає

value of the flag or default

Приклад в міddlварах

```
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)
```

Використання в утилітах

Наприклад, ви можете зібрати всі зареєстровані команди з описом обробника, а потім його можна використовувати для створення довідки щодо команд

```
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 Обробники на основі класів

Обробник (handler) — це корутина, яка приймає подію з контекстними даними та повертає відповідь.

В **aiogram** це може бути більш, ніж просто асинхронна функція, це дозволяє вам використовувати класи, які можна використовувати як обробники подій Telegram для структурування ваших обробників подій і повторного використання коду за допомогою унаслідування та розширення.

Нижче наведено кілька обробників на основі класів, які вам потрібно використовувати у своїх власних обробниках:

BaseHandler

Базовий обробник є загальним абстрактним класом і повинен використовуватися в усіх інших обробниках на основі класу.

```
Import: from aiogram.handlers import BaseHandler
```

За замовчуванням вам потрібно буде перевизначити лише метод `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.

Приклад

```
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)`

Це базовий клас для обробників запитів зворотного виклику.

Приклад:

```
from aiogram.handlers import CallbackQueryHandler

...

@router.callback_query()
class MyHandler(CallbackQueryHandler):
    async def handle(self) -> Any: ...
```

property from_user: *User*

Псевдонім для *event.from_user*

property message: *MaybeInaccessibleMessage* | None

Псевдонім для *event.message*

property callback_data: str | None

Псевдонім для *event.data*

ChosenInlineResultHandler

Це базовий клас для обробників вибраних inline результатів.

Просте застосування

```
from aiogram.handlers import ChosenInlineResultHandler

...

@router.chosen_inline_result()
class MyHandler(ChosenInlineResultHandler):
    async def handle(self) -> Any: ...
```

Розширення

Цей базовий обробник є підкласом *BaseHandler* з деякими розширеннями:

- `self.chat` це псевдонім для `self.event.chat`
- `self.from_user` це псевдонім для `self.event.from_user`

ErrorHandler

Це базовий клас для обробників помилок.

Просте застосування

```
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
        )
```

Розширення

Цей базовий обробник є підкласом *BaseHandler* з деякими розширеннями:

- `self.exception_name` це псевдонім для `self.event.__class__.__name__`
- `self.exception_message` це псевдонім для `str(self.event)`

InlineQueryHandler

Це базовий клас для обробників inline запитів.

Просте застосування

```
from aiogram.handlers import InlineQueryHandler

...

@router.inline_query()
class MyHandler(InlineQueryHandler):
    async def handle(self) -> Any: ...
```

Розширення

Цей базовий обробник є підкласом *BaseHandler* з деякими розширеннями:

- `self.chat` це псевдонім для `self.event.chat`
- `self.query` це псевдонім для `self.event.query`

MessageHandler

Це базовий клас для обробників повідомлень.

Просте застосування

```
from aiogram.handlers import MessageHandler

...

@router.message()
class MyHandler(MessageHandler):
    async def handle(self) -> Any:
        return SendMessage(chat_id=self.chat.id, text="PASS")
```

Розширення

This base handler is subclass of *BaseHandler* with some extensions:

- `self.chat` це псевдонім для `self.event.chat`
- `self.from_user` це псевдонім для `self.event.from_user`

PollHandler

Це базовий клас для обробників опитувань.

Просте застосування

```
from aiogram.handlers import PollHandler

...

@router.poll()
class MyHandler(PollHandler):
    async def handle(self) -> Any: ...
```

Розширення

Цей базовий обробник є підкласом *BaseHandler* з деякими розширеннями:

- `self.question` це псевдонім для `self.event.question`
- `self.options` це псевдонім для `self.event.options`

PreCheckoutQueryHandler

!!! Це базовий клас для обробників запитів перед оформленням замовлення.

Просте застосування

```
from aiogram.handlers import PreCheckoutQueryHandler

...

@router.pre_checkout_query()
class MyHandler(PreCheckoutQueryHandler):
    async def handle(self) -> Any: ...
```

Розширення

Цей базовий обробник є підкласом *BaseHandler* з деякими розширеннями:

- `self.from_user` псевдонім для `self.event.from_user`

ShippingQueryHandler

!!! Це базовий клас для обробників запитів підтвердження доставки.

Просте застосування

```
from aiogram.handlers import ShippingQueryHandler

...

@router.shipping_query()
class MyHandler(ShippingQueryHandler):
    async def handle(self) -> Any: ...
```

Розширення

Цей базовий обробник є підкласом *BaseHandler* з деякими розширеннями:

- `self.from_user` псевдонім для `self.event.from_user`

ChatMemberHandler

Це базовий клас для подій оновлення статусу учасника чату.

Просте застосування

```
from aiogram.handlers import ChatMemberHandler

...

@router.chat_member()
@router.my_chat_member()
class MyHandler(ChatMemberHandler):
    async def handle(self) -> Any: ...
```

Розширення

Цей базовий обробник є підкласом *BaseHandler* з деякими розширеннями:

- `self.chat` псевдонім для `self.event.chat`

2.5 Утиліти

2.5.1 Конструктор клавіатури

Конструктор клавіатури допомагає динамічно генерувати розмітку

Примітка: Зауважте, що якщо у вас є статична розмітка, найкраще визначити її явно, а не використовувати конструктор, але якщо у вас є конфігурація динамічної розмітки, сміливо використовуйте конструктор на свій розсуд.

Приклад використання

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

Попередження: Note that you can't attach reply keyboard builder to inline keyboard builder and vice versa

Клавіатура під повідомленням(Inline Keyboard)

```
class aiogram.utils.keyboard.InlineKeyboardBuilder(markup: List[List[InlineKeyboardButton]] /
                                                    None = None)
```

Конструктор клавіатури під повідомленням успадковує всі методи від універсального конструктора

```
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
```

Додавання нової кнопки до розмітки

```
as_markup() → aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup
```

Створення InlineKeyboardMarkup

```
__init__(markup: List[List[InlineKeyboardButton]] / None = None) → None
```

```
add(*buttons: ButtonType) → KeyboardBuilder[ButtonType]
```

Додавання однієї або кількох кнопок до розмітки.

Параметри

buttons –

Повертає

```
adjust(*sizes: int, repeat: bool = False) → KeyboardBuilder[ButtonType]
```

Налаштування раніше доданих кнопок до певних розмірів рядків.

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

Параметри

- sizes –
- repeat –

Повертає

property buttons: Generator[ButtonType, None, None]

Отримання плоского списку усіх кнопок

Повертає

copy() → *InlineKeyboardBuilder*

Робить повну копію поточного конструктора з розміткою

Повертає

export() → List[List[ButtonType]]

Експортує налаштовану розмітку як список списків кнопок

```
>>> builder = KeyboardBuilder(button_type=InlineKeyboardButton)
>>> ... # Add buttons to builder
>>> markup = InlineKeyboardMarkup(inline_keyboard=builder.export())
```

Повертає

classmethod from_markup(markup: *InlineKeyboardMarkup*) → *InlineKeyboardBuilder*

Create builder from existing markup

Параметри

markup –

Повертає

row(*buttons: ButtonType, width: int | None = None) → KeyboardBuilder[ButtonType]

Додає рядок у розмітку

Коли передано занадто багато кнопок, вони будуть розділені на багато рядків

Параметри

- buttons –
- width –

Повертає

Клавіатура відповідей

```
class aiogram.utils.keyboard.ReplyKeyboardBuilder(markup: List[List[KeyboardButton]] / None = None)
```

Конструктор клавіатури відповідей успадковує всі методи від універсального конструктора

```
button(text: str, request_contact: bool / None = None, request_location: bool / None = None,
        request_poll: KeyboardButtonPollType / None = None, **kwargs: Any) →
    aiogram.utils.keyboard.ReplyKeyboardBuilder
```

Додавання нової кнопки до розмітки

```
as_markup() → aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup
```

Створення ReplyKeyboardMarkup

```
__init__(markup: List[List[KeyboardButton]] / None = None) → None
```

```
add(*buttons: ButtonType) → KeyboardBuilder[ButtonType]
```

Додавання однієї або кількох кнопок до розмітки.

Параметри

`buttons` –

Повертає

```
adjust(*sizes: int, repeat: bool = False) → KeyboardBuilder[ButtonType]
```

Налаштування раніше доданих кнопок до певних розмірів рядків.

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

Параметри

- `sizes` –
- `repeat` –

Повертає

```
property buttons: Generator[ButtonType, None, None]
```

Отримання плоского списку усіх кнопок

Повертає

```
copy() → ReplyKeyboardBuilder
```

Робить повну копію поточного конструктора з розміткою

Повертає

```
export() → List[List[ButtonType]]
```

Експортує налаштовану розмітку як список списків кнопок

```
>>> builder = KeyboardBuilder(button_type=InlineKeyboardButton)
>>> ... # Add buttons to builder
>>> markup = InlineKeyboardMarkup(inline_keyboard=builder.export())
```

Повертає

```
classmethod from_markup(markup: ReplyKeyboardMarkup) → ReplyKeyboardBuilder
```

Create builder from existing markup

Параметри

markup –

Повертає

```
row(*buttons: ButtonType, width: int | None = None) → KeyboardBuilder[ButtonType]
```

Додає рядок у розмітку

Коли передано занадто багато кнопок, вони будуть розділені на багато рядків

Параметри

- buttons –

- width –

Повертає

2.5.2 Переклад

Для того, щоб Ваш бот володів декількома мовами, необхідно додати мінімальну кількість хуків до вашого Python коду

Ці хуки звуться рядками передкладу

Утиліти перекладу побудовані на основі модулю GNU gettext Python і Babel library.

Встановлення

Babel потрібний для забезпечення простоти експорту рядків перекладу з Вашого коду.

Можна встановити безпосередньо з pip:

```
pip install Babel
```

чи як додаткову залежність *aiogram*:

```
pip install aiogram[i18n]
```

Як зробити повідомлення перекладаваними?

Для того, щоб gettext знав які рядки слід перекласти, Вам необхідно відмітити рядки перекладу.

Наприклад:

```
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)
        )
    )
```


Небезпека: f-рядки не можна використовувати як рядки перекладу, оскільки будь-які динамічні змінні слід додати до повідомлення після отримання перекладеного повідомлення

Крім того, якщо Ви бажаєте використати перекладений рядок у фільтрах, Вам треба використати відкладений переклад (lazy gettext):

```
from aiogram import F
from aiogram.utils.i18n import lazy_gettext as __

@router.message(F.text == __("My menu entry"))
...
```

Небезпека: Відкладені виклики gettext слід завжди використовувати, коли поточна мова на даний момент невідома.

Небезпека: Відкладені виклики gettext не можна використовувати як значення для методів API або будь-якого об'єкта Telegram (наприклад, `aiogram.types.inline_keyboard_button.InlineKeyboardButton` тощо)

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.

Налаштування рушія

Коли ваші повідомлення вже готові використовувати `gettext`, Ваш бот повинен знати, як визначити мову користувача.

Поруч з місцем ініціалізації диспетчера має бути створений екземпляр перекладача: `class: aiogram.utils.i18n.I18n`.

```
i18n = I18n(path="locales", default_locale="en", domain="messages")
```

Після цього Вам потрібно буде вибрати одну з вбудованих проміжних програм (middleware) `I18n` або написати власну.

Вбудовані проміжні програми:

SimpleI18nMiddleware

```
class aiogram.utils.i18n.middleware.SimpleI18nMiddleware(i18n: I18n, i18n_key: str | None =
                                                         'i18n', middleware_key: str =
                                                         'i18n_middleware')
```

Проста I18n проміжна програма.

Вибирає код мови з об'єкта User, отриманого в події.

```
__init__(i18n: I18n, i18n_key: str | None = 'i18n', middleware_key: str = 'i18n_middleware') →
None
```

Створення екземпляру проміжної програми.

Параметри

- `i18n` – екземпляр I18n
- `i18n_key` – ключ (назва ключа) екземпляру I18n в контексті
- `middleware_key` – контекстний ключ для цієї проміжної програми

ConstI18nMiddleware

```
class aiogram.utils.i18n.middleware.ConstI18nMiddleware(locale: str, i18n: I18n, i18n_key: str |
                                                         None = 'i18n', middleware_key: str =
                                                         'i18n_middleware')
```

Проміжна програма Const вибирає статично визначену локаль.

```
__init__(locale: str, i18n: I18n, i18n_key: str | None = 'i18n', middleware_key: str =
         'i18n_middleware') → None
```

Створення екземпляру проміжної програми.

Параметри

- `i18n` – екземпляр I18n
- `i18n_key` – ключ (назва ключа) екземпляру I18n в контексті
- `middleware_key` – контекстний ключ для цієї проміжної програми

FSMI18nMiddleware

```
class aiogram.utils.i18n.middleware.FSMI18nMiddleware(i18n: I18n, key: str = 'locale', i18n_key:
                                                         str | None = 'i18n', middleware_key: str =
                                                         'i18n_middleware')
```

Ця проміжна програма зберігає локаль у сховищі FSM.

```
__init__(i18n: I18n, key: str = 'locale', i18n_key: str | None = 'i18n', middleware_key: str =
         'i18n_middleware') → None
```

Створення екземпляру проміжної програми.

Параметри

- `i18n` – екземпляр I18n
- `i18n_key` – ключ (назва ключа) екземпляру I18n в контексті
- `middleware_key` – контекстний ключ для цієї проміжної програми

```
async set_locale(state: FSMContext, locale: str) → None
```

Запис нової локалі у сховище

Параметри

- `state` – екземпляр `FSMContext`
- `locale` – нова локаль

I18nMiddleware

або визначте вашу власну проміжну програму, основану на абстракції `I18nMiddleware`:

```
class aiogram.utils.i18n.middleware.I18nMiddleware(i18n: I18n, i18n_key: str | None = 'i18n',
                                                    middleware_key: str = 'i18n_middleware')
```

Абстракція проміжної програми `I18n`

```
__init__(i18n: I18n, i18n_key: str | None = 'i18n', middleware_key: str = 'i18n_middleware') → None
```

Створення екземпляру проміжної програми.

Параметри

- `i18n` – екземпляр `I18n`
- `i18n_key` – ключ (назва ключа) екземпляру `I18n` в контексті
- `middleware_key` – контекстний ключ для цієї проміжної програми

```
abstract async get_locale(event: TelegramObject, data: Dict[str, Any]) → str
```

Визначення поточної мови користувача на основі події та контексту.

Цей метод повинен бути перевизначеним у дочірніх класах

Параметри

- `event` –
- `data` –

Повертає

```
setup(router: Router, exclude: Set[str] | None = None) → BaseMiddleware
```

Реєстрація проміжної програми для всіх подій у Роутері

Параметри

- `router` –
- `exclude` –

Повертає

Працюємо з Babel

Крок 1: Вибудування текстів

```
pybabel extract --input-dirs=. -o locales/messages.pot
```

Де `--input-dirs=.` - шлях до коду, `locales/messages.pot` — це шаблон, куди витягуватимуться повідомлення, а `messages` — домен перекладу.

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
```

Примітка: Деякі корисні опції:

- Для додавання коментарів для перекладачів, ви можете використовувати інший тег, якщо хочете (TR) `--add-comments=NOTE`
 - Contact email for bugreport `--msgid-bugs-address=EMAIL`
 - Вимкнути коментарі з розташуванням рядків у коді `--no-location`
 - Copyrights `--copyright-holder=AUTHOR`
 - Встановлення назви проекту `--project=MySuperBot`
 - Встановлення версії `--version=2.2`
-

Крок 2: Ініціалізація перекладу

```
pybabel init -i locales/messages.pot -d locales -D messages -l en
```

- `-i locales/messages.pot` - попередньо створений шаблон
- `-d locales` - тека перекладів
- `-D messages` - домен перекладів
- `-l en` - мова. Може бути змінений на будь-який інший дійсний код мови (Наприклад `-l uk` для української мови)

Крок 3: Переклад текстів

Щоб відкрити файл .po, ви можете використовувати базовий текстовий редактор або будь-який редактор .po файлів, напр. [Poedit](#)

Просто відкрийте файл із назвою `locales/{language}/LC_MESSAGES/messages.po` і впишіть переклади

Крок 4: Компіляція перекладів

```
pybabel compile -d locales -D messages
```

Крок 5: Оновлення текстів

Коли ви змінюєте код свого бота, вам потрібно оновити файли .po і .mo

- Крок 5.1: відновлення файлу .pot: команда з кроку 1
- Крок 5.2: оновлення .po файлів

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

- Крок 5.3: оновлення Ваших перекладів: розміщення та інструменти Ви знаєте з кроку 3.
- Крок 5.4: компіляція .mo файлів : команда з кроку 4

2.5.3 Відправник дій у чаті

Відправник

```
class aiogram.utils.chat_action.ChatActionSender(*, bot: Bot, chat_id: str | int,
                                                message_thread_id: int | None = None, action:
                                                str = 'typing', interval: float = 5.0,
                                                initial_sleep: float = 0.0)
```

Ця утиліта допомагає автоматично надсилати дії чату, допоки виконуються тривалі дії ботом, щоб повідомити користувачів бота про те що бот щось робить і не завершив роботу аварійно.

Надає простий для використання контекстний менеджер.

Технічно, відправник запускає фонову задачу з нескінченним циклом, який працює до завершення дії та надсилає дію чату кожні 5 секунд.

```
__init__(*, bot: Bot, chat_id: str | int, message_thread_id: int | None = None, action: str =
        'typing', interval: float = 5.0, initial_sleep: float = 0.0) → None
```

Параметри

- `bot` – екземпляр бота, необов'язковий параметр
- `chat_id` – ідентифікатор цільового чату
- `message_thread_id` – unique identifier for the target message thread; supergroups only
- `action` – тип дії
- `interval` – інтервал між ітераціями

- `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) → ChatActionSender
```

Створення екземпляру відправника з дією `choose_sticker`

```
classmethod find_location(chat_id: int | str, bot: Bot, message_thread_id: int | None = None,
                           interval: float = 5.0, initial_sleep: float = 0.0) → ChatActionSender
```

Створення екземпляру відправника з дією `find_location`

```
classmethod record_video(chat_id: int | str, bot: Bot, message_thread_id: int | None = None,
                          interval: float = 5.0, initial_sleep: float = 0.0) → ChatActionSender
```

Створення екземпляру відправника з дією `record_video`

```
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) →
                               ChatActionSender
```

Створення екземпляру відправника з дією `record_video_note`

```
classmethod record_voice(chat_id: int | str, bot: Bot, message_thread_id: int | None = None,
                          interval: float = 5.0, initial_sleep: float = 0.0) → ChatActionSender
```

Створення екземпляру відправника з дією `record_voice`

```
classmethod typing(chat_id: int | str, bot: Bot, message_thread_id: int | None = None, interval:
                    float = 5.0, initial_sleep: float = 0.0) → ChatActionSender
```

Створення екземпляру відправника з дією `typing`

```
classmethod upload_document(chat_id: int | str, bot: Bot, message_thread_id: int | None =
                             None, interval: float = 5.0, initial_sleep: float = 0.0) →
                             ChatActionSender
```

Створення екземпляру відправника з дією `upload_document`

```
classmethod upload_photo(chat_id: int | str, bot: Bot, message_thread_id: int | None = None,
                          interval: float = 5.0, initial_sleep: float = 0.0) → ChatActionSender
```

Створення екземпляру відправника з дією `upload_photo`

```
classmethod upload_video(chat_id: int | str, bot: Bot, message_thread_id: int | None = None,
                          interval: float = 5.0, initial_sleep: float = 0.0) → ChatActionSender
```

Створення екземпляру відправника з дією `upload_video`

```
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) →
                               ChatActionSender
```

Створення екземпляру відправника з дією `upload_video_note`

```
classmethod upload_voice(chat_id: int | str, bot: Bot, message_thread_id: int | None = None,
                          interval: float = 5.0, initial_sleep: float = 0.0) → ChatActionSender
```

Створення екземпляру відправника з дією `upload_voice`

Використання

```
async with ChatActionSender.typing(bot=bot, chat_id=message.chat.id):
    # Do something...
    # Perform some long calculations
    await message.answer(result)
```

Проміжні програми

```
class aiogram.utils.chat_action.ChatActionMiddleware
```

Допомагає автоматично використовувати відправника дій чату для всіх обробників повідомлень

Використання

Перед використанням слід зареєструвати для події *message*

```
<router or dispatcher>.message.middleware(ChatActionMiddleware())
```

Після цього всі обробники, що працюють довше за *initial_sleep*, виконуватимуть дію „*typing*“ чату

Також відправник може бути налаштованим за допомогою функції міток для певного обробника.

Зміна лише типу дії:

```
@router.message(...)
@flags.chat_action("sticker")
async def my_handler(message: Message): ...
```

Зміна конфігурації відправника:

```
@router.message(...)
@flags.chat_action(initial_sleep=2, action="upload_document", interval=3)
async def my_handler(message: Message): ...
```

2.5.4 Веб Застосунок (WebApp)

Telegram Bot API 6.0 зробив революцію у розробці чат-ботів, використовуючи особливості Веб Застосунків.

Ви можете прочитати більше про це в офіційному [блогі](#) та [документації](#).

aiogram реалізує прості утиліти для усунення головного болю, надаючи готові інструменти перевірки даних із Веб Застосунку Telegram на серверній стороні.

Використання

Наприклад, із фронтенду ви передасте `application/x-www-form-urlencoded` в POST запиті із полем `_auth` у тілі та хочете повернути інформацію про користувача у відповідь як `application/json`

```
from aiogram.utils.web_app import safe_parse_webapp_init_data
from aiohttp.web_request import Request
from aiohttp.web_response import json_response

async def check_data_handler(request: Request):
    bot: Bot = request.app["bot"]

    data = await request.post() # application/x-www-form-urlencoded
    try:
        data = safe_parse_webapp_init_data(token=bot.token, init_data=data["_auth"])
    except ValueError:
        return json_response({"ok": False, "err": "Unauthorized"}, status=401)
    return json_response({"ok": True, "data": data.user.dict()})
```

Функції

`aiogram.utils.web_app.check_webapp_signature(token: str, init_data: str) → bool`

Перевірка вхідного підпису даних ініціалізації Веб Застосунку

Джерело: <https://core.telegram.org/bots/webapps#validating-data-received-via-the-web-app>

Параметри

- `token` – Токен бота
- `init_data` – дані з фронтенду, що підлягають перевірці

Повертає

`aiogram.utils.web_app.parse_webapp_init_data(init_data: str, *, loads: ~typing.Callable[[...], ~typing.Any] = <function loads>) → WebAppInitData`

Аналіз даних ініціалізації Веб Застосунку і повернення їх як об'єкту `WebAppInitData`

Цей метод не забезпечує безпеку, тому вам не варто довіряти цим даним, замість цього використовуйте `safe_parse_webapp_init_data`.

Параметри

- `init_data` – дані з frontend для аналізу
- `loads` –

Повертає

`aiogram.utils.web_app.safe_parse_webapp_init_data(token: str, init_data: str, *, loads: ~typing.Callable[[...], ~typing.Any] = <function loads>) → WebAppInitData`

Перевірка необроблених даних ініціалізації Веб Застосунку і повернення їх як об'єкту `WebAppInitData`

Видає `ValueError`, коли дані недійсні

Параметри

- `token` – токен бота
- `init_data` – дані з фронтенду для аналізу і перевірки
- `loads` –

Повертає

Типи

```
class aiogram.utils.web_app.WebAppInitData(**extra_data: Any)
```

Об'єкт, що містить дані які передаються у Веб Застосунок під час його відкриття. Він порожній, якщо Веб Застосунок було запущено за допомогою кнопки клавіатури.

Джерело: <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}
```

Конфігурація для моделі має бути словником, що відповідає *[ConfigDict][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)}
```

Метадані про поля, визначені на моделі, відображення назв полів у *[FieldInfo][pydantic.fields.FieldInfo]*.

Це замінює *Model.__fields__* з 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.

```
query_id: str | None
```

Унікальний ідентифікатор сеансу Веб Застосунку, необхідний для надсилання повідомлень через метод `answerWebAppQuery`.

```
user: WebAppUser | None
```

Об'єкт, що містить дані про поточного користувача.

```
receiver: WebAppUser | None
```

Об'єкт, що містить дані про співрозмовника поточного користувача в чаті, де бот був запущений через меню вкладення. Повертається тільки для веб-додатків, запущених через меню вкладень.

`chat: WebAppChat | None`

Об'єкт, що містить дані про чат, в якому бот був запущений через меню вкладень. Повертається для супергруп, каналів і групових чатів - тільки для веб-додатків, запущених через меню вкладень.

`chat_type: str | None`

Тип чату, з якого було відкрито веб-додаток. Може бути як «sender» для приватного чату з користувачем, який відкрив посилання, так і «private», «group», «supergroup» або «channel». Повертається тільки для веб-програм, запущених за прямим посиланням.

`chat_instance: str | None`

Глобальний ідентифікатор, що унікальний для чату, з якого було відкрито веб-програму. Повертається тільки для веб-додатків, запущених за прямим посиланням.

`start_param: str | None`

Значення параметра startattach, передане через посилання. Повертається лише для Веб Застосунків, коли їх запускають із меню вкладень за посиланням. Значення параметра start_param також буде передано в GET-параметр tgWebAppStartParam, тому Веб Застосунок може відразу завантажити правильний інтерфейс.

`can_send_after: int | None`

Час в секундах після якого повідомлення може бути відправлене за допомогою метода answerWebAppQuery.

`auth_date: datetime`

Unix час відкриття форми.

`hash: str`

Хеш усіх переданих параметрів, за допомогою якого бот-сервер може перевірити їх дійсність.

`class aiogram.utils.web_app.WebAppUser(**extra_data: Any)`

Об'єкт що містить дані користувача Веб Застосунку.

Джерело: <https://core.telegram.org/bots/webapps#webappuser>

`id: int`

Унікальний ідентифікатор користувача або бота. Це число може мати більше 32 значущих бітів, і деякі мови програмування можуть мати труднощі в його інтерпретації. Він має щонайбільше 52 значущі біти, тому 64-бітне ціле число або тип з плаваючою точністю подвійної точності є безпечним для зберігання цього ідентифікатора.

`is_bot: bool | None`

True, якщо цей користувач бот. Повертається лише в полі отримувача(receiver).

`first_name: str`

Ім'я користувача або бота.

`last_name: str | None`

Прізвище користувача або бота.

`username: str | None`

Нік користувача або бота.

`language_code: str | None`

Мовний тег IETF мови користувача. Повертається лише в полі користувача(user).

`is_premium: bool | None`

True, якщо цей користувач має підписку Telegram Premium.

`added_to_attachment_menu: bool | None`

True, якщо цей користувач додав бота до меню вкладень.

`allows_write_to_pm: bool | None`

True, якщо цей користувач дозволив надсилати йому повідомлення.

`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}`

Конфігурація для моделі має бути словником, що відповідає [*ConfigDict*][*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)}`

Метадані про поля, визначені на моделі, відображення назв полів у [*FieldInfo*][*pydantic.fields.FieldInfo*].

Це замінює *Model.__fields__* з 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-адреса фотографії профілю користувача. Фотографія може бути у форматах .jpeg або .svg. Повертається лише для Веб Застосунків, запущених із меню вкладень.

`class aiogram.utils.web_app.WebAppChat(**extra_data: Any)`

Об'єкт чату.

Джерело: <https://core.telegram.org/bots/webapps#webappchat>

`id: int`

Унікальний ідентифікатор цього чату. Це число може мати більше 32 значущих бітів, і деякі мови програмування можуть мати труднощі в його інтерпретації. Він має щонайбільше 52 значущі біти, тому 64-бітне ціле число або тип з плаваючою точкою подвійної точності є безпечним для зберігання цього ідентифікатора.

`type: str`

Тип чату, може бути «group», «supergroup» або «channel»

`title: str`

Назва чату

`username: str | None`

Нік користувача або бота

`photo_url: str | None`

URL-адреса фотографії чату. Фотографія може бути у форматах .jpeg або .svg. Повертається лише для Веб Застосунків, запущених із меню вкладень.

`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}`

Конфігурація для моделі має бути словником, що відповідає [*ConfigDict*][*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)}`

Метадані про поля, визначені на моделі, відображення назв полів у [*FieldInfo*][*pydantic.fields.FieldInfo*].

Це замінює *Model.__fields__* з 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.

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.

Небезпека: 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):
```

(continues on next page)

(continued from previous page)

```
...
if <everything is ok>:
    callback_answer.text = "All is ok"
```

Description of objects

```
class aiogram.utils.callback_answer.CallbackAnswerMiddleware(pre: bool = False, text: str / None
                                                            = None, show_alert: bool / None
                                                            = None, url: str / None = None,
                                                            cache_time: int / None = None)
```

Bases: *BaseMiddleware*

```
__init__(pre: bool = False, text: str / None = None, show_alert: bool / None = None, url: str /
         None = None, cache_time: int / None = None) → None
```

Inner middleware for callback query handlers, can be useful in bots with a lot of callback handlers to automatically take answer to all requests

Параметри

- **pre** – send answer before execute handler
- **text** – answer with text
- **show_alert** – show alert
- **url** – game url
- **cache_time** – cache answer for some time

```
class aiogram.utils.callback_answer.CallbackAnswer(answered: bool, disabled: bool = False, text:
                                                    str / None = None, show_alert: bool / None
                                                    = None, url: str / None = None, cache_time:
                                                    int / None = None)
```

Bases: *object*

```
__init__(answered: bool, disabled: bool = False, text: str / None = None, show_alert: bool / None =
         None, url: str / None = None, cache_time: int / None = None) → None
```

Callback answer configuration

Параметри

- **answered** – this request is already answered by middleware
- **disabled** – answer will not be performed
- **text** – answer with text
- **show_alert** – show alert
- **url** – game url
- **cache_time** – cache answer for some time

```
disable() → None
```

Deactivate answering for this handler

```
property disabled: bool
```

Indicates that automatic answer is disabled in this handler

```

property answered: bool
    Indicates that request is already answered by middleware
property text: str | None
    Response text :return:
property show_alert: bool | None
    Whether to display an alert
property url: str | None
    Game url
property cache_time: int | None
    Response cache time

```

2.5.6 Formatting

Make your message formatting flexible and simple

This instrument works on top of Message entities instead of using HTML or Markdown markups, you can easily construct your message and sent it to the Telegram without the need to remember tag parity (opening and closing) or escaping user input.

Usage

Basic scenario

Construct your message and send it to the Telegram.

```

content = Text("Hello, ", Bold(message.from_user.full_name), "!")
await message.answer(**content.as_kwargs())

```

Is the same as the next example, but without usage markup

```

await message.answer(
    text=f"Hello, <b>{html.quote(message.from_user.full_name)}!",
    parse_mode=ParseMode.HTML
)

```

Literally when you execute `as_kwargs` method the Text object is converted into text `Hello, Alex!` with entities list `[MessageEntity(type='bold', offset=7, length=4)]` and passed into dict which can be used as `**kwargs` in API call.

The complete list of elements is listed *on this page below*.

Advanced scenario

On top of base elements can be implemented content rendering structures, so, out of the box aiogram has a few already implemented functions that helps you to format your messages:

`aiogram.utils.formatting.as_line(*items: Any, end: str = '\n', sep: str = '') → Text`

Wrap multiple nodes into line with `\n` at the end of line.

Параметри

- `items` – Text or Any
- `end` – ending of the line, by default is `\n`
- `sep` – separator between items, by default is empty string

Повертає

Text

`aiogram.utils.formatting.as_list(*items: Any, sep: str = '\n') → Text`

Wrap each element to separated lines

Параметри

- `items` –
- `sep` –

Повертає

`aiogram.utils.formatting.as_marked_list(*items: Any, marker: str = '- ') → Text`

Wrap elements as marked list

Параметри

- `items` –
- `marker` – line marker, by default is „- „

Повертає

Text

`aiogram.utils.formatting.as_numbered_list(*items: Any, start: int = 1, fmt: str = '{}. ') → Text`

Wrap elements as numbered list

Параметри

- `items` –
- `start` – initial number, by default 1
- `fmt` – number format, by default „{}. „

Повертає

Text

`aiogram.utils.formatting.as_section(title: Any, *body: Any) → Text`

Wrap elements as simple section, section has title and body

Параметри

- `title` –
- `body` –

Повертає

Text

`aiogram.utils.formatting.as_marked_section(title: Any, *body: Any, marker: str = '- ') → Text`

Wrap elements as section with marked list

Параметри

- title –
- body –
- marker –

Повертає

`aiogram.utils.formatting.as_numbered_section(title: Any, *body: Any, start: int = 1, fmt: str = '{}. ') → Text`

Wrap elements as section with numbered list

Параметри

- title –
- body –
- start –
- fmt –

Повертає

`aiogram.utils.formatting.as_key_value(key: Any, value: Any) → Text`

Wrap elements pair as key-value line. ({key} {value})

Параметри

- key –
- value –

Повертає

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),
    ),
)
```

(continues on next page)

(continued from previous page)

```
        as_key_value("Success", 3),
        as_key_value("Failed", 1),
        marker=" ",
    ),
    HashTag("#test"),
    sep="\n\n",
)
```

Will be rendered into:

Success:

Test 1

Test 3

Test 4

Failed:

Test 2

Summary:

Total: 4

Success: 3

Failed: 1

#test

Or as HTML:

```
<b>Success:</b>
Test 1
Test 3
Test 4

<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
```

Повертає

```
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())
```

Параметри

- text_key –
- entities_key –
- replace_parse_mode –
- parse_mode_key –

Повертає

```
as_html() → str
    Render elements tree as HTML markup
as_markdown() → str
    Render elements tree as MarkdownV2 markup
```

Available elements

```
class aiogram.utils.formatting.Text(*body: Any, **params: Any)
    Bases: Iterable[Any]
    Simple text element

class aiogram.utils.formatting.HashTag(*body: Any, **params: Any)
    Bases: Text
    Hashtag element.
```

Попередження: 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.

Попередження: 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.

Попередження: 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*.

Попередження: *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

```
class aiogram.utils.media_group.MediaGroupBuilder(media: List[InputMediaAudio /
                                                    InputMediaPhoto / InputMediaVideo /
                                                    InputMediaDocument] / None = None,
                                                  caption: str / None = None, caption_entities:
                                                  List[MessageEntity] / None = None)

    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.

Параметри

kwargs – Keyword arguments for the media object. The available keyword arguments depend on the media type.

Повертає

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.

Параметри

- **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

Повертає

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) →
    None
```

Add a document to the media group.

Параметри

- **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.

Повертає

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) → None
```

Add a photo to the media group.

Параметри

- **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

Повертає

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.

Параметри

- **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

Повертає

None

```
build() → 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.

Повертає

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) → str`

Create „start“ deep link with your payload.

If you need to encode payload or pass special characters -
set encode as True

Параметри

- `bot` – bot instance
- `payload` – args passed with `/start`
- `encode` – encode payload with `base64url` or custom encoder
- `encoder` – custom encoder callable

Повертає

link

```
aiogram.utils.deep_linking.decode_payload(payload: str, decoder: Callable[[bytes], bytes] / None =
                                         None) → str
```

Decode URL-safe `base64url` payload with decoder.

2.6 Changelog

2.6.1 3.7.0 [UNRELEASED DRAFT] (2024-05-10)

Features

- Added new storage `aiogram.fsm.storage.MongoStorage` for Finite State Machine based on Mongo DB (using `motor` library) [#1434](#)

2.6.2 3.6.0 (2024-05-06)

Features

- Added full support of Bot API 7.3 [#1480](#)

Improved Documentation

- Added telegram objects transformation block in 2.x -> 3.x migration guide [#1412](#)

2.6.3 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

2.6.4 3.4.1 (2024-02-17)

Bugfixes

- Fixed JSON serialization of the *LinkPreviewOptions* class while it is passed as bot-wide default options. #1418

2.6.5 3.4.0 (2024-02-16)

Features

- Reworked bot-wide globals like *parse_mode*, *disable_web_page_preview*, and others to be more flexible.

Попередження: 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 `KeyboardButtonPollType` for `KeyboardButtonPollType.type` field has been 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](#)

2.6.6 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](#)

2.6.7 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

2.6.8 3.1.1 (2023-09-25)

Bugfixes

- Fixed *pydantic* version <2.4, since 2.4 has breaking changes. [#1322](#)

2.6.9 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](#)

2.6.10 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](#)

2.6.11 3.0.0rc2 (2023-08-18)

Bugfixes

- Fixed missing message content types (`ContentType.USER_SHARED`, `ContentType.CHAT_SHARED`) [#1252](#)
- Fixed nested hashtag, cashtag and email message entities not being parsed correctly when these entities are inside another entity. [#1259](#)
- Moved global filters check placement into router to add chance to pass context from global filters into handlers in the same way as it possible in other places [#1266](#)

Improved Documentation

- Added error handling example *examples/error_handling.py* #1099
- Added a few words about skipping pending updates #1251
- Added a section on Dependency Injection technology #1253
- This update includes the addition of a multi-file bot example to the repository. #1254
- Refactored examples code to use aiogram enumerations and enhanced chat messages with markdown beautification's for a more user-friendly display. #1256
- Supplemented «Finite State Machine» section in Migration FAQ #1264
- Removed extra param in docstring of TelegramEventObserver's filter method and fixed typo in I18n documentation. #1268

Misc

- Enhanced the warning message in dispatcher to include a JSON dump of the update when update type is not known. #1269
- Added support for Bot API 6.8 #1275

2.6.12 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](#)

2.6.13 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](#)

2.6.14 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
)
```

Примітка: 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

Небезпека: 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

Попередження: 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

Попередження: 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.

Небезпека: 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](#)

2.6.15 3.0.0b7 (2023-02-18)

Попередження: Note that this version has incompatibility with Python 3.8-3.9 in case when you create an instance of Dispatcher outside of the any coroutine.

Sorry for the inconvenience, it will be fixed in the next version.

This code will not work:

```
dp = Dispatcher()

def main():
    ...
    dp.run_polling(...)

main()
```

But if you change it like this it should works as well:

```
router = Router()

async def main():
    dp = Dispatcher()
    dp.include_router(router)
    ...
    dp.start_polling(...)

asyncio.run(main())
```

Features

- Added missing shortcuts, new enums, reworked old stuff

Breaking All previously added enums is re-generated in new place - *aiogram.enums* instead of *aiogram.types*

Added enums: *aiogram.enums.bot_command_scope_type.BotCommandScopeType*,
aiogram.enums.chat_action.ChatAction, *aiogram.enums.chat_member_status.ChatMemberStatus*,
aiogram.enums.chat_type.ChatType, *aiogram.enums.content_type.ContentType*,
aiogram.enums.dice_emoji.DiceEmoji, *aiogram.enums.inline_query_result_type.InlineQueryResultType*,
aiogram.enums.input_media_type.InputMediaType, *aiogram.enums.mask_position_point.MaskPositionPoint*,
aiogram.enums.menu_button_type.MenuButtonType, *aiogram.enums.message_entity_type.MessageEntityType*,
aiogram.enums.parse_mode.ParseMode, *aiogram.enums.poll_type.PollType*,
aiogram.enums.sticker_type.StickerType, *aiogram.enums.topic_icon_color.TopicIconColor*,
aiogram.enums.update_type.UpdateType,

Added shortcuts:

- **Chat** *aiogram.types.chat.Chat.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](#)

Небезпека: 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](#)

2.6.16 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 ca-certificates the requests to Bot API fails with reason `[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: self signed certificate in certificate chain` [#1066](#)

2.6.17 3.0.0b5 (2022-10-02)

Features

- Add PyPy support and run tests under PyPy [#985](#)
- Added message text to aiogram exceptions representation [#988](#)
- Added warning about using magic filter from `magic_filter` instead of `aiogram`'s ones. Is recommended to use `from aiogram import F` instead of `from magic_filter import F` [#990](#)
- Added more detailed error when server response can't be deserialized. This feature will help to debug unexpected responses from the Server [#1014](#)

Bugfixes

- Reworked error event, introduced `aiogram.types.error_event.ErrorEvent` object. [#898](#)
- Fixed escaping markdown in `aiogram.utils.markdown` module [#903](#)
- Fixed polling crash when Telegram Bot API raises HTTP 429 status-code. [#995](#)
- Fixed empty mention in command parsing, now it will be `None` instead of an empty string [#1013](#)

Improved Documentation

- Initialized Docs translation (added Ukrainian language) [#925](#)

Deprecations and Removals

- Removed filters factory as described in corresponding issue. [#942](#)

Misc

- Now Router/Dispatcher accepts only keyword arguments. [#982](#)

2.6.18 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](#)

2.6.19 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](#)

2.6.20 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](#)

2.6.21 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]):
```

(continues on next page)

(continued from previous page)

```

    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

2.6.22 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

2.6.23 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

2.6.24 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](#)

2.6.25 3.0.0a15 (2021-09-10)

Features

- Ability to iterate over all states in StatesGroup. Aiogram already had in check for states group so this is relative feature. [#666](#)

Bugfixes

- Fixed incorrect type checking in the `aiogram.utils.keyboard.KeyboardBuilder` [#674](#)

Misc

- Disable ContentType filter by default [#668](#)
- Moved update type detection from Dispatcher to Update object [#669](#)
- Updated **pre-commit** config [#681](#)
- Reworked `handlers_in_use` util. Function moved to Router as method `.resolve_used_update_types()` [#682](#)

2.6.26 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

2.6.27 2.14.3 (2021-07-21)

- Fixed ChatMember type detection via adding customizable object serialization mechanism (#624, #623)

2.6.28 2.14.2 (2021-07-26)

- Fixed MemoryStorage cleaner (#619)
- Fixed unused default locale in I18nMiddleware (#562, #563)

2.6.29 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)

2.6.30 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)

2.6.31 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)

2.6.32 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)

2.6.33 2.11.2 (2021-11-10)

- Fixed default parse mode
- Added missing «supports_streaming» argument to answer_video method [#462](#)

2.6.34 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](#))

2.6.35 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](#))

2.6.36 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'`

2.6.37 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)

2.6.38 2.9.2 (2021-06-13)

- Fixed Message.get_full_command() #352
- Fixed markdown util #353

2.6.39 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)

2.6.40 2.8 (2021-04-26)

- Added full support of Bot API 4.8
- Added `Message.answer_dice` and `Message.reply_dice` methods (#306)

2.6.41 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

2.6.42 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)

2.6.43 2.6 (2021-01-23)

- Full support of Telegram Bot API v4.6 (Polls 2.0) #265
- Added new filter - `IsContactSender` (commit)
- Fixed proxy extra dependencies version #262

2.6.44 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

2.6.45 2.5.2 (2021-01-01)

- Get back `quote_html` and `escape_md` functions

2.6.46 2.5.1 (2021-01-01)

- Hot-fix of `CommandStart` filter

2.6.47 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...)

2.6.48 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

2.6.49 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

2.6.50 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)

2.6.51 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

2.6.52 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)

2.6.53 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

2.6.54 1.4 (2021-08-03)

- Bot API 4.0 (#57)

2.6.55 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.

2.6.56 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

2.6.57 1.3.1 (2018-05-27)

2.6.58 1.3 (2021-04-22)

- Allow to use Socks5 proxy (need manually install `aiosocksy`).
- Refactored `aiogram.utils.executor` module.
- **[Warning]** Updated requirements list.

2.6.59 1.2.3 (2018-04-14)

- Fixed API errors detection
- Fixed compability of `setup.py` with pip 10.0.0

2.6.60 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 mmminor improvements.

2.6.61 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.

2.6.62 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.

2.6.63 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`)

2.6.64 1.0.4 (2018-01-10)

2.6.65 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.

2.6.66 1.0.2 (2017-11-29)

2.6.67 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.

2.6.68 1.0 (2017-11-19)

- Remaked data types serialoization/deserialization mechanism (Speed up).
- Fully rewrited all Telegram data types.
- Bot object was fully rewritted (regenerated).
- Full provide Telegram Bot API 3.4+ (with `sendMediaGroup`)
- Warning: Now `BaseStorage.close()` is awaitable! (FSM)
- Fixed compability with uvloop.
- More employments for `aiogram.utils.context`.
- Allowed to disable `ujson`.

- Other bug fixes and minor improvements.
- Migrated from Bitbucket to Github.

2.6.69 0.4.1 (2017-08-03)

2.6.70 0.4 (2017-08-05)

2.6.71 0.3.4 (2017-08-04)

2.6.72 0.3.3 (2017-07-05)

2.6.73 0.3.2 (2017-07-04)

2.6.74 0.3.1 (2017-07-04)

2.6.75 0.2b1 (2017-06-00)

2.6.76 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,mongo,proxy,i18n]"
```

Windows:

```
pip install -e .[dev,test,docs,fast,redis,mongo,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 or/and MongoDB-storage, you will need to test everything works with Redis or/and MongoDB:

```
pytest --redis redis://<host>:<port>/<db> --mongo mongodb://<user>:<password>@<host>:  
↪<port> 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 utilities 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](#).

a

- aiogram.dispatcher.flags, 576
- aiogram.enums.bot_command_scope_type, 489
- aiogram.enums.chat_action, 489
- aiogram.enums.chat_boost_source_type, 490
- aiogram.enums.chat_member_status, 490
- aiogram.enums.chat_type, 491
- aiogram.enums.content_type, 491
- aiogram.enums.currency, 493
- aiogram.enums.dice_emoji, 496
- aiogram.enums.encrypted_passport_element, 496
- aiogram.enums.inline_query_result_type, 497
- aiogram.enums.input_media_type, 498
- aiogram.enums.keyboard_button_poll_type_type, 498
- aiogram.enums.mask_position_point, 498
- aiogram.enums.menu_button_type, 499
- aiogram.enums.message_entity_type, 499
- aiogram.enums.message_origin_type, 500
- aiogram.enums.parse_mode, 500
- aiogram.enums.passport_element_error_type, 500
- aiogram.enums.poll_type, 501
- aiogram.enums.reaction_type_type, 501
- aiogram.enums.sticker_format, 501
- aiogram.enums.sticker_type, 502
- aiogram.enums.topic_icon_color, 502
- aiogram.enums.update_type, 502
- aiogram.exceptions, 574
- aiogram.handlers.callback_query, 578
- aiogram.methods.add_sticker_to_set, 315
- aiogram.methods.answer_callback_query, 333
- aiogram.methods.answer_inline_query, 465
- aiogram.methods.answer_pre_checkout_query, 474
- aiogram.methods.answer_shipping_query, 475
- aiogram.methods.answer_web_app_query, 468
- aiogram.methods.approve_chat_join_request, 335
- aiogram.methods.ban_chat_member, 336
- aiogram.methods.ban_chat_sender_chat, 337
- aiogram.methods.close, 339
- aiogram.methods.close_forum_topic, 340
- aiogram.methods.close_general_forum_topic, 341
- aiogram.methods.copy_message, 342
- aiogram.methods.copy_messages, 344
- aiogram.methods.create_chat_invite_link, 346
- aiogram.methods.create_forum_topic, 347
- aiogram.methods.create_invoice_link, 476
- aiogram.methods.create_new_sticker_set, 316
- aiogram.methods.decline_chat_join_request, 348
- aiogram.methods.delete_chat_photo, 349
- aiogram.methods.delete_chat_sticker_set, 350
- aiogram.methods.delete_forum_topic, 351
- aiogram.methods.delete_message, 451
- aiogram.methods.delete_messages, 453
- aiogram.methods.delete_my_commands, 352
- aiogram.methods.delete_sticker_from_set, 317
- aiogram.methods.delete_sticker_set, 318
- aiogram.methods.delete_webhook, 482
- aiogram.methods.edit_chat_invite_link, 354
- aiogram.methods.edit_forum_topic, 355
- aiogram.methods.edit_general_forum_topic, 356
- aiogram.methods.edit_message_caption, 454
- aiogram.methods.edit_message_live_location, 455
- aiogram.methods.edit_message_media, 458
- aiogram.methods.edit_message_reply_markup, 459
- aiogram.methods.edit_message_text, 460
- aiogram.methods.export_chat_invite_link, 357
- aiogram.methods.forward_message, 358
- aiogram.methods.forward_messages, 360
- aiogram.methods.get_business_connection, 361
- aiogram.methods.get_chat, 362

[aiogram.methods.get_chat_administrators](#), 363
[aiogram.methods.get_chat_member](#), 364
[aiogram.methods.get_chat_member_count](#), 365
[aiogram.methods.get_chat_menu_button](#), 366
[aiogram.methods.get_custom_emoji_stickers](#), 319
[aiogram.methods.get_file](#), 367
[aiogram.methods.get_forum_topic_icon_stickers](#), 368
[aiogram.methods.get_game_high_scores](#), 469
[aiogram.methods.get_me](#), 369
[aiogram.methods.get_my_commands](#), 370
[aiogram.methods.get_my_default_administrator_rights](#), 371
[aiogram.methods.get_my_description](#), 372
[aiogram.methods.get_my_name](#), 373
[aiogram.methods.get_my_short_description](#), 374
[aiogram.methods.get_sticker_set](#), 320
[aiogram.methods.get_updates](#), 483
[aiogram.methods.get_user_chat_boosts](#), 374
[aiogram.methods.get_user_profile_photos](#), 375
[aiogram.methods.get_webhook_info](#), 485
[aiogram.methods.hide_general_forum_topic](#), 376
[aiogram.methods.leave_chat](#), 377
[aiogram.methods.log_out](#), 378
[aiogram.methods.pin_chat_message](#), 379
[aiogram.methods.promote_chat_member](#), 380
[aiogram.methods.reopen_forum_topic](#), 383
[aiogram.methods.reopen_general_forum_topic](#), 384
[aiogram.methods.replace_sticker_in_set](#), 321
[aiogram.methods.restrict_chat_member](#), 385
[aiogram.methods.revoke_chat_invite_link](#), 387
[aiogram.methods.send_animation](#), 388
[aiogram.methods.send_audio](#), 391
[aiogram.methods.send_chat_action](#), 394
[aiogram.methods.send_contact](#), 395
[aiogram.methods.send_dice](#), 397
[aiogram.methods.send_document](#), 400
[aiogram.methods.send_game](#), 470
[aiogram.methods.send_invoice](#), 479
[aiogram.methods.send_location](#), 402
[aiogram.methods.send_media_group](#), 405
[aiogram.methods.send_message](#), 408
[aiogram.methods.send_photo](#), 411
[aiogram.methods.send_poll](#), 413
[aiogram.methods.send_sticker](#), 322
[aiogram.methods.send_venue](#), 417
[aiogram.methods.send_video](#), 420
[aiogram.methods.send_video_note](#), 423
[aiogram.methods.send_voice](#), 426
[aiogram.methods.set_chat_administrator_custom_title](#), 428
[aiogram.methods.set_chat_description](#), 429
[aiogram.methods.set_chat_menu_button](#), 430
[aiogram.methods.set_chat_permissions](#), 432
[aiogram.methods.set_chat_photo](#), 433
[aiogram.methods.set_chat_sticker_set](#), 434
[aiogram.methods.set_chat_title](#), 435
[aiogram.methods.set_custom_emoji_sticker_set_thumbnail](#), 324
[aiogram.methods.set_game_score](#), 472
[aiogram.methods.set_message_reaction](#), 436
[aiogram.methods.set_my_commands](#), 438
[aiogram.methods.set_my_default_administrator_rights](#), 439
[aiogram.methods.set_my_description](#), 440
[aiogram.methods.set_my_name](#), 441
[aiogram.methods.set_my_short_description](#), 442
[aiogram.methods.set_passport_data_errors](#), 487
[aiogram.methods.set_sticker_emoji_list](#), 326
[aiogram.methods.set_sticker_keywords](#), 327
[aiogram.methods.set_sticker_mask_position](#), 328
[aiogram.methods.set_sticker_position_in_set](#), 329
[aiogram.methods.set_sticker_set_thumbnail](#), 330
[aiogram.methods.set_sticker_set_title](#), 331
[aiogram.methods.set_webhook](#), 485
[aiogram.methods.stop_message_live_location](#), 462
[aiogram.methods.stop_poll](#), 464
[aiogram.methods.unban_chat_member](#), 443
[aiogram.methods.unban_chat_sender_chat](#), 445
[aiogram.methods.unhide_general_forum_topic](#), 446
[aiogram.methods.unpin_all_chat_messages](#), 447
[aiogram.methods.unpin_all_forum_topic_messages](#), 448
[aiogram.methods.unpin_all_general_forum_topic_messages](#), 449
[aiogram.methods.unpin_chat_message](#), 450
[aiogram.methods.upload_sticker_file](#), 332
[aiogram.types.animation](#), 19
[aiogram.types.audio](#), 20
[aiogram.types.background_fill](#), 21
[aiogram.types.background_fill_freeform_gradient](#), 21
[aiogram.types.background_fill_gradient](#), 22
[aiogram.types.background_fill_solid](#), 22
[aiogram.types.background_type](#), 23
[aiogram.types.background_type_chat_theme](#), 23

[aiogram.types.background_type_fill, 23](#)
[aiogram.types.background_type_pattern, 24](#)
[aiogram.types.background_type_wallpaper, 25](#)
[aiogram.types.birthdate, 25](#)
[aiogram.types.bot_command, 26](#)
[aiogram.types.bot_command_scope, 26](#)
[aiogram.types.bot_command_scope_all_chat_administrators, 27](#)
[aiogram.types.bot_command_scope_all_group_chats, 27](#)
[aiogram.types.bot_command_scope_all_private_chats, 28](#)
[aiogram.types.bot_command_scope_chat, 28](#)
[aiogram.types.bot_command_scope_chat_administrators, 29](#)
[aiogram.types.bot_command_scope_chat_member, 29](#)
[aiogram.types.bot_command_scope_default, 30](#)
[aiogram.types.bot_description, 30](#)
[aiogram.types.bot_name, 31](#)
[aiogram.types.bot_short_description, 31](#)
[aiogram.types.business_connection, 31](#)
[aiogram.types.business_intro, 32](#)
[aiogram.types.business_location, 32](#)
[aiogram.types.business_messages_deleted, 33](#)
[aiogram.types.business_opening_hours, 33](#)
[aiogram.types.business_opening_hours_interval, 34](#)
[aiogram.types.callback_game, 313](#)
[aiogram.types.callback_query, 34](#)
[aiogram.types.chat, 36](#)
[aiogram.types.chat_administrator_rights, 52](#)
[aiogram.types.chat_background, 54](#)
[aiogram.types.chat_boost, 54](#)
[aiogram.types.chat_boost_added, 55](#)
[aiogram.types.chat_boost_removed, 55](#)
[aiogram.types.chat_boost_source, 56](#)
[aiogram.types.chat_boost_source_gift_code, 56](#)
[aiogram.types.chat_boost_source_giveaway, 56](#)
[aiogram.types.chat_boost_source_premium, 57](#)
[aiogram.types.chat_boost_updated, 58](#)
[aiogram.types.chat_full_info, 58](#)
[aiogram.types.chat_invite_link, 62](#)
[aiogram.types.chat_join_request, 63](#)
[aiogram.types.chat_location, 103](#)
[aiogram.types.chat_member, 104](#)
[aiogram.types.chat_member_administrator, 105](#)
[aiogram.types.chat_member_banned, 107](#)
[aiogram.types.chat_member_left, 107](#)
[aiogram.types.chat_member_member, 108](#)
[aiogram.types.chat_member_owner, 108](#)
[aiogram.types.chat_member_restricted, 109](#)
[aiogram.types.chat_member_updated, 110](#)
[aiogram.types.chat_permissions, 130](#)
[aiogram.types.chat_photo, 132](#)
[aiogram.types.chat_shared, 132](#)
[aiogram.types.chosen_inline_result, 240](#)
[aiogram.types.contact, 133](#)
[aiogram.types.dice, 134](#)
[aiogram.types.document, 134](#)
[aiogram.types.encrypted_credentials, 293](#)
[aiogram.types.encrypted_passport_element, 294](#)
[aiogram.types.error_event, 573](#)
[aiogram.types.external_reply_info, 135](#)
[aiogram.types.file, 137](#)
[aiogram.types.force_reply, 138](#)
[aiogram.types.forum_topic, 138](#)
[aiogram.types.forum_topic_closed, 139](#)
[aiogram.types.forum_topic_created, 139](#)
[aiogram.types.forum_topic_edited, 140](#)
[aiogram.types.forum_topic_reopened, 140](#)
[aiogram.types.game, 313](#)
[aiogram.types.game_high_score, 314](#)
[aiogram.types.general_forum_topic_hidden, 140](#)
[aiogram.types.general_forum_topic_unhidden, 141](#)
[aiogram.types.giveaway, 141](#)
[aiogram.types.giveaway_completed, 142](#)
[aiogram.types.giveaway_created, 142](#)
[aiogram.types.giveaway_winners, 142](#)
[aiogram.types.inaccessible_message, 143](#)
[aiogram.types.inline_keyboard_button, 144](#)
[aiogram.types.inline_keyboard_markup, 145](#)
[aiogram.types.inline_query, 240](#)
[aiogram.types.inline_query_result, 242](#)
[aiogram.types.inline_query_result_article, 243](#)
[aiogram.types.inline_query_result_audio, 244](#)
[aiogram.types.inline_query_result_cached_audio, 246](#)
[aiogram.types.inline_query_result_cached_document, 248](#)
[aiogram.types.inline_query_result_cached_gif, 250](#)
[aiogram.types.inline_query_result_cached_mpeg4_gif, 252](#)
[aiogram.types.inline_query_result_cached_photo, 254](#)
[aiogram.types.inline_query_result_cached_sticker, 256](#)
[aiogram.types.inline_query_result_cached_video, 258](#)
[aiogram.types.inline_query_result_cached_voice, 260](#)

`aiogram.types.inline_query_result_contact`, 263
`aiogram.types.inline_query_result_document`, 264
`aiogram.types.inline_query_result_game`, 267
`aiogram.types.inline_query_result_gif`, 267
`aiogram.types.inline_query_result_location`, 270
`aiogram.types.inline_query_result_mpeg4_gif`, 272
`aiogram.types.inline_query_result_photo`, 274
`aiogram.types.inline_query_result_venue`, 275
`aiogram.types.inline_query_result_video`, 277
`aiogram.types.inline_query_result_voice`, 280
`aiogram.types.inline_query_results_button`, 281
`aiogram.types.input_contact_message_content`, 282
`aiogram.types.input_file`, 146
`aiogram.types.input_invoice_message_content`, 282
`aiogram.types.input_location_message_content`, 285
`aiogram.types.input_media`, 146
`aiogram.types.input_media_animation`, 147
`aiogram.types.input_media_audio`, 148
`aiogram.types.input_media_document`, 149
`aiogram.types.input_media_photo`, 150
`aiogram.types.input_media_video`, 151
`aiogram.types.input_message_content`, 286
`aiogram.types.input_poll_option`, 152
`aiogram.types.input_sticker`, 289
`aiogram.types.input_text_message_content`, 286
`aiogram.types.input_venue_message_content`, 288
`aiogram.types.invoice`, 304
`aiogram.types.keyboard_button`, 153
`aiogram.types.keyboard_button_poll_type`, 154
`aiogram.types.keyboard_button_request_chat`, 155
`aiogram.types.keyboard_button_request_user`, 156
`aiogram.types.keyboard_button_request_users`, 157
`aiogram.types.labeled_price`, 304
`aiogram.types.link_preview_options`, 158
`aiogram.types.location`, 159
`aiogram.types.login_url`, 160
`aiogram.types.mask_position`, 290
`aiogram.types.maybe_inaccessible_message`, 160
`aiogram.types.menu_button`, 161
`aiogram.types.menu_button_commands`, 161
`aiogram.types.menu_button_default`, 162
`aiogram.types.menu_button_web_app`, 162
`aiogram.types.message`, 163
`aiogram.types.message_auto_delete_timer_changed`, 214
`aiogram.types.message_entity`, 215
`aiogram.types.message_id`, 216
`aiogram.types.message_origin`, 216
`aiogram.types.message_origin_channel`, 216
`aiogram.types.message_origin_chat`, 217
`aiogram.types.message_origin_hidden_user`, 218
`aiogram.types.message_origin_user`, 218
`aiogram.types.message_reaction_count_updated`, 219
`aiogram.types.message_reaction_updated`, 219
`aiogram.types.order_info`, 305
`aiogram.types.passport_data`, 295
`aiogram.types.passport_element_error`, 296
`aiogram.types.passport_element_error_data_field`, 296
`aiogram.types.passport_element_error_file`, 297
`aiogram.types.passport_element_error_files`, 298
`aiogram.types.passport_element_error_front_side`, 298
`aiogram.types.passport_element_error_reverse_side`, 299
`aiogram.types.passport_element_error_selfie`, 300
`aiogram.types.passport_element_error_translation_file`, 300
`aiogram.types.passport_element_error_translation_files`, 302
`aiogram.types.passport_element_error_unspecified`, 302
`aiogram.types.passport_file`, 303
`aiogram.types.photo_size`, 220
`aiogram.types.poll`, 221
`aiogram.types.poll_answer`, 222
`aiogram.types.poll_option`, 222
`aiogram.types.pre_checkout_query`, 305
`aiogram.types.proximity_alert_triggered`, 223
`aiogram.types.reaction_count`, 223
`aiogram.types.reaction_type`, 224
`aiogram.types.reaction_type_custom_emoji`, 224
`aiogram.types.reaction_type_emoji`, 224
`aiogram.types.reply_keyboard_markup`, 225
`aiogram.types.reply_keyboard_remove`, 226
`aiogram.types.reply_parameters`, 226
`aiogram.types.response_parameters`, 228
`aiogram.types.sent_web_app_message`, 289

- [aiogram.types.shared_user, 228](#)
- [aiogram.types.shipping_address, 307](#)
- [aiogram.types.shipping_option, 307](#)
- [aiogram.types.shipping_query, 308](#)
- [aiogram.types.sticker, 290](#)
- [aiogram.types.sticker_set, 292](#)
- [aiogram.types.story, 229](#)
- [aiogram.types.successful_payment, 309](#)
- [aiogram.types.switch_inline_query_chosen_chat, 229](#)
- [aiogram.types.text_quote, 230](#)
- [aiogram.types.update, 309](#)
- [aiogram.types.user, 231](#)
- [aiogram.types.user_chat_boosts, 232](#)
- [aiogram.types.user_profile_photos, 233](#)
- [aiogram.types.user_shared, 233](#)
- [aiogram.types.users_shared, 234](#)
- [aiogram.types.venue, 234](#)
- [aiogram.types.video, 235](#)
- [aiogram.types.video_chat_ended, 236](#)
- [aiogram.types.video_chat_participants_invited, 236](#)
- [aiogram.types.video_chat_scheduled, 236](#)
- [aiogram.types.video_chat_started, 237](#)
- [aiogram.types.video_note, 237](#)
- [aiogram.types.voice, 238](#)
- [aiogram.types.web_app_data, 238](#)
- [aiogram.types.web_app_info, 239](#)
- [aiogram.types.webhook_info, 312](#)
- [aiogram.types.write_access_allowed, 239](#)

Symbols

- `__call__()` (*aiogram.dispatcher.middlewares.base.BaseMiddleware* *memod*), 571
 - `__call__()` (*aiogram.filters.base.Filter* *memod*), 528
 - `__init__()` (*aiogram.dispatcher.dispatcher.Dispatcher* *memod*), 514
 - `__init__()` (*aiogram.dispatcher.router.Router* *memod*), 508
 - `__init__()` (*aiogram.filters.command.Command* *memod*), 518
 - `__init__()` (*aiogram.fsm.storage.memory.MemoryStorage* *memod*), 548
 - `__init__()` (*aiogram.fsm.storage.redis.RedisStorage* *memod*), 548
 - `__init__()` (*aiogram.types.input_file.BufferedInputFile* *memod*), 506
 - `__init__()` (*aiogram.types.input_file.FSInputFile* *memod*), 506
 - `__init__()` (*aiogram.utils.callback_answer.CallbackAnswer* *memod*), 600
 - `__init__()` (*aiogram.utils.callback_answer.CallbackAnswerMiddleware* *memod*), 600
 - `__init__()` (*aiogram.utils.chat_action.ChatActionSender* *memod*), 591
 - `__init__()` (*aiogram.utils.formatting.Text* *memod*), 605
 - `__init__()` (*aiogram.utils.i18n.middleware.ConstI18nMiddleware* *memod*), 588
 - `__init__()` (*aiogram.utils.i18n.middleware.FSMI18nMiddleware* *memod*), 588
 - `__init__()` (*aiogram.utils.i18n.middleware.I18nMiddleware* *memod*), 589
 - `__init__()` (*aiogram.utils.i18n.middleware.SimpleI18nMiddleware* *memod*), 588
 - `__init__()` (*aiogram.utils.keyboard.InlineKeyboardBuilder* *memod*), 583
 - `__init__()` (*aiogram.utils.keyboard.ReplyKeyboardBuilder* *memod*), 585
 - `__init__()` (*aiogram.webhook.aiohttp_server.BaseRequestHandler* *memod*), 532
 - `__init__()` (*aiogram.webhook.aiohttp_server.SimpleRequestHandler* *memod*), 533
 - `__init__()` (*aiogram.webhook.aiohttp_server.TokenBasedRequestHandler* *memod*), 533
 - `__init__()` (*aiogram.webhook.security.IPFilter* *memod*), 535
- ## A
- `accent_color_id` (*aiogram.types.chat.Chat* *ampubym*), 37
 - `accent_color_id` (*aiogram.types.chat_full_info.ChatFullInfo* *ampubym*), 59
 - `action` (*aiogram.methods.send_chat_action.SendChatAction* *ampubym*), 394
 - `actions` (*aiogram.fsm.scene.SceneConfig* *ampubym*), 566
 - `active_usernames` (*aiogram.types.chat.Chat* *ampubym*), 37
 - `active_usernames` (*aiogram.types.chat_full_info.ChatFullInfo* *ampubym*), 60
 - `actor_chat` (*aiogram.types.message_reaction_updated.MessageReactionUpdated* *ampubym*), 220
 - `add()` (*aiogram.fsm.scene.SceneRegistry* *memod*), 565
 - `add()` (*aiogram.utils.keyboard.InlineKeyboardBuilder* *memod*), 583
 - `add()` (*aiogram.utils.keyboard.ReplyKeyboardBuilder* *memod*), 585
 - `add()` (*aiogram.utils.media_group.MediaGroupBuilder* *memod*), 608
 - `add_audio()` (*aiogram.utils.media_group.MediaGroupBuilder* *memod*), 609
 - `add_date` (*aiogram.types.chat_boost.ChatBoost* *ampubym*), 54
 - `add_document()` (*aiogram.utils.media_group.MediaGroupBuilder* *memod*), 609

<code>add_photo()</code> (<i>aiogram.utils.media_group.MediaGroupBuilder</i> <i>method</i>), 610	<code>aiogram.enums.currency</code> module, 493
<code>add_to_router()</code> (<i>aiogram.fsm.scene.Scene</i> class <i>method</i>), 564	<code>aiogram.enums.dice_emoji</code> module, 496
<code>add_video()</code> (<i>aiogram.utils.media_group.MediaGroupBuilder</i> <i>method</i>), 610	<code>aiogram.enums.encrypted_passport_element</code> module, 496
<code>added_to_attachment_menu</code> (<i>aiogram.types.user.User</i> <i>attribute</i>), 231	<code>aiogram.enums.inline_query_result_type</code> module, 497
<code>added_to_attachment_menu</code> (<i>aiogram.utils.web_app.WebAppUser</i> <i>attribute</i>), 596	<code>aiogram.enums.input_media_type</code> module, 498
<code>additional_chat_count</code> (<i>aiogram.types.giveaway_winners.GiveawayWinner</i> <i>attribute</i>), 143	<code>aiogram.enums.keyboard_button_poll_type_type</code> module, 498
<code>ADDRESS</code> (<i>aiogram.enums.encrypted_passport_element.EncryptedPassportElement</i> <i>enum</i>), 497	<code>aiogram.enums.mask_position_point</code> module, 498
<code>address</code> (<i>aiogram.methods.send_venue.SendVenue</i> <i>attribute</i>), 417	<code>aiogram.enums.message_entity_type</code> module, 499
<code>address</code> (<i>aiogram.types.business_location.BusinessLocation</i> <i>attribute</i>), 32	<code>aiogram.enums.message_origin_type</code> module, 500
<code>address</code> (<i>aiogram.types.chat_location.ChatLocation</i> <i>attribute</i>), 104	<code>aiogram.enums.parse_mode</code> module, 500
<code>address</code> (<i>aiogram.types.inline_query_result_venue.InlineQueryResultVenue</i> <i>attribute</i>), 276	<code>aiogram.enums.encrypted_passport_element_error_type</code> module, 500
<code>address</code> (<i>aiogram.types.input_venue_message_content.InputVenueMessageContent</i> <i>attribute</i>), 288	<code>aiogram.enums.poll_type</code> module, 501
<code>address</code> (<i>aiogram.types.venue.Venue</i> <i>attribute</i>), 234	<code>aiogram.enums.reaction_type_type</code> module, 501
<code>AddStickerToSet</code> (клас в <i>aiogram.methods.add_sticker_to_set</i>), 315	<code>aiogram.enums.sticker_format</code> module, 501
<code>adjust()</code> (<i>aiogram.utils.keyboard.InlineKeyboardBuilder</i> <i>method</i>), 583	<code>aiogram.enums.sticker_type</code> module, 502
<code>adjust()</code> (<i>aiogram.utils.keyboard.ReplyKeyboardBuilder</i> <i>method</i>), 585	<code>aiogram.enums.topic_icon_color</code> module, 502
<code>ADMINISTRATOR</code> (<i>aiogram.enums.chat_member_status.ChatMemberStatus</i> <i>enum</i>), 490	<code>aiogram.enums.update_type</code> module, 502
<code>AED</code> (<i>aiogram.enums.currency.Currency</i> <i>enum</i>), 493	<code>aiogram.exceptions</code> module, 574
<code>AFN</code> (<i>aiogram.enums.currency.Currency</i> <i>enum</i>), 493	<code>aiogram.handlers.callback_query</code> module, 578
<code>aiogram.dispatcher.flags</code> module, 576	<code>aiogram.methods.add_sticker_to_set</code> module, 315
<code>aiogram.enums.bot_command_scope_type</code> module, 489	<code>aiogram.methods.answer_callback_query</code> module, 333
<code>aiogram.enums.chat_action</code> module, 489	<code>aiogram.methods.answer_inline_query</code> module, 465
<code>aiogram.enums.chat_boost_source_type</code> module, 490	<code>aiogram.methods.answer_pre_checkout_query</code> module, 474
<code>aiogram.enums.chat_member_status</code> module, 490	<code>aiogram.methods.answer_shipping_query</code> module, 475
<code>aiogram.enums.chat_type</code> module, 491	<code>aiogram.methods.answer_web_app_query</code> module, 468
<code>aiogram.enums.content_type</code> module, 491	<code>aiogram.methods.approve_chat_join_request</code> module, 335

<code>aiogram.methods.ban_chat_member</code> module, 336	<code>aiogram.methods.edit_message_reply_markup</code> module, 459
<code>aiogram.methods.ban_chat_sender_chat</code> module, 337	<code>aiogram.methods.edit_message_text</code> module, 460
<code>aiogram.methods.close</code> module, 339	<code>aiogram.methods.export_chat_invite_link</code> module, 357
<code>aiogram.methods.close_forum_topic</code> module, 340	<code>aiogram.methods.forward_message</code> module, 358
<code>aiogram.methods.close_general_forum_topic</code> module, 341	<code>aiogram.methods.forward_messages</code> module, 360
<code>aiogram.methods.copy_message</code> module, 342	<code>aiogram.methods.get_business_connection</code> module, 361
<code>aiogram.methods.copy_messages</code> module, 344	<code>aiogram.methods.get_chat</code> module, 362
<code>aiogram.methods.create_chat_invite_link</code> module, 346	<code>aiogram.methods.get_chat_administrators</code> module, 363
<code>aiogram.methods.create_forum_topic</code> module, 347	<code>aiogram.methods.get_chat_member</code> module, 364
<code>aiogram.methods.create_invoice_link</code> module, 476	<code>aiogram.methods.get_chat_member_count</code> module, 365
<code>aiogram.methods.create_new_sticker_set</code> module, 316	<code>aiogram.methods.get_chat_menu_button</code> module, 366
<code>aiogram.methods.decline_chat_join_request</code> module, 348	<code>aiogram.methods.get_custom_emoji_stickers</code> module, 319
<code>aiogram.methods.delete_chat_photo</code> module, 349	<code>aiogram.methods.get_file</code> module, 367
<code>aiogram.methods.delete_chat_sticker_set</code> module, 350	<code>aiogram.methods.get_forum_topic_icon_stickers</code> module, 368
<code>aiogram.methods.delete_forum_topic</code> module, 351	<code>aiogram.methods.get_game_high_scores</code> module, 469
<code>aiogram.methods.delete_message</code> module, 451	<code>aiogram.methods.get_me</code> module, 369
<code>aiogram.methods.delete_messages</code> module, 453	<code>aiogram.methods.get_my_commands</code> module, 370
<code>aiogram.methods.delete_my_commands</code> module, 352	<code>aiogram.methods.get_my_default_administrator_rights</code> module, 371
<code>aiogram.methods.delete_sticker_from_set</code> module, 317	<code>aiogram.methods.get_my_description</code> module, 372
<code>aiogram.methods.delete_sticker_set</code> module, 318	<code>aiogram.methods.get_my_name</code> module, 373
<code>aiogram.methods.delete_webhook</code> module, 482	<code>aiogram.methods.get_my_short_description</code> module, 374
<code>aiogram.methods.edit_chat_invite_link</code> module, 354	<code>aiogram.methods.get_sticker_set</code> module, 320
<code>aiogram.methods.edit_forum_topic</code> module, 355	<code>aiogram.methods.get_updates</code> module, 483
<code>aiogram.methods.edit_general_forum_topic</code> module, 356	<code>aiogram.methods.get_user_chat_boosts</code> module, 374
<code>aiogram.methods.edit_message_caption</code> module, 454	<code>aiogram.methods.get_user_profile_photos</code> module, 375
<code>aiogram.methods.edit_message_live_location</code> module, 455	<code>aiogram.methods.get_webhook_info</code> module, 485
<code>aiogram.methods.edit_message_media</code> module, 458	<code>aiogram.methods.hide_general_forum_topic</code> module, 376

aiogram.methods.leave_chat module, 377	aiogram.methods.set_chat_administrator_custom_title module, 428
aiogram.methods.log_out module, 378	aiogram.methods.set_chat_description module, 429
aiogram.methods.pin_chat_message module, 379	aiogram.methods.set_chat_menu_button module, 430
aiogram.methods.promote_chat_member module, 380	aiogram.methods.set_chat_permissions module, 432
aiogram.methods.reopen_forum_topic module, 383	aiogram.methods.set_chat_photo module, 433
aiogram.methods.reopen_general_forum_topic module, 384	aiogram.methods.set_chat_sticker_set module, 434
aiogram.methods.replace_sticker_in_set module, 321	aiogram.methods.set_chat_title module, 435
aiogram.methods.restrict_chat_member module, 385	aiogram.methods.set_custom_emoji_sticker_set_thumbnail module, 324
aiogram.methods.revoke_chat_invite_link module, 387	aiogram.methods.set_game_score module, 472
aiogram.methods.send_animation module, 388	aiogram.methods.set_message_reaction module, 436
aiogram.methods.send_audio module, 391	aiogram.methods.set_my_commands module, 438
aiogram.methods.send_chat_action module, 394	aiogram.methods.set_my_default_administrator_rights module, 439
aiogram.methods.send_contact module, 395	aiogram.methods.set_my_description module, 440
aiogram.methods.send_dice module, 397	aiogram.methods.set_my_name module, 441
aiogram.methods.send_document module, 400	aiogram.methods.set_my_short_description module, 442
aiogram.methods.send_game module, 470	aiogram.methods.set_passport_data_errors module, 487
aiogram.methods.send_invoice module, 479	aiogram.methods.set_sticker_emoji_list module, 326
aiogram.methods.send_location module, 402	aiogram.methods.set_sticker_keywords module, 327
aiogram.methods.send_media_group module, 405	aiogram.methods.set_sticker_mask_position module, 328
aiogram.methods.send_message module, 408	aiogram.methods.set_sticker_position_in_set module, 329
aiogram.methods.send_photo module, 411	aiogram.methods.set_sticker_set_thumbnail module, 330
aiogram.methods.send_poll module, 413	aiogram.methods.set_sticker_set_title module, 331
aiogram.methods.send_sticker module, 322	aiogram.methods.set_webhook module, 485
aiogram.methods.send_venue module, 417	aiogram.methods.stop_message_live_location module, 462
aiogram.methods.send_video module, 420	aiogram.methods.stop_poll module, 464
aiogram.methods.send_video_note module, 423	aiogram.methods.unban_chat_member module, 443
aiogram.methods.send_voice module, 426	aiogram.methods.unban_chat_sender_chat module, 445

aiogram.methods.unhide_general_forum_topic module, 446	aiogram.types.bot_description module, 30
aiogram.methods.unpin_all_chat_messages module, 447	aiogram.types.bot_name module, 31
aiogram.methods.unpin_all_forum_topic_messages module, 448	aiogram.types.bot_short_description module, 31
aiogram.methods.unpin_all_general_forum_topic_messages module, 449	aiogram.types.business_connection module, 31
aiogram.methods.unpin_chat_message module, 450	aiogram.types.business_intro module, 32
aiogram.methods.upload_sticker_file module, 332	aiogram.types.business_location module, 32
aiogram.types.animation module, 19	aiogram.types.business_messages_deleted module, 33
aiogram.types.audio module, 20	aiogram.types.business_opening_hours module, 33
aiogram.types.background_fill module, 21	aiogram.types.business_opening_hours_interval module, 34
aiogram.types.background_fill_freeform_gradient module, 21	aiogram.types.callback_game module, 313
aiogram.types.background_fill_gradient module, 22	aiogram.types.callback_query module, 34
aiogram.types.background_fill_solid module, 22	aiogram.types.chat module, 36
aiogram.types.background_type module, 23	aiogram.types.chat_administrator_rights module, 52
aiogram.types.background_type_chat_theme module, 23	aiogram.types.chat_background module, 54
aiogram.types.background_type_fill module, 23	aiogram.types.chat_boost module, 54
aiogram.types.background_type_pattern module, 24	aiogram.types.chat_boost_added module, 55
aiogram.types.background_type_wallpaper module, 25	aiogram.types.chat_boost_removed module, 55
aiogram.types.birthdate module, 25	aiogram.types.chat_boost_source module, 56
aiogram.types.bot_command module, 26	aiogram.types.chat_boost_source_gift_code module, 56
aiogram.types.bot_command_scope module, 26	aiogram.types.chat_boost_source_giveaway module, 56
aiogram.types.bot_command_scope_all_chat_administrators module, 27	aiogram.types.chat_boost_source_premium module, 57
aiogram.types.bot_command_scope_all_group_chats module, 27	aiogram.types.chat_boost_updated module, 58
aiogram.types.bot_command_scope_all_private_chats module, 28	aiogram.types.chat_full_info module, 58
aiogram.types.bot_command_scope_chat module, 28	aiogram.types.chat_invite_link module, 62
aiogram.types.bot_command_scope_chat_administrators module, 29	aiogram.types.chat_join_request module, 63
aiogram.types.bot_command_scope_chat_member module, 29	aiogram.types.chat_location module, 103
aiogram.types.bot_command_scope_default module, 30	aiogram.types.chat_member module, 104

<code>aiogram.types.chat_member_administrator</code> module, 105	<code>aiogram.types.general_forum_topic_hidden</code> module, 140
<code>aiogram.types.chat_member_banned</code> module, 107	<code>aiogram.types.general_forum_topic_unhidden</code> module, 141
<code>aiogram.types.chat_member_left</code> module, 107	<code>aiogram.types.giveaway</code> module, 141
<code>aiogram.types.chat_member_member</code> module, 108	<code>aiogram.types.giveaway_completed</code> module, 142
<code>aiogram.types.chat_member_owner</code> module, 108	<code>aiogram.types.giveaway_created</code> module, 142
<code>aiogram.types.chat_member_restricted</code> module, 109	<code>aiogram.types.giveaway_winners</code> module, 142
<code>aiogram.types.chat_member_updated</code> module, 110	<code>aiogram.types.inaccessible_message</code> module, 143
<code>aiogram.types.chat_permissions</code> module, 130	<code>aiogram.types.inline_keyboard_button</code> module, 144
<code>aiogram.types.chat_photo</code> module, 132	<code>aiogram.types.inline_keyboard_markup</code> module, 145
<code>aiogram.types.chat_shared</code> module, 132	<code>aiogram.types.inline_query</code> module, 240
<code>aiogram.types.chosen_inline_result</code> module, 240	<code>aiogram.types.inline_query_result</code> module, 242
<code>aiogram.types.contact</code> module, 133	<code>aiogram.types.inline_query_result_article</code> module, 243
<code>aiogram.types.dice</code> module, 134	<code>aiogram.types.inline_query_result_audio</code> module, 244
<code>aiogram.types.document</code> module, 134	<code>aiogram.types.inline_query_result_cached_audio</code> module, 246
<code>aiogram.types.encrypted_credentials</code> module, 293	<code>aiogram.types.inline_query_result_cached_document</code> module, 248
<code>aiogram.types.encrypted_passport_element</code> module, 294	<code>aiogram.types.inline_query_result_cached_gif</code> module, 250
<code>aiogram.types.error_event</code> module, 573	<code>aiogram.types.inline_query_result_cached_mpeg4_gif</code> module, 252
<code>aiogram.types.external_reply_info</code> module, 135	<code>aiogram.types.inline_query_result_cached_photo</code> module, 254
<code>aiogram.types.file</code> module, 137	<code>aiogram.types.inline_query_result_cached_sticker</code> module, 256
<code>aiogram.types.force_reply</code> module, 138	<code>aiogram.types.inline_query_result_cached_video</code> module, 258
<code>aiogram.types.forum_topic</code> module, 138	<code>aiogram.types.inline_query_result_cached_voice</code> module, 260
<code>aiogram.types.forum_topic_closed</code> module, 139	<code>aiogram.types.inline_query_result_contact</code> module, 263
<code>aiogram.types.forum_topic_created</code> module, 139	<code>aiogram.types.inline_query_result_document</code> module, 264
<code>aiogram.types.forum_topic_edited</code> module, 140	<code>aiogram.types.inline_query_result_game</code> module, 267
<code>aiogram.types.forum_topic_reopened</code> module, 140	<code>aiogram.types.inline_query_result_gif</code> module, 267
<code>aiogram.types.game</code> module, 313	<code>aiogram.types.inline_query_result_location</code> module, 270
<code>aiogram.types.game_high_score</code> module, 314	<code>aiogram.types.inline_query_result_mpeg4_gif</code> module, 272

<code>aiogram.types.inline_query_result_photo</code> module, 274	<code>aiogram.types.link_preview_options</code> module, 158
<code>aiogram.types.inline_query_result_venue</code> module, 275	<code>aiogram.types.location</code> module, 159
<code>aiogram.types.inline_query_result_video</code> module, 277	<code>aiogram.types.login_url</code> module, 160
<code>aiogram.types.inline_query_result_voice</code> module, 280	<code>aiogram.types.mask_position</code> module, 290
<code>aiogram.types.inline_query_results_button</code> module, 281	<code>aiogram.types.maybe_inaccessible_message</code> module, 160
<code>aiogram.types.input_contact_message_content</code> module, 282	<code>aiogram.types.menu_button</code> module, 161
<code>aiogram.types.input_file</code> module, 146	<code>aiogram.types.menu_button_commands</code> module, 161
<code>aiogram.types.input_invoice_message_content</code> module, 282	<code>aiogram.types.menu_button_default</code> module, 162
<code>aiogram.types.input_location_message_content</code> module, 285	<code>aiogram.types.menu_button_web_app</code> module, 162
<code>aiogram.types.input_media</code> module, 146	<code>aiogram.types.message</code> module, 163
<code>aiogram.types.input_media_animation</code> module, 147	<code>aiogram.types.message_auto_delete_timer_changed</code> module, 214
<code>aiogram.types.input_media_audio</code> module, 148	<code>aiogram.types.message_entity</code> module, 215
<code>aiogram.types.input_media_document</code> module, 149	<code>aiogram.types.message_id</code> module, 216
<code>aiogram.types.input_media_photo</code> module, 150	<code>aiogram.types.message_origin</code> module, 216
<code>aiogram.types.input_media_video</code> module, 151	<code>aiogram.types.message_origin_channel</code> module, 216
<code>aiogram.types.input_message_content</code> module, 286	<code>aiogram.types.message_origin_chat</code> module, 217
<code>aiogram.types.input_poll_option</code> module, 152	<code>aiogram.types.message_origin_hidden_user</code> module, 218
<code>aiogram.types.input_sticker</code> module, 289	<code>aiogram.types.message_origin_user</code> module, 218
<code>aiogram.types.input_text_message_content</code> module, 286	<code>aiogram.types.message_reaction_count_updated</code> module, 219
<code>aiogram.types.input_venue_message_content</code> module, 288	<code>aiogram.types.message_reaction_updated</code> module, 219
<code>aiogram.types.invoice</code> module, 304	<code>aiogram.types.order_info</code> module, 305
<code>aiogram.types.keyboard_button</code> module, 153	<code>aiogram.types.passport_data</code> module, 295
<code>aiogram.types.keyboard_button_poll_type</code> module, 154	<code>aiogram.types.passport_element_error</code> module, 296
<code>aiogram.types.keyboard_button_request_chat</code> module, 155	<code>aiogram.types.passport_element_error_data_field</code> module, 296
<code>aiogram.types.keyboard_button_request_user</code> module, 156	<code>aiogram.types.passport_element_error_file</code> module, 297
<code>aiogram.types.keyboard_button_request_users</code> module, 157	<code>aiogram.types.passport_element_error_files</code> module, 298
<code>aiogram.types.labeled_price</code> module, 304	<code>aiogram.types.passport_element_error_front_side</code> module, 298

aiogram.types.passport_element_error_reverse_side	aiogram.types.story
module, 299	module, 229
aiogram.types.passport_element_error_selfie	aiogram.types.successful_payment
module, 300	module, 309
aiogram.types.passport_element_error_translation_file	aiogram.types.switch_inline_query_chosen_chat
module, 300	module, 229
aiogram.types.passport_element_error_translation_file	aiogram.types.text_quote
module, 302	module, 230
aiogram.types.passport_element_error_unspecified_file	aiogram.types.update
module, 302	module, 309
aiogram.types.passport_file	aiogram.types.user
module, 303	module, 231
aiogram.types.photo_size	aiogram.types.user_chat_boosts
module, 220	module, 232
aiogram.types.poll	aiogram.types.user_profile_photos
module, 221	module, 233
aiogram.types.poll_answer	aiogram.types.user_shared
module, 222	module, 233
aiogram.types.poll_option	aiogram.types.users_shared
module, 222	module, 234
aiogram.types.pre_checkout_query	aiogram.types.venue
module, 305	module, 234
aiogram.types.proximity_alert_triggered	aiogram.types.video
module, 223	module, 235
aiogram.types.reaction_count	aiogram.types.video_chat_ended
module, 223	module, 236
aiogram.types.reaction_type	aiogram.types.video_chat_participants_invited
module, 224	module, 236
aiogram.types.reaction_type_custom_emoji	aiogram.types.video_chat_scheduled
module, 224	module, 236
aiogram.types.reaction_type_emoji	aiogram.types.video_chat_started
module, 224	module, 237
aiogram.types.reply_keyboard_markup	aiogram.types.video_note
module, 225	module, 237
aiogram.types.reply_keyboard_remove	aiogram.types.voice
module, 226	module, 238
aiogram.types.reply_parameters	aiogram.types.web_app_data
module, 226	module, 238
aiogram.types.response_parameters	aiogram.types.web_app_info
module, 228	module, 239
aiogram.types.sent_web_app_message	aiogram.types.webhook_info
module, 289	module, 312
aiogram.types.shared_user	aiogram.types.write_access_allowed
module, 228	module, 239
aiogram.types.shipping_address	AiogramError, 574
module, 307	AiohttpSession (клас в aiogram.client.session.aiohttp), 16
aiogram.types.shipping_option	ALL (aiogram.enums.currency.Currency аmpубым), 493
module, 307	ALL_CHAT_ADMINISTRATORS (aiogram.enums.bot_command_scope_type.BotCommandScope аmpубым), 489
aiogram.types.shipping_query	ALL_GROUP_CHATS (aiogram.enums.bot_command_scope_type.BotCommandScope аmpубым), 489
module, 308	
aiogram.types.sticker	
module, 290	
aiogram.types.sticker_set	
module, 292	

<code>ampubym</code>), 489	<code>ampubym</code>), 323
<code>ALL_PRIVATE_CHATS</code> (aiogram.enums.bot_command_scope_type.BotCommandScopeType), 489	<code>allow_sending_without_reply</code> (aiogram.methods.send_venue.SendVenue), 418
<code>allow_bot_chats</code> (aiogram.types.switch_inline_query_chosen_chat.SwitchInlineQueryChosenChat), 230	<code>allow_sending_without_reply</code> (aiogram.methods.send_video.SendVideo), 421
<code>allow_channel_chats</code> (aiogram.types.switch_inline_query_chosen_chat.SwitchInlineQueryChosenChat), 230	<code>allow_sending_without_reply</code> (aiogram.methods.send_video_note.SendVideoNote), 424
<code>allow_group_chats</code> (aiogram.types.switch_inline_query_chosen_chat.SwitchInlineQueryChosenChat), 230	<code>allow_sending_without_reply</code> (aiogram.methods.send_voice.SendVoice), 427
<code>allow_sending_without_reply</code> (aiogram.methods.copy_message.CopyMessage), 343	<code>allow_sending_without_reply</code> (aiogram.types.reply_parameters.ReplyParameters), 227
<code>allow_sending_without_reply</code> (aiogram.methods.send_animation.SendAnimation), 390	<code>allow_user_chats</code> (aiogram.types.switch_inline_query_chosen_chat.SwitchInlineQueryChosenChat), 229
<code>allow_sending_without_reply</code> (aiogram.methods.send_audio.SendAudio), 392	<code>allowed_updates</code> (aiogram.methods.get_updates.GetUpdates), 484
<code>allow_sending_without_reply</code> (aiogram.methods.send_contact.SendContact), 396	<code>allowed_updates</code> (aiogram.methods.set_webhook.SetWebhook), 486
<code>allow_sending_without_reply</code> (aiogram.methods.send_dice.SendDice), 398	<code>allowed_updates</code> (aiogram.types.webhook_info.WebhookInfo), 313
<code>allow_sending_without_reply</code> (aiogram.methods.send_document.SendDocument), 401	<code>allows_multiple_answers</code> (aiogram.methods.send_poll.SendPoll), 415
<code>allow_sending_without_reply</code> (aiogram.methods.send_game.SendGame), 471	<code>allows_multiple_answers</code> (aiogram.types.poll.Poll), 221
<code>allow_sending_without_reply</code> (aiogram.methods.send_invoice.SendInvoice), 481	<code>allows_write_to_pm</code> (aiogram.utils.web_app.WebAppUser), 597
<code>allow_sending_without_reply</code> (aiogram.methods.send_location.SendLocation), 404	<code>AMD</code> (aiogram.enums.currency.Currency), 493
<code>allow_sending_without_reply</code> (aiogram.methods.send_media_group.SendMediaGroup), 406	<code>amount</code> (aiogram.types.labeled_price.LabeledPrice), 305
<code>allow_sending_without_reply</code> (aiogram.methods.send_message.SendMessage), 409	<code>ANIMATED</code> (aiogram.enums.sticker_format.StickerFormat), 501
<code>allow_sending_without_reply</code> (aiogram.methods.send_photo.SendPhoto), 412	<code>ANIMATION</code> (aiogram.enums.content_type.ContentType), 491
<code>allow_sending_without_reply</code> (aiogram.methods.send_poll.SendPoll), 416	<code>ANIMATION</code> (aiogram.enums.input_media_type.InputMediaType), 498
<code>allow_sending_without_reply</code> (aiogram.methods.send_sticker.SendSticker), 416	<code>animation</code> (aiogram.methods.send_animation.SendAnimation), 389
	<code>animation</code> (aiogram.types.external_reply_info.ExternalReplyInfo), 136
	<code>animation</code> (aiogram.types.game.Game), 314
	<code>animation</code> (aiogram.types.message.Message), 166

<code>Animation</code> (класс в <code>aiogram.types.animation</code>), 19	<code>mod)</code> , 196
<code>answer()</code> (<code>aiogram.types.callback_query.CallbackQuery</code> memod), 35	<code>answer_dice_pm()</code> (<code>aiogram.types.chat_join_request.ChatJoinRequest</code> memod), 91
<code>answer()</code> (<code>aiogram.types.chat_join_request.ChatJoinRequest</code> memod), 64	<code>answer_document()</code> (<code>aiogram.types.chat_member_updated.ChatMemberUpdated</code> memod), 73
<code>answer()</code> (<code>aiogram.types.chat_member_updated.ChatMemberUpdated</code> memod), 111	<code>answer_document()</code> (<code>aiogram.types.chat_member_updated.ChatMemberUpdated</code> memod), 116
<code>answer()</code> (<code>aiogram.types.inline_query.InlineQuery</code> memod), 241	<code>answer_document_pm()</code> (<code>aiogram.types.message.Message</code> memod), 179
<code>answer()</code> (<code>aiogram.types.message.Message</code> memod), 189	<code>answer_document_pm()</code> (<code>aiogram.types.chat_join_request.ChatJoinRequest</code> memod), 75
<code>answer()</code> (<code>aiogram.types.pre_checkout_query.PreCheckoutQuery</code> memod), 306	<code>answer_game()</code> (<code>aiogram.types.chat_join_request.ChatJoinRequest</code> memod), 76
<code>answer()</code> (<code>aiogram.types.shipping_query.ShippingQuery</code> memod), 308	<code>answer_game()</code> (<code>aiogram.types.chat_member_updated.ChatMemberUpdated</code> memod), 117
<code>answer_animation()</code> (<code>aiogram.types.chat_join_request.ChatJoinRequest</code> memod), 66	<code>answer_game()</code> (<code>aiogram.types.message.Message</code> memod), 181
<code>answer_animation()</code> (<code>aiogram.types.chat_member_updated.ChatMemberUpdated</code> memod), 112	<code>answer_game_pm()</code> (<code>aiogram.types.chat_join_request.ChatJoinRequest</code> memod), 77
<code>answer_animation()</code> (<code>aiogram.types.message.Message</code> memod), 172	<code>answer_invoice()</code> (<code>aiogram.types.chat_join_request.ChatJoinRequest</code> memod), 78
<code>answer_animation_pm()</code> (<code>aiogram.types.chat_join_request.ChatJoinRequest</code> memod), 67	<code>answer_invoice()</code> (<code>aiogram.types.chat_member_updated.ChatMemberUpdated</code> memod), 118
<code>answer_audio()</code> (<code>aiogram.types.chat_join_request.ChatJoinRequest</code> memod), 69	<code>answer_invoice()</code> (<code>aiogram.types.message.Message</code> memod), 183
<code>answer_audio()</code> (<code>aiogram.types.chat_member_updated.ChatMemberUpdated</code> memod), 113	<code>answer_invoice_pm()</code> (<code>aiogram.types.chat_join_request.ChatJoinRequest</code> memod), 79
<code>answer_audio()</code> (<code>aiogram.types.message.Message</code> memod), 174	<code>answer_location()</code> (<code>aiogram.types.chat_join_request.ChatJoinRequest</code> memod), 81
<code>answer_audio_pm()</code> (<code>aiogram.types.chat_join_request.ChatJoinRequest</code> memod), 70	<code>answer_location()</code> (<code>aiogram.types.chat_member_updated.ChatMemberUpdated</code> memod), 119
<code>answer_contact()</code> (<code>aiogram.types.chat_join_request.ChatJoinRequest</code> memod), 71	<code>answer_location_pm()</code> (<code>aiogram.types.chat_join_request.ChatJoinRequest</code> memod), 83
<code>answer_contact()</code> (<code>aiogram.types.chat_member_updated.ChatMemberUpdated</code> memod), 115	<code>answer_media_group()</code> (<code>aiogram.types.chat_join_request.ChatJoinRequest</code> memod), 84
<code>answer_contact()</code> (<code>aiogram.types.message.Message</code> memod), 177	<code>answer_media_group()</code> (<code>aiogram.types.chat_member_updated.ChatMemberUpdated</code> memod), 121
<code>answer_contact_pm()</code> (<code>aiogram.types.chat_join_request.ChatJoinRequest</code> memod), 72	
<code>answer_dice()</code> (<code>aiogram.types.chat_join_request.ChatJoinRequest</code> memod), 90	
<code>answer_dice()</code> (<code>aiogram.types.chat_member_updated.ChatMemberUpdated</code> memod), 124	
<code>answer_dice()</code> (<code>aiogram.types.message.Message</code> memod), 186	

answer_media_group() (aiogram.types.message.Message memod), 188
answer_media_group_pm() (aiogram.types.chat_join_request.ChatJoinRequest memod), 84
answer_photo() (aiogram.types.chat_join_request.ChatJoinRequest memod), 85
answer_photo() (aiogram.types.chat_member_updated.ChatMemberUpdated memod), 121
answer_photo() (aiogram.types.message.Message memod), 191
answer_photo_pm() (aiogram.types.chat_join_request.ChatJoinRequest memod), 86
answer_pm() (aiogram.types.chat_join_request.ChatJoinRequest memod), 65
answer_poll() (aiogram.types.chat_join_request.ChatJoinRequest memod), 87
answer_poll() (aiogram.types.chat_member_updated.ChatMemberUpdated memod), 122
answer_poll() (aiogram.types.message.Message memod), 193
answer_poll_pm() (aiogram.types.chat_join_request.ChatJoinRequest memod), 89
answer_sticker() (aiogram.types.chat_join_request.ChatJoinRequest memod), 92
answer_sticker() (aiogram.types.chat_member_updated.ChatMemberUpdated memod), 125
answer_sticker() (aiogram.types.message.Message memod), 197
answer_sticker_pm() (aiogram.types.chat_join_request.ChatJoinRequest memod), 93
answer_venue() (aiogram.types.chat_join_request.ChatJoinRequest memod), 94
answer_venue() (aiogram.types.chat_member_updated.ChatMemberUpdated memod), 126
answer_venue() (aiogram.types.message.Message memod), 199
answer_venue_pm() (aiogram.types.chat_join_request.ChatJoinRequest memod), 95
answer_video() (aiogram.types.chat_join_request.ChatJoinRequest memod), 96
answer_video() (aiogram.types.chat_member_updated.ChatMemberUpdated memod), 127
answer_video() (aiogram.types.message.Message memod), 201
answer_video_note() (aiogram.types.chat_join_request.ChatJoinRequest memod), 99
answer_video_note() (aiogram.types.chat_member_updated.ChatMemberUpdated memod), 128
answer_video_note() (aiogram.types.message.Message memod), 204
answer_video_note_pm() (aiogram.types.chat_join_request.ChatJoinRequest memod), 100
answer_video_pm() (aiogram.types.chat_join_request.ChatJoinRequest memod), 97
answer_voice() (aiogram.types.chat_join_request.ChatJoinRequest memod), 101
answer_voice() (aiogram.types.chat_member_updated.ChatMemberUpdated memod), 129
answer_voice() (aiogram.types.message.Message memod), 206
answer_voice_pm() (aiogram.types.chat_join_request.ChatJoinRequest memod), 102
AnswerCallbackQuery (клас в aiogram.methods.answer_callback_query), 483
answered (aiogram.utils.callback_answer.CallbackAnswer property), 600
AnswerInlineQuery (клас в aiogram.methods.answer_inline_query), 465
AnswerPreCheckoutQuery (клас в aiogram.methods.answer_pre_checkout_query), 474
AnswerShippingQuery (клас в aiogram.methods.answer_shipping_query), 475
AnswerWebAppQuery (клас в aiogram.methods.answer_web_app_query), 468
ANY (aiogram.enums.content_type.ContentType ампубым), 491
api_url() (aiogram.client.telegram.TelegramAPIServer ампубым), 14
approve() (aiogram.types.chat_join_request.ChatJoinRequest memod), 64
ApproveChatJoinRequest (клас в ai-

ogram.methods.approve_chat_join_request), auth_date (aiogram.utils.web_app.WebAppInitData
 335 ampubym), 596
 args (aiogram.filters.command.CommandObject author_signature (aiogram.types.message.Message
 ampubym), 519 ampubym), 166
 ARS (aiogram.enums.currency.Currency ampubym), author_signature (ai-
 493 ogram.types.message_origin_channel.MessageOriginChannel
 ARTICLE (aiogram.enums.inline_query_result_type.InlineQueryResultType), 217
 ampubym), 497 author_signature (ai-
 as_handler() (aiogram.fsm.scene.Scene class ogram.types.message_origin_chat.MessageOriginChat
 method), 565 ampubym), 217
 as_html() (aiogram.utils.formatting.Text memod), available_reactions (aiogram.types.chat.Chat
 605 ampubym), 37
 as_key_value() (в модули ai- available_reactions (ai-
 ogram.utils.formatting), 603 ogram.types.chat_full_info.ChatFullInfo
 as_kwargs() (aiogram.utils.formatting.Text ме- ampubym), 60
 mod), 605
 as_line() (в модули aiogram.utils.formatting), 602
 as_list() (в модули aiogram.utils.formatting), 602
 as_markdown() (aiogram.utils.formatting.Text ме-
 mod), 605
 as_marked_list() (в модули ai-
 ogram.utils.formatting), 602
 as_marked_section() (в модули ai-
 ogram.utils.formatting), 603
 as_numbered_list() (в модули ai-
 ogram.utils.formatting), 602
 as_numbered_section() (в модули ai-
 ogram.utils.formatting), 603
 as_router() (aiogram.fsm.scene.Scene class
 method), 565
 as_section() (в модули aiogram.utils.formatting),
 602
 AUD (aiogram.enums.currency.Currency ampubym),
 493
 AUDIO (aiogram.enums.content_type.ContentType ampubym), 491
 AUDIO (aiogram.enums.inline_query_result_type.InlineQueryResultType ampubym), 497
 AUDIO (aiogram.enums.input_media_type.InputMediaType ampubym), 498
 audio (aiogram.methods.send_audio.SendAudio
 ampubym), 391
 audio (aiogram.types.external_reply_info.ExternalReplyInfo
 ampubym), 136
 audio (aiogram.types.message.Message ampubym),
 166
 Audio (клас в aiogram.types.audio), 20
 audio_duration (ai-
 ogram.types.inline_query_result_audio.InlineQueryResultAudio
 ampubym), 246
 audio_file_id (aiogram.types.inline_query_result_cached_audio.InlineQueryResultCachedAudio
 ampubym), 248
 audio_url (aiogram.types.inline_query_result_audio.InlineQueryResultAudio ampubym), 245
 auth_date (aiogram.utils.web_app.WebAppInitData
 ampubym), 596
 author_signature (aiogram.types.message.Message
 ampubym), 166
 author_signature (ai-
 ogram.types.message_origin_channel.MessageOriginChannel
 ampubym), 217
 author_signature (ai-
 ogram.types.message_origin_chat.MessageOriginChat
 ampubym), 217
 available_reactions (aiogram.types.chat.Chat
 ampubym), 37
 available_reactions (ai-
 ogram.types.chat_full_info.ChatFullInfo
 ampubym), 60
 AZN (aiogram.enums.currency.Currency ampubym),
 493
 B
 back() (aiogram.fsm.scene.SceneWizard memod),
 567
 background_custom_emoji_id (ai-
 ogram.types.chat.Chat ampubym), 37
 background_custom_emoji_id (ai-
 ogram.types.chat_full_info.ChatFullInfo
 ampubym), 60
 BackgroundFill (клас в ai-
 ogram.types.background_fill), 21
 BackgroundFillFreeformGradient (клас в ai-
 ogram.types.background_fill_freeform_gradient),
 21
 BackgroundFillGradient (клас в ai-
 ogram.types.background_fill_gradient),
 22
 BackgroundFillSolid (клас в ai-
 ogram.types.background_fill_solid), 22
 BackgroundType (клас в ai-
 ogram.types.background_type), 23
 BackgroundTypeChatTheme (клас в ai-
 ogram.types.background_type_chat_theme),
 23
 BackgroundTypeFill (клас в ai-
 ogram.types.background_type_fill), 23
 BackgroundTypePattern (клас в ai-
 ogram.types.background_type_pattern),
 24
 BackgroundTypeWallpaper (клас в ai-
 ogram.types.background_type_wallpaper),
 24
 BAM (aiogram.enums.currency.Currency ampubym),
 493
 ban() (aiogram.types.chat.Chat memod), 50
 ban_sender_chat() (aiogram.types.chat.Chat ме-
 mod), 41

BanChatMember (клас в aiogram.methods.ban_chat_member), 336	boost_count (aiogram.types.chat_boost_added.ChatBoostAdded ampубym), 55
BanChatSenderChat (клас в aiogram.methods.ban_chat_sender_chat), 337	boost_id (aiogram.types.chat_boost.ChatBoost ampубym), 54
BANK_STATEMENT (aiogram.enums.encrypted_passport_element.EncryptedPassportElement ampубym), 497	boost_id (aiogram.types.chat_boost_removed.ChatBoostRemoved ampубym), 55
base (aiogram.client.telegram.TelegramAPIServer ampубym), 14	boosts (aiogram.types.user_chat_boosts.UserChatBoosts ampубym), 232
BaseMiddleware (клас в aiogram.dispatcher.middlewares.base), 571	bot_administrator_rights (aiogram.types.keyboard_button_request_chat.KeyboardButton ampубym), 156
BaseRequestHandler (клас в aiogram.webhook.aiohttp_server), 532	BOT_COMMAND (aiogram.enums.message_entity_type.MessageEntityType ampубym), 499
BaseSession (клас в aiogram.client.session.base), 15	bot_is_member (aiogram.types.keyboard_button_request_chat.KeyboardButton ampубym), 156
BaseStorage (клас в aiogram.fsm.storage.base), 550	bot_username (aiogram.types.login_url.LoginUrl ampубym), 160
BASKETBALL (aiogram.enums.dice_emoji.DiceEmoji ampубym), 496	BotCommand (клас в aiogram.types.bot_command), 26
BASKETBALL (aiogram.types.dice.DiceEmoji ampубym), 134	BotCommand (клас в aiogram.utils.formatting), 606
BDT (aiogram.enums.currency.Currency ampубym), 493	BotCommandScope (клас в aiogram.types.bot_command_scope), 26
BGN (aiogram.enums.currency.Currency ampубym), 493	BotCommandScopeAllChatAdministrators (клас в aiogram.types.bot_command_scope_all_chat_administrators), 27
big_file_id (aiogram.types.chat_photo.ChatPhoto ampубym), 132	BotCommandScopeAllGroupChats (клас в aiogram.types.bot_command_scope_all_group_chats), 27
big_file_unique_id (aiogram.types.chat_photo.ChatPhoto ampубym), 132	BotCommandScopeAllPrivateChats (клас в aiogram.types.bot_command_scope_all_private_chats), 28
bio (aiogram.types.chat.Chat ampубym), 37	BotCommandScopeChat (клас в aiogram.types.bot_command_scope_chat), 28
bio (aiogram.types.chat_full_info.ChatFullInfo ampубym), 61	BotCommandScopeChatAdministrators (клас в aiogram.types.bot_command_scope_chat_administrators), 29
bio (aiogram.types.chat_join_request.ChatJoinRequest ampубym), 63	BotCommandScopeChatMember (клас в aiogram.types.bot_command_scope_chat_member), 29
birthdate (aiogram.types.chat.Chat ampубym), 37	BotCommandScopeDefault (клас в aiogram.types.bot_command_scope_default), 30
birthdate (aiogram.types.chat_full_info.ChatFullInfo ampубym), 60	BotCommandScopeType (клас в aiogram.enums.bot_command_scope_type), 489
Birthdate (клас в aiogram.types.birthdate), 25	BotDescription (клас в aiogram.types.bot_description), 30
BLOCKQUOTE (aiogram.enums.message_entity_type.MessageEntityType ampубym), 499	BotName (клас в aiogram.types.bot_name), 31
BLUE (aiogram.enums.topic_icon_color.TopicIconColor ampубym), 502	BotShortDescription (клас в aiogram.types.bot_short_description), 31
BND (aiogram.enums.currency.Currency ampубym), 493	bottom_color (aiogram.types.background_fill_gradient.BackgroundFillGradient ampубym), 55
BOB (aiogram.enums.currency.Currency ampубym), 493	
BOLD (aiogram.enums.message_entity_type.MessageEntityType ampубym), 499	
Bold (клас в aiogram.utils.formatting), 606	
boost (aiogram.types.chat_boost_updated.ChatBoostUpdated ampубym), 58	
BOOST_ADDED (aiogram.enums.content_type.ContentType ampубym), 492	
boost_added (aiogram.types.message.Message ampубym), 55	

[illegible]

33		CallbackQuery (клас в ai-
BusinessOpeningHours (клас в ai-	ogram.types.callback_query), 34	
ogram.types.business_opening_hours),	CallbackQueryHandler (клас в ai-	
33	ogram.handlers.callback_query), 578	
BusinessOpeningHoursInterval (клас в ai-	can_add_web_page_previews (ai-	
ogram.types.business_opening_hours_interval),	ogram.types.chat_member_restricted.ChatMemberRestricted	
34	ampubym), 110	
button(aiogram.methods.answer_inline_query.AnswerInlineQuery	can_add_web_page_previews (ai-	
ampubym), 467	ogram.types.chat_permissions.ChatPermissions	
button_text(aiogram.types.web_app_data.WebAppData	ampubym), 131	
ampubym), 238	can_be_edited(aiogram.types.chat_member_administrator.ChatMember	
buttons(aiogram.utils.keyboard.InlineKeyboardBuilder	ampubym), 105	
property), 584	can_change_info (ai-	
buttons(aiogram.utils.keyboard.ReplyKeyboardBuilder	ogram.methods.promote_chat_member.PromoteChatMember	
property), 585	ampubym), 382	
BYN(aiogram.enums.currency.Currency ampubym),	can_change_info (ai-	
493	ogram.types.chat_administrator_rights.ChatAdministratorR	
	ampubym), 53	
C	can_change_info (ai-	
cache_time(aiogram.methods.answer_callback_query.AnswerCallbackQuery	ogram.types.chat_member_administrator.ChatMemberAdmi	
ampubym), 334	ampubym), 106	
cache_time(aiogram.methods.answer_inline_query.AnswerInlineQuery	can_change_info (ai-	
ampubym), 466	ogram.types.chat_member_restricted.ChatMemberRestricted	
cache_time(aiogram.utils.callback_answer.CallbackAnswer	ampubym), 110	
property), 601	can_change_info (ai-	
CAD(aiogram.enums.currency.Currency ampubym),	ogram.types.chat_permissions.ChatPermissions	
493	ampubym), 131	
callback_data(aiogram.handlers.callback_query.CallbackQueryHandler	can_change_info (ai-	
property), 578	ogram.types.user.User ampubym), 231	
callback_data(aiogram.types.inline_keyboard_button.InlineKeyboardButton	can_delete_messages (ai-	
ampubym), 144	ogram.methods.promote_chat_member.PromoteChatMember	
callback_game(aiogram.types.inline_keyboard_button.InlineKeyboardButton	ampubym), 381	
ampubym), 145	can_delete_messages (ai-	
CALLBACK_QUERY (ai-	ogram.types.chat_administrator_rights.ChatAdministratorR	
ogram.enums.update_type.UpdateType	ampubym), 53	
ampubym), 503	can_delete_messages (ai-	
callback_query(aiogram.types.update.Update	ogram.types.chat_member_administrator.ChatMemberAdmi	
ampubym), 311	ampubym), 106	
callback_query_id (ai-	can_delete_stories (ai-	
ogram.methods.answer_callback_query.AnswerCallbackQuery	ogram.methods.promote_chat_member.PromoteChatMember	
ampubym), 334	ampubym), 382	
callback_query_without_state (ai-	can_delete_stories (ai-	
ogram.fsm.scene.SceneConfig ampubym),	ogram.types.chat_administrator_rights.ChatAdministratorR	
566	ampubym), 53	
CallbackAnswer (клас в ai-	can_delete_stories (ai-	
ogram.utils.callback_answer), 600	ogram.types.chat_member_administrator.ChatMemberAdmi	
CallbackAnswerException, 574	ampubym), 106	
CallbackAnswerMiddleware (клас в ai-	can_edit_messages (ai-	
ogram.utils.callback_answer), 600	ogram.methods.promote_chat_member.PromoteChatMember	
CallbackData (клас в ai-	ampubym), 382	
ogram.filters.callback_data), 525	can_edit_messages (ai-	
CallbackGame (клас в ai-	ogram.types.chat_administrator_rights.ChatAdministratorR	
ogram.types.callback_game), 313	ampubym), 53	
	can_edit_messages (ai-	

<code>ogram.methods.promote_chat_member.PromoteChatMember</code>	<code>ogram.types.chat_member_restricted.ChatMemberRestricted</code>
<code>ampubym), 381</code>	<code>ampubym), 109</code>
<code>can_restrict_members</code>	<code>(ai- can_send_videos</code>
<code>ogram.types.chat_administrator_rights.ChatAdministratorRights</code>	<code>ogram.types.chat_permissions.ChatPermissions</code>
<code>ampubym), 53</code>	<code>ampubym), 131</code>
<code>can_restrict_members</code>	<code>(ai- can_send_voice_notes</code>
<code>ogram.types.chat_member_administrator.ChatMemberAdministrator</code>	<code>ogram.types.chat_member_restricted.ChatMemberRestricted</code>
<code>ampubym), 106</code>	<code>ampubym), 110</code>
<code>can_send_after</code>	<code>(ai- can_send_voice_notes</code>
<code>ogram.utils.web_app.WebAppInitData</code>	<code>ogram.types.chat_permissions.ChatPermissions</code>
<code>ampubym), 596</code>	<code>ampubym), 131</code>
<code>can_send_audios</code>	<code>(ai- can_set_sticker_set</code>
<code>ogram.types.chat_member_restricted.ChatMemberRestricted</code>	<code>(aiogram.types.chat.Chat</code>
<code>ampubym), 109</code>	<code>ampubym), 38</code>
<code>can_send_audios</code>	<code>can_set_sticker_set</code>
<code>ogram.types.chat_permissions.ChatPermissions</code>	<code>(ai- ogram.types.chat_full_info.ChatFullInfo</code>
<code>ampubym), 131</code>	<code>ampubym), 62</code>
<code>can_send_documents</code>	<code>caption (aiogram.methods.copy_message.CopyMessage</code>
<code>ogram.types.chat_member_restricted.ChatMemberRestricted</code>	<code>ampubym), 342</code>
<code>ampubym), 109</code>	<code>ampubym), 454</code>
<code>can_send_documents</code>	<code>(ai- caption (aiogram.methods.send_animation.SendAnimation</code>
<code>ogram.types.chat_permissions.ChatPermissions</code>	<code>ampubym), 389</code>
<code>ampubym), 131</code>	<code>caption (aiogram.methods.send_audio.SendAudio</code>
<code>can_send_messages</code>	<code>(ai- ampubym), 392</code>
<code>ogram.types.chat_member_restricted.ChatMemberRestricted</code>	<code>ampubym), 401</code>
<code>ampubym), 109</code>	<code>caption (aiogram.methods.send_document.SendDocument</code>
<code>can_send_messages</code>	<code>(ai- caption (aiogram.methods.send_photo.SendPhoto</code>
<code>ogram.types.chat_permissions.ChatPermissions</code>	<code>ampubym), 411</code>
<code>ampubym), 131</code>	<code>caption (aiogram.methods.send_video.SendVideo</code>
<code>can_send_other_messages</code>	<code>(ai- ampubym), 421</code>
<code>ogram.types.chat_member_restricted.ChatMemberRestricted</code>	<code>ampubym), 426</code>
<code>ampubym), 110</code>	<code>caption (aiogram.methods.send_voice.SendVoice</code>
<code>can_send_other_messages</code>	<code>(ai- caption (aiogram.types.inline_query_result_audio.InlineQueryResultAudio</code>
<code>ogram.types.chat_permissions.ChatPermissions</code>	<code>ampubym), 245</code>
<code>ampubym), 131</code>	<code>caption (aiogram.types.inline_query_result_cached_audio.InlineQueryResultCachedAudio</code>
<code>can_send_photos</code>	<code>(ai- ampubym), 248</code>
<code>ogram.types.chat_member_restricted.ChatMemberRestricted</code>	<code>ampubym), 250</code>
<code>ampubym), 109</code>	<code>caption (aiogram.types.inline_query_result_cached_document.InlineQueryResultCachedDocument</code>
<code>can_send_photos</code>	<code>(ai- caption (aiogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif</code>
<code>ogram.types.chat_permissions.ChatPermissions</code>	<code>ampubym), 252</code>
<code>ampubym), 131</code>	<code>caption (aiogram.types.inline_query_result_cached_mpeg4_gif.InlineQueryResultCachedMpeg4Gif</code>
<code>can_send_polls</code>	<code>(ai- ampubym), 254</code>
<code>ogram.types.chat_member_restricted.ChatMemberRestricted</code>	<code>ampubym), 256</code>
<code>ampubym), 110</code>	<code>caption (aiogram.types.inline_query_result_cached_photo.InlineQueryResultCachedPhoto</code>
<code>can_send_polls</code>	<code>(ai- caption (aiogram.types.inline_query_result_cached_video.InlineQueryResultCachedVideo</code>
<code>ogram.types.chat_permissions.ChatPermissions</code>	<code>ampubym), 260</code>
<code>ampubym), 131</code>	<code>caption (aiogram.types.inline_query_result_cached_voice.InlineQueryResultCachedVoice</code>
<code>can_send_video_notes</code>	<code>(ai- ampubym), 262</code>
<code>ogram.types.chat_member_restricted.ChatMemberRestricted</code>	<code>ampubym), 266</code>
<code>ampubym), 109</code>	<code>caption (aiogram.types.inline_query_result_document.InlineQueryResultDocument</code>
<code>can_send_video_notes</code>	<code>(ai- caption (aiogram.types.inline_query_result_gif.InlineQueryResultGif</code>
<code>ogram.types.chat_permissions.ChatPermissions</code>	<code>ampubym), 269</code>
<code>ampubym), 131</code>	<code>caption (aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif</code>
<code>can_send_videos</code>	<code>(ai- ampubym), 273</code>

<code>caption</code> (<code>aiogram.types.inline_query_result_photo.InlineQueryResultPhoto</code>), 275	<code>caption_entities</code> (<code>aiogram.types.inline_query_result_cached_mpeg4_gif.InlineQueryResultCachedMpeg4Gif</code>), 254
<code>caption</code> (<code>aiogram.types.inline_query_result_video.InlineQueryResultVideo</code>), 279	<code>caption_entities</code> (<code>aiogram.types.inline_query_result_cached_photo.InlineQueryResultCachedPhoto</code>), 256
<code>caption</code> (<code>aiogram.types.inline_query_result_voice.InlineQueryResultVoice</code>), 280	<code>caption_entities</code> (<code>aiogram.types.inline_query_result_cached_video.InlineQueryResultCachedVideo</code>), 260
<code>caption</code> (<code>aiogram.types.input_media_animation.InputMediaAnimation</code>), 147	<code>caption_entities</code> (<code>aiogram.types.inline_query_result_cached_voice.InlineQueryResultCachedVoice</code>), 262
<code>caption</code> (<code>aiogram.types.input_media_audio.InputMediaAudio</code>), 149	<code>caption_entities</code> (<code>aiogram.types.inline_query_result_document.InlineQueryResultDocument</code>), 266
<code>caption</code> (<code>aiogram.types.input_media_document.InputMediaDocument</code>), 150	<code>caption_entities</code> (<code>aiogram.types.message.Message</code>), 167
<code>caption</code> (<code>aiogram.types.input_media_photo.InputMediaPhoto</code>), 151	<code>copy_message</code> (<code>aiogram.methods.copy_message.CopyMessage</code>), 343
<code>caption</code> (<code>aiogram.types.input_media_video.InputMediaVideo</code>), 152	<code>edit_message_caption</code> (<code>aiogram.methods.edit_message_caption.EditMessageCaption</code>), 455
<code>caption_entities</code> (<code>aiogram.types.message.Message</code>), 167	<code>send_animation</code> (<code>aiogram.methods.send_animation.SendAnimation</code>), 389
<code>caption_entities</code> (<code>aiogram.methods.copy_message.CopyMessage</code>), 343	<code>send_audio</code> (<code>aiogram.methods.send_audio.SendAudio</code>), 392
<code>caption_entities</code> (<code>aiogram.methods.edit_message_caption.EditMessageCaption</code>), 455	<code>send_document</code> (<code>aiogram.methods.send_document.SendDocument</code>), 401
<code>caption_entities</code> (<code>aiogram.methods.send_animation.SendAnimation</code>), 389	<code>send_photo</code> (<code>aiogram.methods.send_photo.SendPhoto</code>), 411
<code>caption_entities</code> (<code>aiogram.methods.send_audio.SendAudio</code>), 392	<code>send_video</code> (<code>aiogram.methods.send_video.SendVideo</code>), 421
<code>caption_entities</code> (<code>aiogram.methods.send_document.SendDocument</code>), 401	<code>send_voice</code> (<code>aiogram.methods.send_voice.SendVoice</code>), 426
<code>caption_entities</code> (<code>aiogram.methods.send_photo.SendPhoto</code>), 411	<code>types</code> (<code>aiogram.types</code>), 246
<code>caption_entities</code> (<code>aiogram.methods.send_video.SendVideo</code>), 421	<code>types</code> (<code>aiogram.types</code>), 248
<code>caption_entities</code> (<code>aiogram.methods.send_voice.SendVoice</code>), 426	<code>types</code> (<code>aiogram.types</code>), 250
<code>caption_entities</code> (<code>aiogram.types.inline_query_result_audio.InlineQueryResultAudio</code>), 246	<code>types</code> (<code>aiogram.types</code>), 252
<code>caption_entities</code> (<code>aiogram.types.inline_query_result_cached_audio.InlineQueryResultCachedAudio</code>), 248	<code>types</code> (<code>aiogram.types</code>), 254
<code>caption_entities</code> (<code>aiogram.types.inline_query_result_cached_document.InlineQueryResultCachedDocument</code>), 250	<code>types</code> (<code>aiogram.types</code>), 256
<code>caption_entities</code> (<code>aiogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif</code>), 252	<code>types</code> (<code>aiogram.types</code>), 258

CHANNEL (<i>aiogram.enums.message_origin_type.MessageOriginType</i> <i>ampubym</i>), 503	
<i>ampubym</i>), 500	chat_boost (<i>aiogram.types.update.Update</i> <i>ampubym</i>), 312
CHANNEL_CHAT_CREATED (<i>aiogram.enums.content_type.ContentType</i> <i>ampubym</i>), 492	chat_has_username (<i>aiogram.types.keyboard_button_request_chat.KeyboardButtonRequestChat</i> <i>ampubym</i>), 155
channel_chat_created (<i>aiogram.types.message.Message</i> <i>ampubym</i>), 168	chat_id (<i>aiogram.methods.approve_chat_join_request.ApproveChatJoinRequest</i> <i>ampubym</i>), 335
CHANNEL_POST (<i>aiogram.enums.update_type.UpdateType</i> <i>ampubym</i>), 502	chat_id (<i>aiogram.methods.ban_chat_member.BanChatMember</i> <i>ampubym</i>), 336
channel_post (<i>aiogram.types.update.Update</i> <i>ampubym</i>), 310	chat_id (<i>aiogram.methods.ban_chat_sender_chat.BanChatSenderChat</i> <i>ampubym</i>), 338
CHAT (<i>aiogram.enums.bot_command_scope_type.BotCommandScopeType</i> <i>ampubym</i>), 489	chat_id (<i>aiogram.methods.close_forum_topic.CloseForumTopic</i> <i>ampubym</i>), 340
CHAT (<i>aiogram.enums.message_origin_type.MessageOriginType</i> <i>ampubym</i>), 500	chat_id (<i>aiogram.methods.close_general_forum_topic.CloseGeneralForumTopic</i> <i>ampubym</i>), 341
chat (<i>aiogram.types.business_messages_deleted.BusinessMessagesDeleted</i> <i>ampubym</i>), 33	chat_id (<i>aiogram.methods.copy_message.CopyMessage</i> <i>ampubym</i>), 342
chat (<i>aiogram.types.chat_boost_removed.ChatBoostRemoved</i> <i>ampubym</i>), 55	chat_id (<i>aiogram.methods.copy_messages.CopyMessages</i> <i>ampubym</i>), 344
chat (<i>aiogram.types.chat_boost_updated.ChatBoostUpdated</i> <i>ampubym</i>), 58	chat_id (<i>aiogram.methods.create_chat_invite_link.CreateChatInviteLink</i> <i>ampubym</i>), 346
chat (<i>aiogram.types.chat_join_request.ChatJoinRequest</i> <i>ampubym</i>), 63	chat_id (<i>aiogram.methods.create_forum_topic.CreateForumTopic</i> <i>ampubym</i>), 347
chat (<i>aiogram.types.chat_member_updated.ChatMemberUpdated</i> <i>ampubym</i>), 111	chat_id (<i>aiogram.methods.decline_chat_join_request.DeclineChatJoinRequest</i> <i>ampubym</i>), 348
chat (<i>aiogram.types.external_reply_info.ExternalReplyInfo</i> <i>ampubym</i>), 135	chat_id (<i>aiogram.methods.delete_chat_photo.DeleteChatPhoto</i> <i>ampubym</i>), 349
chat (<i>aiogram.types.giveaway_winners.GiveawayWinners</i> <i>ampubym</i>), 143	chat_id (<i>aiogram.methods.delete_chat_sticker_set.DeleteChatStickerSet</i> <i>ampubym</i>), 350
chat (<i>aiogram.types.inaccessible_message.InaccessibleMessage</i> <i>ampubym</i>), 143	chat_id (<i>aiogram.methods.delete_forum_topic.DeleteForumTopic</i> <i>ampubym</i>), 351
chat (<i>aiogram.types.message.Message</i> <i>ampubym</i>), 165	chat_id (<i>aiogram.methods.delete_message.DeleteMessage</i> <i>ampubym</i>), 452
chat (<i>aiogram.types.message_origin_channel.MessageOriginChannel</i> <i>ampubym</i>), 216	chat_id (<i>aiogram.methods.delete_messages.DeleteMessages</i> <i>ampubym</i>), 453
chat (<i>aiogram.types.message_reaction_count_updated.MessageReactionCountUpdated</i> <i>ampubym</i>), 219	chat_id (<i>aiogram.methods.edit_chat_invite_link.EditChatInviteLink</i> <i>ampubym</i>), 354
chat (<i>aiogram.types.message_reaction_updated.MessageReactionUpdated</i> <i>ampubym</i>), 219	chat_id (<i>aiogram.methods.edit_forum_topic.EditForumTopic</i> <i>ampubym</i>), 355
chat (<i>aiogram.types.story.Story</i> <i>ampubym</i>), 229	chat_id (<i>aiogram.methods.edit_general_forum_topic.EditGeneralForumTopic</i> <i>ampubym</i>), 356
chat (<i>aiogram.utils.web_app.WebAppInitData</i> <i>ampubym</i>), 595	chat_id (<i>aiogram.methods.edit_message_caption.EditMessageCaption</i> <i>ampubym</i>), 454
Chat (клас в <i>aiogram.types.chat</i>), 36	chat_id (<i>aiogram.methods.edit_message_live_location.EditMessageLiveLocation</i> <i>ampubym</i>), 456
CHAT_ADMINISTRATORS (<i>aiogram.enums.bot_command_scope_type.BotCommandScopeType</i> <i>ampubym</i>), 489	chat_id (<i>aiogram.methods.edit_message_media.EditMessageMedia</i> <i>ampubym</i>), 458
CHAT_BACKGROUND_SET (<i>aiogram.enums.content_type.ContentType</i> <i>ampubym</i>), 492	chat_id (<i>aiogram.methods.edit_message_reply_markup.EditMessageReplyMarkup</i> <i>ampubym</i>), 459
chat_background_set (<i>aiogram.types.message.Message</i> <i>ampubym</i>), 169	chat_id (<i>aiogram.methods.edit_message_text.EditMessageText</i> <i>ampubym</i>), 461
CHAT_BOOST (<i>aiogram.enums.update_type.UpdateType</i> <i>ampubym</i>), 312	chat_id (<i>aiogram.methods.export_chat_invite_link.ExportChatInviteLink</i> <i>ampubym</i>), 357

<code>chat_id (aiogram.methods.forward_message.ForwardMessage, 359</code>	<code>chat_id (aiogram.methods.send_message.SendMessage, 408</code>
<code>chat_id (aiogram.methods.forward_messages.ForwardMessages, 360</code>	<code>chat_id (aiogram.methods.send_photo.SendPhoto, 411</code>
<code>chat_id (aiogram.methods.get_chat.GetChat, 362</code>	<code>chat_id (aiogram.methods.send_poll.SendPoll, 414</code>
<code>chat_id (aiogram.methods.get_chat_administrators.GetChatAdministrators, 363</code>	<code>chat_id (aiogram.methods.send_sticker.SendSticker, 322</code>
<code>chat_id (aiogram.methods.get_chat_member.GetChatMember, 364</code>	<code>chat_id (aiogram.methods.send_venue.SendVenue, 417</code>
<code>chat_id (aiogram.methods.get_chat_member_count.GetChatMemberCount, 365</code>	<code>chat_id (aiogram.methods.send_video.SendVideo, 420</code>
<code>chat_id (aiogram.methods.get_chat_menu_button.GetChatMenuButton, 366</code>	<code>chat_id (aiogram.methods.send_video_note.SendVideoNote, 423</code>
<code>chat_id (aiogram.methods.get_game_high_scores.GetGameHighScores, 470</code>	<code>chat_id (aiogram.methods.send_voice.SendVoice, 426</code>
<code>chat_id (aiogram.methods.get_user_chat_boosts.GetUserChatBoosts, 374</code>	<code>chat_id (aiogram.methods.set_chat_administrator_custom_title.SetChatAdministratorCustomTitle, 428</code>
<code>chat_id (aiogram.methods.hide_general_forum_topic.HideGeneralForumTopic, 376</code>	<code>chat_id (aiogram.methods.set_chat_description.SetChatDescription, 429</code>
<code>chat_id (aiogram.methods.leave_chat.LeaveChat, 377</code>	<code>chat_id (aiogram.methods.set_chat_menu_button.SetChatMenuButton, 431</code>
<code>chat_id (aiogram.methods.pin_chat_message.PinChatMessage, 379</code>	<code>chat_id (aiogram.methods.set_chat_permissions.SetChatPermissions, 432</code>
<code>chat_id (aiogram.methods.promote_chat_member.PromoteChatMember, 381</code>	<code>chat_id (aiogram.methods.set_chat_photo.SetChatPhoto, 433</code>
<code>chat_id (aiogram.methods.reopen_forum_topic.ReopenForumTopic, 383</code>	<code>chat_id (aiogram.methods.set_chat_sticker_set.SetChatStickerSet, 434</code>
<code>chat_id (aiogram.methods.reopen_general_forum_topic.ReopenGeneralForumTopic, 384</code>	<code>chat_id (aiogram.methods.set_chat_title.SetChatTitle, 435</code>
<code>chat_id (aiogram.methods.restrict_chat_member.RestrictChatMember, 385</code>	<code>chat_id (aiogram.methods.set_game_score.SetGameScore, 473</code>
<code>chat_id (aiogram.methods.revoke_chat_invite_link.RevokeChatInviteLink, 387</code>	<code>chat_id (aiogram.methods.set_message_reaction.SetMessageReaction, 436</code>
<code>chat_id (aiogram.methods.send_animation.SendAnimation, 388</code>	<code>chat_id (aiogram.methods.stop_message_live_location.StopMessageLiveLocation, 463</code>
<code>chat_id (aiogram.methods.send_audio.SendAudio, 391</code>	<code>chat_id (aiogram.methods.stop_poll.StopPoll, 464</code>
<code>chat_id (aiogram.methods.send_chat_action.SendChatAction, 394</code>	<code>chat_id (aiogram.methods.unban_chat_member.UnbanChatMember, 443</code>
<code>chat_id (aiogram.methods.send_contact.SendContact, 395</code>	<code>chat_id (aiogram.methods.unban_chat_sender_chat.UnbanChatSenderChat, 445</code>
<code>chat_id (aiogram.methods.send_dice.SendDice, 398</code>	<code>chat_id (aiogram.methods.unhide_general_forum_topic.UnhideGeneralForumTopic, 446</code>
<code>chat_id (aiogram.methods.send_document.SendDocument, 400</code>	<code>chat_id (aiogram.methods.unpin_all_chat_messages.UnpinAllChatMessages, 447</code>
<code>chat_id (aiogram.methods.send_game.SendGame, 471</code>	<code>chat_id (aiogram.methods.unpin_all_forum_topic_messages.UnpinAllForumTopicMessages, 448</code>
<code>chat_id (aiogram.methods.send_invoice.SendInvoice, 479</code>	<code>chat_id (aiogram.methods.unpin_all_general_forum_topic_messages.UnpinAllGeneralForumTopicMessages, 449</code>
<code>chat_id (aiogram.methods.send_location.SendLocation, 403</code>	<code>chat_id (aiogram.methods.unpin_chat_message.UnpinChatMessage, 450</code>
<code>chat_id (aiogram.methods.send_media_group.SendMediaGroup, 405</code>	<code>chat_id (aiogram.types.bot_command_scope_chat.BotCommandScopeChat, 28</code>

<code>chat_id</code> (<code>aiogram.types.bot_command_scope_chat_administrator.BotCommandScopeChatAdministrator</code>), 29	<code>ChatBoostSource</code> (клас в <code>aiogram.types.chat_boost_source</code>), 56
<code>chat_id</code> (<code>aiogram.types.bot_command_scope_chat_member.BotCommandScopeChatMember</code>), 29	<code>ChatBoostSourceGiftCode</code> (клас в <code>aiogram.types.chat_boost_source_gift_code</code>), 56
<code>chat_id</code> (<code>aiogram.types.chat_shared.ChatShared</code>), 133	<code>ChatBoostSourceGiveaway</code> (клас в <code>aiogram.types.chat_boost_source_giveaway</code>), 56
<code>chat_id</code> (<code>aiogram.types.reply_parameters.ReplyParameters</code>), 227	<code>ChatBoostSourcePremium</code> (клас в <code>aiogram.types.chat_boost_source_premium</code>), 57
<code>chat_instance</code> (<code>aiogram.types.callback_query.CallbackQuery</code>), 34	<code>ChatBoostSourceType</code> (клас в <code>aiogram.types.chat_boost_source_type</code>), 490
<code>chat_instance</code> (<code>aiogram.utils.web_app.WebAppInitData</code>), 596	<code>ChatBoostUpdated</code> (клас в <code>aiogram.types.chat_boost_updated</code>), 58
<code>chat_is_channel</code> (<code>aiogram.types.keyboard_button_request_chat.KeyboardButtonRequestChat</code>), 155	<code>ChatFullInfo</code> (клас в <code>aiogram.types.chat_full_info</code>), 58
<code>chat_is_created</code> (<code>aiogram.types.keyboard_button_request_chat.KeyboardButtonRequestChat</code>), 156	<code>ChatInviteLink</code> (клас в <code>aiogram.types.chat_invite_link</code>), 62
<code>chat_is_forum</code> (<code>aiogram.types.keyboard_button_request_chat.KeyboardButtonRequestChat</code>), 155	<code>ChatJoinRequest</code> (клас в <code>aiogram.types.chat_join_request</code>), 63
<code>CHAT_JOIN_REQUEST</code> (<code>aiogram.enums.update_type.UpdateType</code>), 503	<code>ChatLocation</code> (клас в <code>aiogram.types.chat_location</code>), 103
<code>chat_join_request</code> (<code>aiogram.types.update.Update</code>), 311	<code>ChatMember</code> (клас в <code>aiogram.types.chat_member</code>), 104
<code>CHAT_MEMBER</code> (<code>aiogram.enums.bot_command_scope_type.BotCommandScopeType</code>), 489	<code>ChatMemberAdministrator</code> (клас в <code>aiogram.types.chat_member_administrator</code>), 105
<code>CHAT_MEMBER</code> (<code>aiogram.enums.update_type.UpdateType</code>), 503	<code>ChatMemberBanned</code> (клас в <code>aiogram.types.chat_member_banned</code>), 107
<code>chat_member</code> (<code>aiogram.types.update.Update</code>), 311	<code>ChatMemberLeft</code> (клас в <code>aiogram.types.chat_member_left</code>), 107
<code>CHAT_SHARED</code> (<code>aiogram.enums.content_type.ContentType</code>), 492	<code>ChatMemberMember</code> (клас в <code>aiogram.types.chat_member_member</code>), 108
<code>chat_shared</code> (<code>aiogram.types.message.Message</code>), 168	<code>ChatMemberOwner</code> (клас в <code>aiogram.types.chat_member_owner</code>), 108
<code>chat_type</code> (<code>aiogram.types.inline_query.InlineQuery</code>), 241	<code>ChatMemberRestricted</code> (клас в <code>aiogram.types.chat_member_restricted</code>), 109
<code>chat_type</code> (<code>aiogram.utils.web_app.WebAppInitData</code>), 596	<code>ChatMemberStatus</code> (клас в <code>aiogram.enums.chat_member_status</code>), 490
<code>ChatAction</code> (клас в <code>aiogram.enums.chat_action</code>), 489	<code>ChatMemberUpdated</code> (клас в <code>aiogram.types.chat_member_updated</code>), 110
<code>ChatActionMiddleware</code> (клас в <code>aiogram.utils.chat_action</code>), 593	<code>ChatMemberUpdatedFilter</code> (клас в <code>aiogram.filters.chat_member_updated</code>), 520
<code>ChatActionSender</code> (клас в <code>aiogram.utils.chat_action</code>), 591	<code>ChatPermissions</code> (клас в <code>aiogram.types.chat_permissions</code>), 130
<code>ChatAdministratorRights</code> (клас в <code>aiogram.types.chat_administrator_rights</code>), 52	<code>ChatPhoto</code> (клас в <code>aiogram.types.chat_photo</code>), 132
<code>ChatBackground</code> (клас в <code>aiogram.types.chat_background</code>), 54	
<code>ChatBoost</code> (клас в <code>aiogram.types.chat_boost</code>), 54	
<code>ChatBoostAdded</code> (клас в <code>aiogram.types.chat_boost_added</code>), 55	
<code>ChatBoostRemoved</code> (клас в <code>aiogram.types.chat_boost_removed</code>), 55	

chats (*aiogram.types.giveaway.Giveaway* *ампубум*), 141
ChatShared (клас в *aiogram.types.chat_shared*), 132
ChatType (клас в *aiogram.enums.chat_type*), 491
check_flags() (в модулі *aiogram.dispatcher.flags*), 576
check_response() (*aiogram.client.session.base.BaseSession* *метод*), 15
check_webapp_signature() (в модулі *aiogram.utils.web_app*), 594
CHF (*aiogram.enums.currency.Currency* *ампубум*), 493
CHIN (*aiogram.enums.mask_position_point.MaskPositionPoint* *ампубум*), 498
CHOOSE_STICKER (*aiogram.enums.chat_action.ChatAction* *ампубум*), 490
choose_sticker() (*aiogram.utils.chat_action.ChatActionSender* *class method*), 592
CHOSEN_INLINE_RESULT (*aiogram.enums.update_type.UpdateType* *ампубум*), 503
chosen_inline_result (*aiogram.types.update.Update* *ампубум*), 311
ChosenInlineResult (клас в *aiogram.types.chosen_inline_result*), 240
city (*aiogram.types.shipping_address.ShippingAddress* *ампубум*), 307
clear_data() (*aiogram.fsm.scene.SceneWizard* *метод*), 567
ClientDecodeError, 575
Close (клас в *aiogram.methods.close*), 339
close() (*aiogram.client.session.base.BaseSession* *метод*), 15
close() (*aiogram.fsm.scene.ScenesManager* *метод*), 566
close() (*aiogram.fsm.storage.base.BaseStorage* *метод*), 550
close() (*aiogram.webhook.aihttp_server.SimpleRequestHandler* *метод*), 533
close_date (*aiogram.methods.send_poll.SendPoll* *ампубум*), 415
close_date (*aiogram.types.poll.Poll* *ампубум*), 222
CloseForumTopic (клас в *aiogram.methods.close_forum_topic*), 340
CloseGeneralForumTopic (клас в *aiogram.methods.close_general_forum_topic*), 341
closing_minute (*aiogram.types.business_opening_hours_interval.BusinessOpeningHoursInterval* *ампубум*), 34
CLP (*aiogram.enums.currency.Currency* *ампубум*), 494
CNY (*aiogram.enums.currency.Currency* *ампубум*), 494
CODE (*aiogram.enums.message_entity_type.MessageEntityType* *ампубум*), 499
Code (клас в *aiogram.utils.formatting*), 607
color (*aiogram.types.background_fill_solid.BackgroundFillSolid* *ампубум*), 22
colors (*aiogram.types.background_fill_freeform_gradient.BackgroundFillFreeformGradient* *ампубум*), 22
command (*aiogram.filters.command.CommandObject* *ампубум*), 519
Command (*aiogram.types.bot_command.BotCommand* *ампубум*), 26
Command (клас в *aiogram.filters.command*), 518
CommandObject (клас в *aiogram.filters.command*), 519
COMMANDS (*aiogram.enums.menu_button_type.MenuButtonType* *ампубум*), 499
commands (*aiogram.methods.set_my_commands.SetMyCommands* *ампубум*), 438
CONNECTED_WEBSITE (*aiogram.enums.content_type.ContentType* *ампубум*), 492
connected_website (*aiogram.types.message.Message* *ампубум*), 168
Const118nMiddleware (клас в *aiogram.utils.i18n.middleware*), 588
CONTACT (*aiogram.enums.content_type.ContentType* *ампубум*), 491
CONTACT (*aiogram.enums.inline_query_result_type.InlineQueryResultType* *ампубум*), 497
contact (*aiogram.types.external_reply_info.ExternalReplyInfo* *ампубум*), 136
contact (*aiogram.types.message.Message* *ампубум*), 167
Contact (клас в *aiogram.types.contact*), 133
content_type (*aiogram.types.message.Message* *property*), 170
Content (клас в *aiogram.enums.content_type*), 491
COP (*aiogram.enums.currency.Currency* *ампубум*), 494
copy() (*aiogram.utils.keyboard.InlineKeyboardBuilder* *метод*), 584
copy() (*aiogram.utils.keyboard.ReplyKeyboardBuilder* *метод*), 585
copy_to() (*aiogram.types.message.Message* *метод*), 207
CopyMessage (клас в *aiogram.types.message*), 342
CopyMessages (клас в *aiogram.types.messages*), 342

ogram.methods.copy_messages), 344
 correct_option_id (aiogram.methods.send_poll.SendPoll ampu-
 bym), 415
 correct_option_id (aiogram.types.poll.Poll ampu-
 bym), 221
 country_code (aiogram.types.shipping_address.ShippingAddress
 ampuбym), 307
 country_codes (aiogram.types.giveaway.Giveaway ampuбym), 141
 CRC (aiogram.enums.currency.Currency ampuбym), 494
 create_invite_link() (aiogram.types.chat.Chat ampuбym), 43
 create_start_link() (в модулі aiogram.utils.deep_linking), 612
 CreateChatInviteLink (клас в aiogram.methods.create_chat_invite_link), 346
 CreateForumTopic (клас в aiogram.methods.create_forum_topic), 347
 CreateInvoiceLink (клас в aiogram.methods.create_invoice_link), 476
 CreateNewStickerSet (клас в aiogram.methods.create_new_sticker_set), 316
 creates_join_request (aiogram.methods.create_chat_invite_link.CreateChatInviteLink ampuбym), 346
 creates_join_request (aiogram.methods.edit_chat_invite_link.EditChatInviteLink ampuбym), 354
 creates_join_request (aiogram.types.chat_invite_link.ChatInviteLink ampuбym), 62
 CREATOR (aiogram.enums.chat_member_status.ChatMemberStatus ampuбym), 490
 creator (aiogram.types.chat_invite_link.ChatInviteLink ampuбym), 62
 credentials (aiogram.types.passport_data.PassportData ampuбym), 295
 currency (aiogram.methods.create_invoice_link.CreateInvoiceLink ampuбym), 477
 currency (aiogram.methods.send_invoice.SendInvoice ampuбym), 480
 currency (aiogram.types.input_invoice_message_content.InputInvoiceMessageContent ampuбym), 284
 currency (aiogram.types.invoice.Invoice ampuбym), 304
 currency (aiogram.types.pre_checkout_query.PreCheckoutQuery ampuбym), 305
 currency (aiogram.types.successful_payment.SuccessfulPayment ampuбym), 309
 Currency (клас в aiogram.enums.currency), 493
 CUSTOM_EMOJI (aiogram.enums.message_entity_type.MessageEntityType ampuбym), 499
 CUSTOM_EMOJI (aiogram.enums.reaction_type_type.ReactionTypeType ampuбym), 501
 CUSTOM_EMOJI (aiogram.enums.sticker_type_type.StickerTypeType ampuбym), 502
 custom_emoji_id (aiogram.methods.set_custom_emoji_sticker_set_thumbnail.SetCustomEmojiStickerSetThumbnail ampuбym), 325
 custom_emoji_id (aiogram.types.message_entity.MessageEntityType ampuбym), 215
 custom_emoji_id (aiogram.types.reaction_type_custom_emoji.ReactionTypeCustomEmoji ampuбym), 224
 custom_emoji_id (aiogram.types.sticker.Sticker ampuбym), 291
 custom_emoji_ids (aiogram.methods.get_custom_emoji_stickers.GetCustomEmojiStickers ampuбym), 319
 custom_emoji_sticker_set_name (aiogram.types.chat.Chat ampuбym), 38
 custom_emoji_sticker_set_name (aiogram.types.chat_full_info.ChatFullInfo ampuбym), 62
 custom_title (aiogram.methods.set_chat_administrator_custom_title.SetChatAdministratorCustomTitle ampuбym), 428
 custom_title (aiogram.types.chat_member_administrator.ChatMemberAdministrator ampuбym), 106
 custom_title (aiogram.types.chat_member_owner.ChatMemberOwner ampuбym), 108
 CustomEmoji (клас в aiogram.utils.formatting), 607
 CZK (aiogram.enums.currency.Currency ampuбym), 494
 D
 dark_theme_dimming (aiogram.types.background_type_fill.BackgroundTypeFill ampuбym), 24
 dark_theme_dimming (aiogram.types.background_type_wallpaper.BackgroundTypeWallpaper ampuбym), 25
 DART (aiogram.enums.dice_emoji.DiceEmoji ampuбym), 496
 DART (aiogram.types.dice.DiceEmoji ampuбym), 134
 DATA (aiogram.enums.message_type_type.MessageTypeType ampuбym), 500
 data (aiogram.types.callback_query.CallbackQuery ampuбym), 35
 data (aiogram.types.encrypted_credentials.EncryptedCredentials ampuбym), 293

data (*aiogram.types.encrypted_passport_element.EncryptedPassportElement* *ампубум*), 292
data (*aiogram.types.passport_data.PassportData* *ампубум*), 294
data (*aiogram.types.web_app_data.WebAppData* *ампубум*), 295
data_hash (*aiogram.types.passport_element_error_data_field.PassportElementErrorDataField* *ампубум*), 238
date (*aiogram.types.business_connection.BusinessConnection* *ампубум*), 32
date (*aiogram.types.chat_join_request.ChatJoinRequest* *ампубум*), 63
date (*aiogram.types.chat_member_updated.ChatMemberUpdated* *ампубум*), 111
date (*aiogram.types.inaccessible_message.InaccessibleMessage* *ампубум*), 144
date (*aiogram.types.message.Message* *ампубум*), 165
date (*aiogram.types.message_origin_channel.MessageOriginChannel* *ампубум*), 216
date (*aiogram.types.message_origin_chat.MessageOriginChat* *ампубум*), 217
date (*aiogram.types.message_origin_hidden_user.MessageOriginHiddenUser* *ампубум*), 218
date (*aiogram.types.message_origin_user.MessageOriginUser* *ампубум*), 218
date (*aiogram.types.message_reaction_count_updated.MessageReactionCountUpdated* *ампубум*), 219
date (*aiogram.types.message_reaction_updated.MessageReactionUpdated* *ампубум*), 220
day (*aiogram.types.birthdate.Birthdate* *ампубум*), 25
decline() (*aiogram.types.chat_join_request.ChatJoinRequest* *метод*), 64
DeclineChatJoinRequest (*клас в aiogram.methods.decline_chat_join_request*), 348
decode_payload() (*в модулі aiogram.utils.deep_linking*), 613
DEFAULT (*aiogram.enums.bot_command_scope_type.BotCommandScopeType* *ампубум*), 489
DEFAULT (*aiogram.enums.menu_button_type.MenuButtonType* *ампубум*), 499
DefaultKeyBuilder (*клас в aiogram.fsm.storage.base*), 549
delete() (*aiogram.types.message.Message* *метод*), 212
DELETE_CHAT_PHOTO (*aiogram.enums.content_type.ContentType* *ампубум*), 492
delete_chat_photo (*aiogram.types.message.Message* *ампубум*), 167
delete_from_set() (*aiogram.types.sticker.Sticker* *метод*), 42
delete_message() (*aiogram.types.chat.Chat* *метод*), 42
delete_photo() (*aiogram.types.chat.Chat* *метод*), 51
delete_reply_markup() (*aiogram.types.message.Message* *метод*), 210
delete_sticker_set() (*aiogram.types.chat.Chat* *метод*), 45
DeleteChatPhoto (*клас в aiogram.methods.delete_chat_photo*), 349
DeleteChatStickerSet (*клас в aiogram.methods.delete_chat_sticker_set*), 350
DELETED_BUSINESS_MESSAGES (*aiogram.enums.update_type.UpdateType* *ампубум*), 503
DeletedBusinessMessages (*aiogram.types.update.Update* *ампубум*), 311
DeleteForumTopic (*клас в aiogram.methods.delete_forum_topic*), 351
DeleteMessage (*клас в aiogram.methods.delete_message*), 451
DeleteMessagesCountUpdated (*клас в aiogram.methods.delete_messages*), 453
DeleteMyCommands (*клас в aiogram.methods.delete_my_commands*), 352
DeleteStickerFromSet (*клас в aiogram.methods.delete_sticker_from_set*), 317
DeleteStickerSet (*клас в aiogram.methods.delete_sticker_set*), 318
DeleteWebhook (*клас в aiogram.methods.delete_webhook*), 482
description (*aiogram.methods.create_invoice_link.CreateInvoiceLink* *метод*), 477
description (*aiogram.methods.send_invoice.SendInvoice* *метод*), 480
description (*aiogram.methods.set_chat_description.SetChatDescription* *ампубум*), 430
description (*aiogram.methods.set_my_description.SetMyDescription* *ампубум*), 440
description (*aiogram.types.bot_command.BotCommand* *ампубум*), 26
description (*aiogram.types.bot_description.BotDescription* *ампубум*), 30
description (*aiogram.types.chat.Chat* *ампубум*), 38
description (*aiogram.types.chat_full_info.ChatFullInfo* *ампубум*), 61

<code>description</code> (<code>aiogram.types.game.Game</code> <code>ampubym</code>), 313	<code>ogram.methods.forward_messages.ForwardMessages</code> <code>ampubym</code>), 360
<code>description</code> (<code>aiogram.types.inline_query_result_article</code> <code>ampubym</code>), 244	<code>DisableNotification</code> (<code>aiogram.methods.pin_chat_message.PinChatMessage</code> <code>ampubym</code>), 379
<code>description</code> (<code>aiogram.types.inline_query_result_cached_document</code> <code>ampubym</code>), 250	<code>InlineQueryResultCachedDocument</code> <code>disable_notification</code> (<code>aiogram.methods.send_animation.SendAnimation</code> <code>ampubym</code>), 389
<code>description</code> (<code>aiogram.types.inline_query_result_cached_photo</code> <code>ampubym</code>), 256	<code>InlineQueryResultCachedPhoto</code> <code>disable_notification</code> (<code>aiogram.methods.send_audio.SendAudio</code> <code>ampubym</code>), 398
<code>description</code> (<code>aiogram.types.inline_query_result_cached_video</code> <code>ampubym</code>), 260	<code>InlineQueryResultCachedVideo</code> <code>disable_notification</code> (<code>aiogram.methods.send_document.SendDocument</code> <code>ampubym</code>), 401
<code>description</code> (<code>aiogram.types.inline_query_result_document</code> <code>ampubym</code>), 266	<code>InlineQueryResultDocument</code> <code>disable_notification</code> (<code>aiogram.methods.send_game.SendGame</code> <code>ampubym</code>), 471
<code>description</code> (<code>aiogram.types.inline_query_result_photo</code> <code>ampubym</code>), 275	<code>InlineQueryResultPhoto</code> <code>send_contact.SendContact</code> <code>ampubym</code>), 396
<code>description</code> (<code>aiogram.types.inline_query_result_video</code> <code>ampubym</code>), 279	<code>InlineQueryResultVideo</code> <code>ogram.methods.send_dice.SendDice</code> <code>ampubym</code>), 398
<code>description</code> (<code>aiogram.types.input_invoice_message_content</code> <code>ampubym</code>), 284	<code>InputInvoiceMessageContent</code> <code>disable_notification</code> (<code>aiogram.methods.send_location.SendLocation</code> <code>ampubym</code>), 404
<code>description</code> (<code>aiogram.types.invoice.Invoice</code> <code>ampubym</code>), 304	<code>ogram.methods.send_document.SendDocument</code> <code>ampubym</code>), 401
<code>DetailedAiogramError</code> , 574	<code>disable_notification</code> (<code>aiogram.methods.send_media_group.SendMediaGroup</code> <code>ampubym</code>), 406
<code>DICE</code> (<code>aiogram.enums.content_type.ContentType</code> <code>ampubym</code>), 491	<code>ogram.methods.send_game.SendGame</code> <code>ampubym</code>), 471
<code>DICE</code> (<code>aiogram.enums.dice_emoji.DiceEmoji</code> <code>ampubym</code>), 496	<code>disable_notification</code> (<code>aiogram.methods.send_invoice.SendInvoice</code> <code>ampubym</code>), 481
<code>DICE</code> (<code>aiogram.types.dice.DiceEmoji</code> <code>ampubym</code>), 134	<code>ogram.methods.send_location.SendLocation</code> <code>ampubym</code>), 404
<code>dice</code> (<code>aiogram.types.external_reply_info.ExternalReplyInfo</code> <code>ampubym</code>), 136	<code>disable_notification</code> (<code>aiogram.methods.send_message.SendMessage</code> <code>ampubym</code>), 409
<code>dice</code> (<code>aiogram.types.message.Message</code> <code>ampubym</code>), 167	<code>ogram.methods.send_location.SendLocation</code> <code>ampubym</code>), 404
<code>Dice</code> (клас в <code>aiogram.types.dice</code>), 134	<code>disable_notification</code> (<code>aiogram.methods.send_media_group.SendMediaGroup</code> <code>ampubym</code>), 406
<code>DiceEmoji</code> (клас в <code>aiogram.enums.dice_emoji</code>), 496	<code>ogram.methods.send_message.SendMessage</code> <code>ampubym</code>), 409
<code>DiceEmoji</code> (клас в <code>aiogram.types.dice</code>), 134	<code>disable_notification</code> (<code>aiogram.methods.send_photo.SendPhoto</code> <code>ampubym</code>), 412
<code>disable()</code> (<code>aiogram.utils.callback_answer.CallbackAnswer</code> <code>метод</code>), 600	<code>ogram.methods.send_message.SendMessage</code> <code>ampubym</code>), 409
<code>disable_content_type_detection</code> (<code>aiogram.methods.send_document.SendDocument</code> <code>ampubym</code>), 401	<code>disable_notification</code> (<code>aiogram.methods.send_photo.SendPhoto</code> <code>ampubym</code>), 412
<code>disable_content_type_detection</code> (<code>aiogram.types.input_media_document.InputMediaDocument</code> <code>ampubym</code>), 150	<code>disable_notification</code> (<code>aiogram.methods.send_poll.SendPoll</code> <code>ampubym</code>), 415
<code>disable_edit_message</code> (<code>aiogram.methods.set_game_score.SetGameScore</code> <code>ampubym</code>), 473	<code>disable_notification</code> (<code>aiogram.methods.send_sticker.SendSticker</code> <code>ampubym</code>), 323
<code>disable_notification</code> (<code>aiogram.methods.copy_message.CopyMessage</code> <code>ampubym</code>), 343	<code>disable_notification</code> (<code>aiogram.methods.send_venue.SendVenue</code> <code>ampubym</code>), 418
<code>disable_notification</code> (<code>aiogram.methods.copy_messages.CopyMessages</code> <code>ampubym</code>), 345	<code>disable_notification</code> (<code>aiogram.methods.send_video.SendVideo</code> <code>ampubym</code>), 421
<code>disable_notification</code> (<code>aiogram.methods.forward_message.ForwardMessage</code> <code>ampubym</code>), 359	<code>disable_notification</code> (<code>aiogram.methods.send_video_note.SendVideoNote</code> <code>ampubym</code>), 424
<code>disable_notification</code> (<code>aiogram.methods.forward_messages.ForwardMessages</code> <code>ampubym</code>), 360	<code>disable_notification</code> (<code>aiogram.methods.send_video_note.SendVideoNote</code> <code>ampubym</code>), 424

<code>ogram.methods.send_voice.SendVoice</code> <code>ampu6ym</code>), 427	<code>ogram.methods.set_webhook.SetWebhook</code> <code>ampu6ym</code>), 486
<code>disable_web_page_preview</code> (<code>aiogram.methods.edit_message_text.EditMessageText</code> <code>ampu6ym</code>), 461	<code>duration</code> (<code>aiogram.methods.send_animation.SendAnimation</code> <code>ampu6ym</code>), 389
<code>disable_web_page_preview</code> (<code>aiogram.methods.send_message.SendMessage</code> <code>ampu6ym</code>), 409	<code>duration</code> (<code>aiogram.methods.send_audio.SendAudio</code> <code>ampu6ym</code>), 392
<code>disable_web_page_preview</code> (<code>aiogram.methods.send_message.SendMessage</code> <code>ampu6ym</code>), 409	<code>duration</code> (<code>aiogram.methods.send_video.SendVideo</code> <code>ampu6ym</code>), 420
<code>disable_web_page_preview</code> (<code>aiogram.types.input_text_message_content.InputTextMessageContent</code> <code>ampu6ym</code>), 287	<code>duration</code> (<code>aiogram.methods.send_video_note.SendVideoNote</code> <code>ampu6ym</code>), 423
<code>disabled</code> (<code>aiogram.utils.callback_answer.CallbackAnswer</code> <code>property</code>), 600	<code>duration</code> (<code>aiogram.methods.send_voice.SendVoice</code> <code>ampu6ym</code>), 427
<code>Dispatcher</code> (клас в <code>aiogram.dispatcher.dispatcher</code>), 513	<code>duration</code> (<code>aiogram.types.animation.Animation</code> <code>ampu6ym</code>), 19
<code>distance</code> (<code>aiogram.types.proximity_alert_triggered.ProximityAlertTriggered</code> <code>ampu6ym</code>), 223	<code>duration</code> (<code>aiogram.types.audio.Audio</code> <code>ampu6ym</code>), 20
<code>DKK</code> (<code>aiogram.enums.currency.Currency</code> <code>ampu6ym</code>), 494	<code>duration</code> (<code>aiogram.types.input_media_animation.InputMediaAnimation</code> <code>ampu6ym</code>), 148
<code>do()</code> (<code>aiogram.types.chat.Chat</code> <code>memo6</code>), 44	<code>duration</code> (<code>aiogram.types.input_media_audio.InputMediaAudio</code> <code>ampu6ym</code>), 149
<code>DOCUMENT</code> (<code>aiogram.enums.content_type.ContentType</code> <code>ampu6ym</code>), 491	<code>duration</code> (<code>aiogram.types.input_media_video.InputMediaVideo</code> <code>ampu6ym</code>), 152
<code>DOCUMENT</code> (<code>aiogram.enums.inline_query_result_type.InlineQueryResultType</code> <code>ampu6ym</code>), 497	<code>duration</code> (<code>aiogram.types.video.Video</code> <code>ampu6ym</code>), 235
<code>DOCUMENT</code> (<code>aiogram.enums.input_media_type.InputMediaType</code> <code>ampu6ym</code>), 498	<code>duration</code> (<code>aiogram.types.video_chat_ended.VideoChatEnded</code> <code>ampu6ym</code>), 236
<code>document</code> (<code>aiogram.methods.send_document.SendDocument</code> <code>ampu6ym</code>), 400	<code>duration</code> (<code>aiogram.types.video_note.VideoNote</code> <code>ampu6ym</code>), 237
<code>document</code> (<code>aiogram.types.background_type_pattern.BackgroundTypePattern</code> <code>ampu6ym</code>), 24	<code>duration</code> (<code>aiogram.types.voice.Voice</code> <code>ampu6ym</code>), 238
<code>document</code> (<code>aiogram.types.background_type_wallpaper.BackgroundTypeWallpaper</code> <code>ampu6ym</code>), 25	<code>DZD</code> (<code>aiogram.enums.currency.Currency</code> <code>ampu6ym</code>), 494
<code>document</code> (<code>aiogram.types.external_reply_info.ExternalReplyInfo</code> <code>memo6</code>), 212	<code>edit_caption()</code> (<code>aiogram.types.message.Message</code> <code>ampu6ym</code>), 166
<code>document</code> (<code>aiogram.types.message.Message</code> <code>ampu6ym</code>), 166	<code>edit_invite_link()</code> (<code>aiogram.types.chat.Chat</code> <code>memo6</code>), 43
<code>Document</code> (клас в <code>aiogram.types.document</code>), 134	<code>edit_live_location()</code> (<code>aiogram.types.message.Message</code> <code>ampu6ym</code>), 211
<code>document_file_id</code> (<code>aiogram.types.inline_query_result_cached_document.InlineQueryResultCachedDocument</code> <code>ampu6ym</code>), 250	<code>edit_reply_markup()</code> (<code>aiogram.types.message.Message</code> <code>memo6</code>), 210
<code>document_url</code> (<code>aiogram.types.inline_query_result_document.InlineQueryResultDocument</code> <code>ampu6ym</code>), 266	<code>edit_text()</code> (<code>aiogram.types.message.Message</code> <code>memo6</code>), 208
<code>DOP</code> (<code>aiogram.enums.currency.Currency</code> <code>ampu6ym</code>), 494	<code>EditChatInviteLink</code> (клас в <code>aiogram.methods.edit_chat_invite_link</code>), 354
<code>download()</code> (<code>aiogram.client.bot.Bot</code> <code>memo6</code>), 505	<code>EDITED_BUSINESS_MESSAGE</code> (<code>aiogram.enums.update_type.UpdateType</code> <code>ampu6ym</code>), 503
<code>download_file()</code> (<code>aiogram.client.bot.Bot</code> <code>memo6</code>), 504	
<code>DRIVER_LICENSE</code> (<code>aiogram.enums.encrypted_passport_element.EncryptedPassportElement</code> <code>ampu6ym</code>), 496	
<code>drop_pending_updates</code> (<code>aiogram.methods.delete_webhook.DeleteWebhook</code> <code>ampu6ym</code>), 482	
<code>drop_pending_updates</code> (<code>aiogram.methods.delete_webhook.DeleteWebhook</code> <code>ampu6ym</code>), 482	

edited_business_message (aiogram.types.update.Update ampубым), 310
 EDITED_CHANNEL_POST (aiogram.enums.update_type.UpdateType ampубым), 502
 edited_channel_post (aiogram.types.update.Update ampубым), 310
 EDITED_MESSAGE (aiogram.enums.update_type.UpdateType ampубым), 502
 edited_message (aiogram.types.update.Update ampубым), 310
 EditForumTopic (клас в aiogram.methods.edit_forum_topic), 355
 EditGeneralForumTopic (клас в aiogram.methods.edit_general_forum_topic), 356
 EditMessageCaption (клас в aiogram.methods.edit_message_caption), 454
 EditMessageLiveLocation (клас в aiogram.methods.edit_message_live_location), 455
 EditMessageMedia (клас в aiogram.methods.edit_message_media), 458
 EditMessageReplyMarkup (клас в aiogram.methods.edit_message_reply_markup), 459
 EditMessageText (клас в aiogram.methods.edit_message_text), 460
 EGP (aiogram.enums.currency.Currency ampубым), 494
 element_hash (aiogram.types.passport_element_error_unspecified.PassportElementErrorUnspecified ampубым), 303
 EMAIL (aiogram.enums.encrypted_passport_element.EncryptedPassportElement ampубым), 497
 EMAIL (aiogram.enums.message_entity_type.MessageEntityType ampубым), 499
 email (aiogram.types.encrypted_passport_element.EncryptedPassportElement ampубым), 294
 email (aiogram.types.order_info.OrderInfo ampубым), 305
 Email (клас в aiogram.utils.formatting), 606
 EMOJI (aiogram.enums.reaction_type_type.ReactionTypeType ampубым), 501
 emoji (aiogram.methods.send_dice.SendDice ampубым), 398
 emoji (aiogram.methods.send_sticker.SendSticker ampубым), 323
 emoji (aiogram.types.dice.Dice ampубым), 134
 emoji (aiogram.types.reaction_type_emoji.ReactionTypeEmoji ampубым), 573
 emoji (aiogram.types.sticker.Sticker ampубым), 291
 emoji_list (aiogram.methods.set_sticker_emoji_list.SetStickerEmojiList ampубым), 326
 emoji_list (aiogram.types.input_sticker.InputSticker ampубым), 289
 emoji_status_custom_emoji_id (aiogram.types.chat.Chat ampубым), 38
 emoji_status_custom_emoji_id (aiogram.types.chat_full_info.ChatFullInfo ampубым), 60
 emoji_status_expiration_date (aiogram.types.chat.Chat ampубым), 38
 emoji_status_expiration_date (aiogram.types.chat_full_info.ChatFullInfo ampубым), 61
 EncryptedCredentials (клас в aiogram.types.encrypted_credentials), 293
 EncryptedPassportElement (клас в aiogram.enums.encrypted_passport_element), 496
 EncryptedPassportElement (клас в aiogram.types.encrypted_passport_element), 294
 enter() (aiogram.fsm.scene.ScenesManager метод), 566
 enter() (aiogram.fsm.scene.Scene Wizard метод), 567
 entities (aiogram.methods.edit_message_text.EditMessageText ampубым), 461
 entities (aiogram.methods.send_message.SendMessage ampубым), 408
 entities (aiogram.types.input_text_message_content.InputTextMessageContent ampубым), 287
 entities (aiogram.types.message.Message ampубым), 280
 entities (aiogram.types.text_quote.TextQuote ampубым), 280
 error_message (aiogram.methods.answer_pre_checkout_query.AnswerPreCheckoutQuery ampубым), 475
 error_message (aiogram.methods.answer_shipping_query.AnswerShippingQuery ampубым), 475
 ErrorEvent (клас в aiogram.types.error_event), 573
 errors (aiogram.methods.set_passport_data_errors.SetPassportDataErrors ampубым), 488
 ETB (aiogram.enums.currency.Currency ampубым), 494
 EUR (aiogram.enums.currency.Currency ampубым), 494
 event (aiogram.types.update.Update property), 312
 event_type (aiogram.types.update.Update property), 312
 exception (aiogram.types.error_event.ErrorEvent ampубым), 573
 ExceptionMessageFilter (клас в aiogram.types.exception_message_filter.ExceptionMessageFilter ampубым), 573

ogram.filters.exception), 528
 exceptions (aiogram.filters.exception.ExceptionTypeFilter
 ampuбym), 528
 ExceptionTypeFilter (клас в aiogram.filters.exception), 528
 exit() (aiogram.fsm.scene.SceneWizard метод), 567
 expiration_date (aiogram.types.chat_boost.ChatBoost ampuбym), 54
 expire_date (aiogram.methods.create_chat_invite_link.CreateChatInviteLink ampuбym), 346
 expire_date (aiogram.methods.edit_chat_invite_link.EditChatInviteLink ampuбym), 354
 expire_date (aiogram.types.chat_invite_link.ChatInviteLink ampuбym), 299
 ampuбym), 63
 explanation (aiogram.methods.send_poll.SendPoll ampuбym), 415
 explanation (aiogram.types.poll.Poll ampuбym), 221
 explanation_entities (aiogram.methods.send_poll.SendPoll ampuбym), 415
 explanation_entities (aiogram.types.poll.Poll ampuбym), 221
 explanation_parse_mode (aiogram.methods.send_poll.SendPoll ampuбym), 415
 export() (aiogram.utils.keyboard.InlineKeyboardBuilder метод), 584
 export() (aiogram.utils.keyboard.ReplyKeyboardBuilder метод), 585
 export_invite_link() (aiogram.types.chat.Chat метод), 44
 ExportChatInviteLink (клас в aiogram.methods.export_chat_invite_link), 357
 external_reply (aiogram.types.message.Message ampuбym), 165
 ExternalReplyInfo (клас в aiogram.types.external_reply_info), 135
 extract_flags() (в модулі aiogram.dispatcher.flags), 576
 extract_from() (aiogram.types.message_entity.MessageEntity метод), 215
 EYES (aiogram.enums.mask_position_point.MaskPositionPoint ampuбym), 498
F
 feed_raw_update() (aiogram.dispatcher.dispatcher.Dispatcher метод), 514
 feed_update() (aiogram.dispatcher.dispatcher.Dispatcher метод), 514
 field_name (aiogram.types.passport_element_error_data_field.PassportElementErrorDataField ampuбym), 297
 file (aiogram.client.telegram.TelegramAPIServer ampuбym), 14
 FILE (aiogram.enums.passport_element_error_type.PassportElementErrorType ampuбym), 500
 File (клас в aiogram.types.file), 137
 file_date (aiogram.types.passport_file.PassportFile ampuбym), 303
 file_hash (aiogram.types.passport_element_error_file.PassportElementErrorFile ampuбym), 297
 file_hash (aiogram.types.passport_element_error_front_side.PassportElementErrorFrontSide ampuбym), 299
 file_hash (aiogram.types.passport_element_error_reverse_side.PassportElementErrorReverseSide ampuбym), 299
 file_hash (aiogram.types.passport_element_error_selfie.PassportElementErrorSelfie ampuбym), 300
 file_hash (aiogram.types.passport_element_error_translation_file.PassportElementErrorTranslationFile ampuбym), 301
 file_hashes (aiogram.types.passport_element_error_files.PassportElementErrorFiles ampuбym), 298
 file_hashes (aiogram.types.passport_element_error_translation_files.PassportElementErrorTranslationFiles ampuбym), 302
 file_id (aiogram.methods.get_file.GetFile ampuбym), 367
 file_id (aiogram.types.animation.Animation ampuбym), 19
 file_id (aiogram.types.audio.Audio ampuбym), 20
 file_id (aiogram.types.document.Document ampuбym), 134
 file_id (aiogram.types.file.File ampuбym), 137
 file_id (aiogram.types.passport_file.PassportFile ampuбym), 303
 file_id (aiogram.types.photo_size.PhotoSize ampuбym), 220
 file_id (aiogram.types.sticker.Sticker ampuбym), 290
 file_id (aiogram.types.video.Video ampuбym), 235
 file_id (aiogram.types.video_note.VideoNote ampuбym), 237
 file_id (aiogram.types.voice.Voice ampuбym), 238
 file_name (aiogram.types.animation.Animation ampuбym), 20
 file_name (aiogram.types.audio.Audio ampuбym), 20
 file_name (aiogram.types.document.Document ampuбym), 135
 file_name (aiogram.types.video.Video ampuбym), 235
 file_path (aiogram.types.file.File ampuбym), 137
 file_size (aiogram.types.animation.Animation ampuбym), 20

<code>file_size</code> (<i>aiogram.types.audio.Audio</i> <i>ампубум</i>), 21	<code>FIND_LOCATION</code> (<i>aiogram.enums.chat_action.ChatAction</i> <i>ампубум</i>), 490
<code>file_size</code> (<i>aiogram.types.document.Document</i> <i>ампубум</i>), 135	<code>find_location()</code> (<i>aiogram.utils.chat_action.ChatActionSender</i> <i>class method</i>), 592
<code>file_size</code> (<i>aiogram.types.file.File</i> <i>ампубум</i>), 137	<code>first_name</code> (<i>aiogram.methods.send_contact.SendContact</i> <i>ампубум</i>), 396
<code>file_size</code> (<i>aiogram.types.passport_file.PassportFile</i> <i>ампубум</i>), 304	<code>first_name</code> (<i>aiogram.types.chat.Chat</i> <i>ампубум</i>), 36
<code>file_size</code> (<i>aiogram.types.photo_size.PhotoSize</i> <i>ампубум</i>), 220	<code>first_name</code> (<i>aiogram.types.chat_full_info.ChatFullInfo</i> <i>ампубум</i>), 60
<code>file_size</code> (<i>aiogram.types.sticker.Sticker</i> <i>ампубум</i>), 291	<code>first_name</code> (<i>aiogram.types.contact.Contact</i> <i>ампубум</i>), 133
<code>file_size</code> (<i>aiogram.types.video.Video</i> <i>ампубум</i>), 235	<code>first_name</code> (<i>aiogram.types.inline_query_result_contact.InlineQueryResultContact</i> <i>ампубум</i>), 263
<code>file_size</code> (<i>aiogram.types.video_note.VideoNote</i> <i>ампубум</i>), 237	<code>first_name</code> (<i>aiogram.types.input_contact_message_content.InputContactMessageContent</i> <i>ампубум</i>), 282
<code>file_size</code> (<i>aiogram.types.voice.Voice</i> <i>ампубум</i>), 238	<code>first_name</code> (<i>aiogram.types.shared_user.SharedUser</i> <i>ампубум</i>), 228
<code>file_unique_id</code> (<i>aiogram.types.animation.Animation</i> <i>ампубум</i>), 19	<code>first_name</code> (<i>aiogram.types.user.User</i> <i>ампубум</i>), 231
<code>file_unique_id</code> (<i>aiogram.types.audio.Audio</i> <i>ампубум</i>), 20	<code>first_name</code> (<i>aiogram.utils.web_app.WebAppUser</i> <i>ампубум</i>), 596
<code>file_unique_id</code> (<i>aiogram.types.document.Document</i> <i>ампубум</i>), 134	<code>FOOTBALL</code> (<i>aiogram.enums.dice_emoji.DiceEmoji</i> <i>ампубум</i>), 496
<code>file_unique_id</code> (<i>aiogram.types.file.File</i> <i>ампубум</i>), 137	<code>FOOTBALL</code> (<i>aiogram.types.dice.DiceEmoji</i> <i>ампубум</i>), 134
<code>file_unique_id</code> (<i>aiogram.types.passport_file.PassportFile</i> <i>ампубум</i>), 303	<code>for_channels</code> (<i>aiogram.methods.get_my_default_administrator_rights.GetMyDefaultAdministratorRights</i> <i>ампубум</i>), 371
<code>file_unique_id</code> (<i>aiogram.types.photo_size.PhotoSize</i> <i>ампубум</i>), 220	<code>for_channels</code> (<i>aiogram.methods.set_my_default_administrator_rights.SetMyDefaultAdministratorRights</i> <i>ампубум</i>), 440
<code>file_unique_id</code> (<i>aiogram.types.sticker.Sticker</i> <i>ампубум</i>), 290	<code>force</code> (<i>aiogram.methods.set_game_score.SetGameScore</i> <i>ампубум</i>), 473
<code>file_unique_id</code> (<i>aiogram.types.video.Video</i> <i>ампубум</i>), 235	<code>force_reply</code> (<i>aiogram.types.force_reply.ForceReply</i> <i>ампубум</i>), 138
<code>file_unique_id</code> (<i>aiogram.types.video_note.VideoNote</i> <i>ампубум</i>), 237	<code>ForceReply</code> (клас в <i>aiogram.types.force_reply</i>), 138
<code>file_unique_id</code> (<i>aiogram.types.voice.Voice</i> <i>ампубум</i>), 238	<code>FOREHEAD</code> (<i>aiogram.enums.mask_position_point.MaskPositionPoint</i> <i>ампубум</i>), 498
<code>file_url()</code> (<i>aiogram.client.telegram.TelegramAPIServer</i> <i>метод</i>), 14	<code>format</code> (<i>aiogram.methods.set_sticker_set_thumbnail.SetStickerSetThumbnail</i> <i>ампубум</i>), 330
<code>FILES</code> (<i>aiogram.enums.passport_element_error_type.PassportElementErrorType</i> <i>ампубум</i>), 500	<code>format</code> (<i>aiogram.types.input_sticker.InputSticker</i> <i>ампубум</i>), 289
<code>files</code> (<i>aiogram.types.encrypted_passport_element.EncryptedPassportElement</i> <i>ампубум</i>), 294	<code>FORUM_TOPIC_CLOSED</code> (<i>aiogram.enums.content_type.ContentType</i> <i>ампубум</i>), 491
<code>fill</code> (<i>aiogram.types.background_type_fill.BackgroundTypeFill</i> <i>ампубум</i>), 24	<code>forum_topic_closed</code> (<i>aiogram.types.message.Message</i> <i>ампубум</i>), 169
<code>fill</code> (<i>aiogram.types.background_type_pattern.BackgroundTypePattern</i> <i>ампубум</i>), 24	<code>FORUM_TOPIC_CREATED</code> (<i>aiogram.enums.content_type.ContentType</i> <i>ампубум</i>), 492
<code>Filter</code> (клас в <i>aiogram.filters.base</i>), 528	<code>forum_topic_created</code> (<i>aiogram.types.message.Message</i> <i>ампубум</i>), 169
<code>filter()</code> (<i>aiogram.filters.callback_data.CallbackData</i> <i>class method</i>), 525	<code>FORUM_TOPIC_EDITED</code> (<i>aiogram.enums.content_type.ContentType</i> <i>ампубум</i>), 493

ogram.enums.content_type.ContentType
ampubym), 492
forum_topic_edited (*aiogram.types.message.Message* *ampubym*), 169
FORUM_TOPIC_REOPENED (*aiogram.enums.content_type.ContentType* *ampubym*), 492
forum_topic_reopened (*aiogram.types.message.Message* *ampubym*), 169
ForumTopic (клас в *aiogram.types.forum_topic*), 138
ForumTopicClosed (клас в *aiogram.types.forum_topic_closed*), 139
ForumTopicCreated (клас в *aiogram.types.forum_topic_created*), 139
ForumTopicEdited (клас в *aiogram.types.forum_topic_edited*), 140
ForumTopicReopened (клас в *aiogram.types.forum_topic_reopened*), 140
forward() (*aiogram.types.message.Message* метод), 209
forward_date (*aiogram.types.message.Message* *ampubym*), 170
forward_from (*aiogram.types.message.Message* *ampubym*), 170
forward_from_chat (*aiogram.types.message.Message* *ampubym*), 170
forward_from_message_id (*aiogram.types.message.Message* *ampubym*), 170
forward_origin (*aiogram.types.message.Message* *ampubym*), 165
forward_sender_name (*aiogram.types.message.Message* *ampubym*), 170
forward_signature (*aiogram.types.message.Message* *ampubym*), 170
forward_text (*aiogram.types.login_url.LoginUrl* *ampubym*), 160
ForwardMessage (клас в *aiogram.methods.forward_message*), 358
ForwardMessages (клас в *aiogram.methods.forward_messages*), 360
foursquare_id (*aiogram.methods.send_venue.SendVenue* *ampubym*), 417
foursquare_id (*aiogram.types.inline_query_result_venue.InlineQueryResultVenue* *ampubym*), 277
foursquare_id (*aiogram.types.input_venue_message_content.InputVenueMessageContent* *ampubym*), 288
foursquare_id (*aiogram.types.venue.Venue* *ampubym*), 234
foursquare_type (*aiogram.methods.send_venue.SendVenue* *ampubym*), 417
foursquare_type (*aiogram.types.inline_query_result_venue.InlineQueryResultVenue* *ampubym*), 277
foursquare_type (*aiogram.types.input_venue_message_content.InputVenueMessageContent* *ampubym*), 288
foursquare_type (*aiogram.types.venue.Venue* *ampubym*), 235
from_attachment_menu (*aiogram.types.write_access_allowed.WriteAccessAllowed* *ampubym*), 239
from_base() (*aiogram.client.telegram.TelegramAPIServer* class method), 15
from_chat_id (*aiogram.methods.copy_message.CopyMessage* *ampubym*), 342
from_chat_id (*aiogram.methods.copy_messages.CopyMessages* *ampubym*), 344
from_chat_id (*aiogram.methods.forward_message.ForwardMessage* *ampubym*), 359
from_chat_id (*aiogram.methods.forward_messages.ForwardMessages* *ampubym*), 360
from_file() (*aiogram.types.input_file.BufferedInputFile* class method), 146
from_markup() (*aiogram.utils.keyboard.InlineKeyboardBuilder* class method), 584
from_markup() (*aiogram.utils.keyboard.ReplyKeyboardBuilder* class method), 585
from_request (*aiogram.types.write_access_allowed.WriteAccessAllowed* *ampubym*), 239
from_url() (*aiogram.fsm.storage.redis.RedisStorage* class method), 548
from_user (*aiogram.handlers.callback_query.CallbackQueryHandler* property), 578
from_user (*aiogram.types.callback_query.CallbackQuery* *ampubym*), 34
from_user (*aiogram.types.chat_join_request.ChatJoinRequest* *ampubym*), 63
from_user (*aiogram.types.chat_member_updated.ChatMemberUpdated* *ampubym*), 111
from_user (*aiogram.types.chosen_inline_result.ChosenInlineResult* *ampubym*), 240
from_user (*aiogram.types.inline_query.InlineQuery* *ampubym*), 240
from_user (*aiogram.types.message.Message* *ampubym*), 165
from_user (*aiogram.types.pre_checkout_query.PreCheckoutQuery* *ampubym*), 305
from_user (*aiogram.types.shipping_query.ShippingQuery* *ampubym*), 308
FRONT_SIDE (*aiogram.enums.passport_element_error_type.PassportElementErrorType*), 548

`ampuonym`), 500
`front_side` (`aiogram.types.encrypted_passport_element.EncryptedPassportElement` `ampuonym`), 295
`FSInputFile` (клас в `aiogram.types.input_file`), 146, 506
`FSMI18nMiddleware` (клас в `aiogram.utils.i18n.middleware`), 588
`full_name` (`aiogram.types.chat.Chat` property), 41
`full_name` (`aiogram.types.user.User` property), 232
G
`GAME` (`aiogram.enums.content_type.ContentType` `ampuonym`), 491
`GAME` (`aiogram.enums.inline_query_result_type.InlineQueryResultType` `ampuonym`), 497
`game` (`aiogram.types.external_reply_info.ExternalReplyInfo` `ampuonym`), 136
`game` (`aiogram.types.message.Message` `ampuonym`), 167
`Game` (клас в `aiogram.types.game`), 313
`game_short_name` (`aiogram.methods.send_game.SendGame` `ampuonym`), 471
`game_short_name` (`aiogram.types.callback_query.CallbackQuery` `ampuonym`), 35
`game_short_name` (`aiogram.types.inline_query_result_game.InlineQueryResultGame` `ampuonym`), 267
`GameHighScore` (клас в `aiogram.types.game_high_score`), 314
`GBP` (`aiogram.enums.currency.Currency` `ampuonym`), 494
`GEL` (`aiogram.enums.currency.Currency` `ampuonym`), 494
`GENERAL_FORUM_TOPIC_HIDDEN` (`aiogram.enums.content_type.ContentType` `ampuonym`), 492
`general_forum_topic_hidden` (`aiogram.types.message.Message` `ampuonym`), 169
`GENERAL_FORUM_TOPIC_UNHIDDEN` (`aiogram.enums.content_type.ContentType` `ampuonym`), 492
`general_forum_topic_unhidden` (`aiogram.types.message.Message` `ampuonym`), 169
`GeneralForumTopicHidden` (клас в `aiogram.types.general_forum_topic_hidden`), 140
`GeneralForumTopicUnhidden` (клас в `aiogram.types.general_forum_topic_unhidden`), 141
`get()` (`aiogram.fsm.scene.SceneRegistry` `memod`), 561
`get_administrators()` (`aiogram.types.chat.Chat` `memod`), 42
`get_data()` (`aiogram.fsm.scene.SceneWizard` `memod`), 567
`get_data()` (`aiogram.fsm.storage.base.BaseStorage` `memod`), 550
`get_flag()` (в `модулі aiogram.dispatcher.flags`), 576
`get_locale()` (`aiogram.utils.i18n.middleware.I18nMiddleware` `memod`), 589
`get_member()` (`aiogram.types.chat.Chat` `memod`), 45
`get_member_count()` (`aiogram.types.chat.Chat` `memod`), 46
`get_profile_photos()` (`aiogram.types.user.User` `memod`), 232
`get_state()` (`aiogram.fsm.storage.base.BaseStorage` `memod`), 550
`get_url()` (`aiogram.types.message.Message` `memod`), 214
`GetBusinessConnection` (клас в `aiogram.methods.get_business_connection`), 361
`GetChat` (клас в `aiogram.methods.get_chat`), 362
`GetChatAdministrators` (клас в `aiogram.methods.get_chat_administrators`), 363
`GetChatMember` (клас в `aiogram.methods.get_chat_member`), 364
`GetChatMemberCount` (клас в `aiogram.methods.get_chat_member_count`), 365
`GetChatMenuButton` (клас в `aiogram.methods.get_chat_menu_button`), 366
`GetCustomEmojiStickers` (клас в `aiogram.methods.get_custom_emoji_stickers`), 319
`GetFile` (клас в `aiogram.methods.get_file`), 367
`GetForumTopicIconStickers` (клас в `aiogram.methods.get_forum_topic_icon_stickers`), 368
`GetGameHighScores` (клас в `aiogram.methods.get_game_high_scores`), 469
`GetMe` (клас в `aiogram.methods.get_me`), 369
`GetMyCommands` (клас в `aiogram.methods.get_my_commands`), 370
`GetMyDefaultAdministratorRights` (клас в `aiogram.methods.get_my_default_administrator_rights`), 371
`GetMyDescription` (клас в `aiogram.methods.get_my_description`),

372	GetMyName (класс в aiogram.methods.get_my_name), 373	giveaway_message_id (aiogram.types.chat_boost_source_giveaway.ChatBoostSourceC
373	GetMyShortDescription (класс в aiogram.methods.get_my_short_description), 374	giveaway_message_id (aiogram.types.giveaway_winners.GiveawayWinners
374	GetStickerSet (класс в aiogram.methods.get_sticker_set), 320	giveaway_message_id (aiogram.types.giveaway_winners.GiveawayWinners
483	GetUpdates (класс в aiogram.methods.get_updates), 483	GIVEAWAY_WINNERS (aiogram.enums.content_type.ContentType
374	GetUserChatBoosts (класс в aiogram.methods.get_user_chat_boosts), 374	giveaway_winners (aiogram.types.external_reply_info.ExternalReplyInfo
375	GetUserProfilePhotos (класс в aiogram.methods.get_user_profile_photos), 375	giveaway_winners (aiogram.types.message.Message
485	GetWebhookInfo (класс в aiogram.methods.get_webhook_info), 485	GiveawayCompleted (класс в aiogram.types.giveaway_completed), 142
497	GIF (aiogram.enums.inline_query_result_type.InlineQueryResultType), 497	GiveawayCreated (класс в aiogram.types.giveaway_created), 142
269	gif_duration (aiogram.types.inline_query_result_gif.InlineQueryResultGif), 269	GiveawayWinners (класс в aiogram.types.giveaway_winners), 142
252	gif_file_id (aiogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif), 252	google_place_id (aiogram.types.giveaway_winners), 142
269	gif_height (aiogram.types.inline_query_result_gif.InlineQueryResultGif), 269	google_place_id (aiogram.types.giveaway_winners), 142
268	gif_url (aiogram.types.inline_query_result_gif.InlineQueryResultGif), 268	google_place_id (aiogram.types.giveaway_winners), 142
268	gif_width (aiogram.types.inline_query_result_gif.InlineQueryResultGif), 268	google_place_id (aiogram.types.giveaway_winners), 142
490	GIFT_CODE (aiogram.enums.chat_boost_source_type.ChatBoostSourceType), 490	google_place_id (aiogram.types.giveaway_winners), 142
490	GIVEAWAY (aiogram.enums.chat_boost_source_type.ChatBoostSourceType), 490	google_place_id (aiogram.types.giveaway_winners), 142
493	GIVEAWAY (aiogram.enums.content_type.ContentType), 493	google_place_id (aiogram.types.giveaway_winners), 142
136	giveaway (aiogram.types.external_reply_info.ExternalReplyInfo), 136	google_place_id (aiogram.types.giveaway_winners), 142
169	giveaway (aiogram.types.message.Message), 169	google_place_id (aiogram.types.giveaway_winners), 142
141	Giveaway (класс в aiogram.types.giveaway), 141	google_place_id (aiogram.types.giveaway_winners), 142
493	GIVEAWAY_COMPLETED (aiogram.enums.content_type.ContentType), 493	google_place_id (aiogram.types.giveaway_winners), 142
169	giveaway_completed (aiogram.types.message.Message), 169	google_place_id (aiogram.types.giveaway_winners), 142
493	GIVEAWAY_CREATED (aiogram.enums.content_type.ContentType), 493	google_place_id (aiogram.types.giveaway_winners), 142
169	giveaway_created (aiogram.types.message.Message), 169	google_place_id (aiogram.types.giveaway_winners), 142
169	giveaway_message (aiogram.types.giveaway_completed.GiveawayCompleted), 169	google_place_id (aiogram.types.giveaway_winners), 142

- 167
- GTQ (*aiogram.enums.currency.Currency* *ампубѣм*), 494
- ## H
- handlers (*aiogram.fsm.scene.SceneConfig* *ампубѣм*), 566
- has_aggressive_anti_spam_enabled (*aiogram.types.chat.Chat* *ампубѣм*), 38
- has_aggressive_anti_spam_enabled (*aiogram.types.chat_full_info.ChatFullInfo* *ампубѣм*), 61
- has_custom_certificate (*aiogram.types.webhook_info.WebhookInfo* *ампубѣм*), 312
- has_hidden_members (*aiogram.types.chat.Chat* *ампубѣм*), 38
- has_hidden_members (*aiogram.types.chat_full_info.ChatFullInfo* *ампубѣм*), 62
- has_media_spoiler (*aiogram.types.external_reply_info.ExternalReplyInfo* *ампубѣм*), 136
- has_media_spoiler (*aiogram.types.message.Message* *ампубѣм*), 167
- has_private_forwards (*aiogram.types.chat.Chat* *ампубѣм*), 38
- has_private_forwards (*aiogram.types.chat_full_info.ChatFullInfo* *ампубѣм*), 61
- has_protected_content (*aiogram.types.chat.Chat* *ампубѣм*), 39
- has_protected_content (*aiogram.types.chat_full_info.ChatFullInfo* *ампубѣм*), 62
- has_protected_content (*aiogram.types.message.Message* *ампубѣм*), 166
- has_public_winners (*aiogram.types.giveaway.Giveaway* *ампубѣм*), 141
- has_restricted_voice_and_video_messages (*aiogram.types.chat.Chat* *ампубѣм*), 39
- has_restricted_voice_and_video_messages (*aiogram.types.chat_full_info.ChatFullInfo* *ампубѣм*), 61
- has_spoiler (*aiogram.methods.send_animation.SendAnimation* *ампубѣм*), 389
- has_spoiler (*aiogram.methods.send_photo.SendPhoto* *ампубѣм*), 412
- has_spoiler (*aiogram.methods.send_video.SendVideo* *ампубѣм*), 421
- has_spoiler (*aiogram.types.input_media_animation.InputMediaAnimation* *ампубѣм*), 148
- has_spoiler (*aiogram.types.input_media_photo.InputMediaPhoto* *ампубѣм*), 151
- has_spoiler (*aiogram.types.input_media_video.InputMediaVideo* *ампубѣм*), 152
- has_visible_history (*aiogram.types.chat.Chat* *ампубѣм*), 39
- has_visible_history (*aiogram.types.chat_full_info.ChatFullInfo* *ампубѣм*), 62
- hash (*aiogram.types.encrypted_credentials.EncryptedCredentials* *ампубѣм*), 293
- hash (*aiogram.types.encrypted_passport_element.EncryptedPassportElement* *ампубѣм*), 294
- hash (*aiogram.utils.web_app.WebAppInitData* *ампубѣм*), 596
- HASHTAG (*aiogram.enums.message_entity_type.MessageEntityType* *ампубѣм*), 499
- HashTag (клас в *aiogram.utils.formatting*), 605
- heading (*aiogram.methods.edit_message_live_location.EditMessageLiveLocation* *ампубѣм*), 457
- heading (*aiogram.methods.send_location.SendLocation* *ампубѣм*), 403
- heading (*aiogram.types.inline_query_result_location.InlineQueryResultLocation* *ампубѣм*), 271
- heading (*aiogram.types.input_location_message_content.InputLocationMessageContent* *ампубѣм*), 286
- heading (*aiogram.types.location.Location* *ампубѣм*), 159
- height (*aiogram.methods.send_animation.SendAnimation* *ампубѣм*), 389
- height (*aiogram.methods.send_video.SendVideo* *ампубѣм*), 421
- height (*aiogram.types.animation.Animation* *ампубѣм*), 19
- height (*aiogram.types.input_media_animation.InputMediaAnimation* *ампубѣм*), 148
- height (*aiogram.types.input_media_video.InputMediaVideo* *ампубѣм*), 152
- height (*aiogram.types.photo_size.PhotoSize* *ампубѣм*), 220
- height (*aiogram.types.sticker.Sticker* *ампубѣм*), 291
- height (*aiogram.types.video.Video* *ампубѣм*), 235
- HIDDEN_USER (*aiogram.enums.message_origin_type.MessageOriginType* *ампубѣм*), 500
- hide_url (*aiogram.types.inline_query_result_article.InlineQueryResultArticle* *ампубѣм*), 244
- HideGeneralForumTopic (клас в *aiogram.methods.hide_general_forum_topic*), 376
- HKD (*aiogram.enums.currency.Currency* *ампубѣм*), 494

HNL (<i>aiogram.enums.currency.Currency</i> <i>ampubym</i>), 494	id (<i>aiogram.types.chat.Chat</i> <i>ampubym</i>), 36
horizontal_accuracy (<i>aiogram.methods.edit_message_live_location.EditMessageLiveLocation</i> <i>ampubym</i>), 457	id (<i>aiogram.types.chat_full_info.ChatFullInfo</i> <i>ampubym</i>), 59
horizontal_accuracy (<i>aiogram.methods.send_location.SendLocation</i> <i>ampubym</i>), 403	id (<i>aiogram.types.inline_query.InlineQuery</i> <i>ampubym</i>), 240
horizontal_accuracy (<i>aiogram.types.inline_query_result_location.InlineQueryResultLocation</i> <i>ampubym</i>), 271	id (<i>aiogram.types.inline_query_result_article.InlineQueryResultArticle</i> <i>ampubym</i>), 243
horizontal_accuracy (<i>aiogram.types.inline_query_result_location.InlineQueryResultLocation</i> <i>ampubym</i>), 271	id (<i>aiogram.types.inline_query_result_audio.InlineQueryResultAudio</i> <i>ampubym</i>), 245
horizontal_accuracy (<i>aiogram.types.input_location_message_content.InputLocationMessageContent</i> <i>ampubym</i>), 285	id (<i>aiogram.types.inline_query_result_cached_audio.InlineQueryResultCachedAudio</i> <i>ampubym</i>), 247
horizontal_accuracy (<i>aiogram.types.location.Location</i> <i>ampubym</i>), 159	id (<i>aiogram.types.inline_query_result_cached_document.InlineQueryResultCachedDocument</i> <i>ampubym</i>), 250
HRK (<i>aiogram.enums.currency.Currency</i> <i>ampubym</i>), 494	id (<i>aiogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif</i> <i>ampubym</i>), 251
HTML (<i>aiogram.enums.parse_mode.ParseMode</i> <i>ampubym</i>), 500	id (<i>aiogram.types.inline_query_result_cached_mpeg4_gif.InlineQueryResultCachedMpeg4Gif</i> <i>ampubym</i>), 254
html_text (<i>aiogram.types.message.Message</i> <i>property</i>), 170	id (<i>aiogram.types.inline_query_result_cached_photo.InlineQueryResultCachedPhoto</i> <i>ampubym</i>), 256
HUF (<i>aiogram.enums.currency.Currency</i> <i>ampubym</i>), 494	id (<i>aiogram.types.inline_query_result_cached_sticker.InlineQueryResultCachedSticker</i> <i>ampubym</i>), 257
	id (<i>aiogram.types.inline_query_result_cached_video.InlineQueryResultCachedVideo</i> <i>ampubym</i>), 260
I18nMiddleware (<i>клас в aiogram.utils.i18n.middleware</i>), 589	id (<i>aiogram.types.inline_query_result_cached_voice.InlineQueryResultCachedVoice</i> <i>ampubym</i>), 261
icon_color (<i>aiogram.methods.create_forum_topic.CreateForumTopic</i> <i>ampubym</i>), 347	id (<i>aiogram.types.inline_query_result_contact.InlineQueryResultContact</i> <i>ampubym</i>), 263
icon_color (<i>aiogram.types.forum_topic.ForumTopic</i> <i>ampubym</i>), 139	id (<i>aiogram.types.inline_query_result_document.InlineQueryResultDocument</i> <i>ampubym</i>), 266
icon_color (<i>aiogram.types.forum_topic_created.ForumTopicCreated</i> <i>ampubym</i>), 139	id (<i>aiogram.types.inline_query_result_game.InlineQueryResultGame</i> <i>ampubym</i>), 267
icon_custom_emoji_id (<i>aiogram.methods.create_forum_topic.CreateForumTopic</i> <i>ampubym</i>), 347	id (<i>aiogram.types.inline_query_result_gif.InlineQueryResultGif</i> <i>ampubym</i>), 268
icon_custom_emoji_id (<i>aiogram.methods.edit_forum_topic.EditForumTopic</i> <i>ampubym</i>), 355	id (<i>aiogram.types.inline_query_result_location.InlineQueryResultLocation</i> <i>ampubym</i>), 270
icon_custom_emoji_id (<i>aiogram.types.forum_topic.ForumTopic</i> <i>ampubym</i>), 139	id (<i>aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif</i> <i>ampubym</i>), 273
icon_custom_emoji_id (<i>aiogram.methods.edit_forum_topic.EditForumTopic</i> <i>ampubym</i>), 355	id (<i>aiogram.types.inline_query_result_photo.InlineQueryResultPhoto</i> <i>ampubym</i>), 274
icon_custom_emoji_id (<i>aiogram.types.forum_topic_created.ForumTopicCreated</i> <i>ampubym</i>), 139	id (<i>aiogram.types.inline_query_result_venue.InlineQueryResultVenue</i> <i>ampubym</i>), 276
icon_custom_emoji_id (<i>aiogram.types.forum_topic.ForumTopic</i> <i>ampubym</i>), 139	id (<i>aiogram.types.inline_query_result_video.InlineQueryResultVideo</i> <i>ampubym</i>), 278
icon_custom_emoji_id (<i>aiogram.types.forum_topic_created.ForumTopicCreated</i> <i>ampubym</i>), 139	id (<i>aiogram.types.inline_query_result_voice.InlineQueryResultVoice</i> <i>ampubym</i>), 280
icon_custom_emoji_id (<i>aiogram.types.forum_topic_edited.ForumTopicEdited</i> <i>ampubym</i>), 140	id (<i>aiogram.types.poll.Poll</i> <i>ampubym</i>), 221
id (<i>aiogram.types.business_connection.BusinessConnection</i> <i>ampubym</i>), 31	id (<i>aiogram.types.pre_checkout_query.PreCheckoutQuery</i> <i>ampubym</i>), 305
id (<i>aiogram.types.callback_query.CallbackQuery</i> <i>ampubym</i>), 34	id (<i>aiogram.types.shipping_option.ShippingOption</i> <i>ampubym</i>), 307
	id (<i>aiogram.types.shipping_query.ShippingQuery</i> <i>ampubym</i>), 308
	id (<i>aiogram.types.story.Story</i> <i>ampubym</i>), 229
	id (<i>aiogram.types.user.User</i> <i>ampubym</i>), 231

<code>id</code> (<i>aiogram.utils.web_app.WebAppChat</i> <i>ampubym</i>), 597	<code>INLINE_QUERY</code> (<i>aiogram.enums.update_type.UpdateType</i> <i>ampubym</i>), 503
<code>id</code> (<i>aiogram.utils.web_app.WebAppUser</i> <i>ampubym</i>), 596	<code>inline_query</code> (<i>aiogram.types.update.Update</i> <i>ampubym</i>), 311
<code>IDENTITY_CARD</code> (<i>aiogram.enums.encrypted_passport_element.EncryptedPassportElement</i> <i>ampubym</i>), 496	<code>inline_query_id</code> (<i>aiogram.methods.answer_inline_query.AnswerInlineQuery</i> <i>ampubym</i>), 466
<code>IDR</code> (<i>aiogram.enums.currency.Currency</i> <i>ampubym</i>), 494	<code>InlineKeyboardBuilder</code> (клас в <i>aiogram.utils.keyboard</i>), 583
<code>ILS</code> (<i>aiogram.enums.currency.Currency</i> <i>ampubym</i>), 494	<code>InlineKeyboardButton</code> (клас в <i>aiogram.types.inline_keyboard_button</i>), 144
<code>InaccessibleMessage</code> (клас в <i>aiogram.types.inaccessible_message</i>), 143	<code>InlineKeyboardMarkup</code> (клас в <i>aiogram.types.inline_keyboard_markup</i>), 145
<code>include_router()</code> (<i>aiogram.dispatcher.router.Router</i> <i>метод</i>), 508	<code>InlineQuery</code> (клас в <i>aiogram.types.inline_query</i>), 240
<code>include_routers()</code> (<i>aiogram.dispatcher.router.Router</i> <i>метод</i>), 508	<code>InlineQueryResult</code> (клас в <i>aiogram.types.inline_query_result</i>), 242
<code>inline_keyboard</code> (<i>aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup</i> <i>ampubym</i>), 145	<code>InlineQueryResultArticle</code> (клас в <i>aiogram.types.inline_query_result_article</i>), 243
<code>inline_message_id</code> (<i>aiogram.methods.edit_message_caption.EditMessageCaption</i> <i>ampubym</i>), 454	<code>InlineQueryResultAudio</code> (клас в <i>aiogram.types.inline_query_result_audio</i>), 244
<code>inline_message_id</code> (<i>aiogram.methods.edit_message_live_location.EditMessageLiveLocation</i> <i>ampubym</i>), 456	<code>InlineQueryResultCachedAudio</code> (клас в <i>aiogram.types.inline_query_result_cached_audio</i>), 246
<code>inline_message_id</code> (<i>aiogram.methods.edit_message_media.EditMessageMedia</i> <i>ampubym</i>), 458	<code>InlineQueryResultCachedDocument</code> (клас в <i>aiogram.types.inline_query_result_cached_document</i>), 248
<code>inline_message_id</code> (<i>aiogram.methods.edit_message_reply_markup.EditMessageReplyMarkup</i> <i>ampubym</i>), 460	<code>InlineQueryResultCachedGif</code> (клас в <i>aiogram.types.inline_query_result_cached_gif</i>), 250
<code>inline_message_id</code> (<i>aiogram.methods.edit_message_text.EditMessageText</i> <i>ampubym</i>), 461	<code>InlineQueryResultCachedMpeg4Gif</code> (клас в <i>aiogram.types.inline_query_result_cached_mpeg4_gif</i>), 252
<code>inline_message_id</code> (<i>aiogram.methods.get_game_high_scores.GetGameHighScores</i> <i>ampubym</i>), 470	<code>InlineQueryResultCachedPhoto</code> (клас в <i>aiogram.types.inline_query_result_cached_photo</i>), 254
<code>inline_message_id</code> (<i>aiogram.methods.set_game_score.SetGameScore</i> <i>ampubym</i>), 473	<code>InlineQueryResultCachedSticker</code> (клас в <i>aiogram.types.inline_query_result_cached_sticker</i>), 256
<code>inline_message_id</code> (<i>aiogram.methods.stop_message_live_location.StopMessageLiveLocation</i> <i>ampubym</i>), 463	<code>InlineQueryResultCachedVideo</code> (клас в <i>aiogram.types.inline_query_result_cached_video</i>), 258
<code>inline_message_id</code> (<i>aiogram.types.callback_query.CallbackQuery</i> <i>ampubym</i>), 35	<code>InlineQueryResultCachedVoice</code> (клас в <i>aiogram.types.inline_query_result_cached_voice</i>), 260
<code>inline_message_id</code> (<i>aiogram.types.chosen_inline_result.ChosenInlineResult</i> <i>ampubym</i>), 240	<code>InlineQueryResultContact</code> (клас в <i>aiogram.types.inline_query_result_contact</i>), 263
<code>inline_message_id</code> (<i>aiogram.types.sent_web_app_message.SentWebAppMessage</i> <i>ampubym</i>), 289	<code>InlineQueryResultDocument</code> (клас в <i>aiogram.types.inline_query_result_document</i>), 263

264		ampubym), 254
InlineQueryResultGame	(клас в aiogram.types.inline_query_result_game),	(ai- input_message_content ogram.types.inline_query_result_cached_photo.InlineQueryResultGame), 254
267		ampubym), 256
InlineQueryResultGif	(клас в aiogram.types.inline_query_result_gif),	(ai- input_message_content ogram.types.inline_query_result_cached_sticker.InlineQueryResultGif), 256
267		ampubym), 258
InlineQueryResultLocation	(клас в aiogram.types.inline_query_result_location),	(ai- input_message_content ogram.types.inline_query_result_cached_video.InlineQueryResultLocation), 260
270		ampubym), 260
InlineQueryResultMpeg4Gif	(клас в aiogram.types.inline_query_result_mpeg4_gif),	(ai- input_message_content ogram.types.inline_query_result_cached_voice.InlineQueryResultMpeg4Gif), 262
272		ampubym), 262
InlineQueryResultPhoto	(клас в aiogram.types.inline_query_result_photo),	(ai- input_message_content ogram.types.inline_query_result_contact.InlineQueryResultPhoto), 264
274		ampubym), 264
InlineQueryResultsButton	(клас в aiogram.types.inline_query_results_button),	(ai- input_message_content ogram.types.inline_query_result_document.InlineQueryResultsButton), 266
281		ampubym), 266
InlineQueryResultType	(клас в aiogram.enums.inline_query_result_type),	(ai- input_message_content ogram.types.inline_query_result_gif.InlineQueryResultType), 269
497		ampubym), 269
InlineQueryResultVenue	(клас в aiogram.types.inline_query_result_venue),	(ai- input_message_content ogram.types.inline_query_result_location.InlineQueryResultVenue), 271
275		ampubym), 271
InlineQueryResultVideo	(клас в aiogram.types.inline_query_result_video),	(ai- input_message_content ogram.types.inline_query_result_mpeg4_gif.InlineQueryResultVideo), 273
277		ampubym), 273
InlineQueryResultVoice	(клас в aiogram.types.inline_query_result_voice),	(ai- input_message_content ogram.types.inline_query_result_photo.InlineQueryResultVoice), 275
280		ampubym), 275
input_field_placeholder	(ai- ogram.types.force_reply.ForceReply ampubym), 138	(ai- input_message_content ogram.types.inline_query_result_venue.InlineQueryResultVoice), 277
input_field_placeholder	(ai- ogram.types.reply_keyboard_markup.ReplyKeyboardMarkup ampubym), 225	(ai- input_message_content ogram.types.inline_query_result_video.InlineQueryResultVideo), 279
input_message_content	(ai- ogram.types.inline_query_result_article.InlineQueryResultArticle ampubym), 243	(ai- input_message_content ogram.types.inline_query_result_voice.InlineQueryResultVoice), 281
input_message_content	(ai- ogram.types.inline_query_result_audio.InlineQueryResultAudio ampubym), 246	(ai- InputContactMessageContent (клас в aiogram.types.input_contact_message_content), 282
input_message_content	(ai- ogram.types.inline_query_result_cached_audio.InlineQueryResultCachedAudio ampubym), 248	(ai- InputFile (клас в aiogram.types.input_file), 146
input_message_content	(ai- ogram.types.inline_query_result_cached_document.InlineQueryResultCachedDocument ampubym), 250	(ai- InputInvoiceMessageContent (клас в aiogram.types.input_invoice_message_content), 282
input_message_content	(ai- ogram.types.inline_query_result_cached_location.InlineQueryResultCachedLocation ampubym), 252	(ai- InputLocationMessageContent (клас в aiogram.types.input_location_message_content), 285
input_message_content	(ai- ogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif ampubym), 252	(ai- InputMedia (клас в aiogram.types.input_media), 146
input_message_content	(ai- ogram.types.inline_query_result_cached_mpeg4_gif.InlineQueryResultCachedMpeg4Gif ampubym), 254	(клас в ai- InputMediaAnimation (клас в aiogram.types.input_media_animation), 146

147				<i>ogram.types.pre_checkout_query.PreCheckoutQuery</i>
<code>InputMediaAudio</code>	(клас в <i>aiogram.types.input_media_audio</i>), 148		<code>invoice_payload</code>	(<i>aiogram.types.shipping_query.ShippingQuery</i> <i>ampubym</i>), 308
<code>InputMediaDocument</code>	(клас в <i>aiogram.types.input_media_document</i>), 149		<code>invoice_payload</code>	(<i>aiogram.types.successful_payment.SuccessfulPayment</i> <i>ampubym</i>), 309
<code>InputMediaPhoto</code>	(клас в <i>aiogram.types.input_media_photo</i>), 150		<code>ip_address</code>	(<i>aiogram.methods.set_webhook.SetWebhook</i> <i>ampubym</i>), 486
<code>InputMediaType</code>	(клас в <i>aiogram.enums.input_media_type</i>), 498		<code>ip_address</code>	(<i>aiogram.types.webhook_info.WebhookInfo</i> <i>ampubym</i>), 312
<code>InputMediaVideo</code>	(клас в <i>aiogram.types.input_media_video</i>), 151		<code>ip_filter_middleware()</code>	(в модулі <i>aiogram.webhook.aiohttp_server</i>), 534
<code>InputMessageContent</code>	(клас в <i>aiogram.types.input_message_content</i>), 286		<code>IPFilter</code>	(клас в <i>aiogram.webhook.security</i>), 535
<code>InputPollOption</code>	(клас в <i>aiogram.types.input_poll_option</i>), 152		<code>is_animated</code>	(<i>aiogram.types.sticker.Sticker</i> <i>ampubym</i>), 291
<code>InputSticker</code>	(клас в <i>aiogram.types.input_sticker</i>), 289		<code>is_animated</code>	(<i>aiogram.types.sticker_set.StickerSet</i> <i>ampubym</i>), 292
<code>InputTextMessageContent</code>	(клас в <i>aiogram.types.input_text_message_content</i>), 286		<code>is_anonymous</code>	(<i>aiogram.methods.promote_chat_member.PromoteChatMember</i> <i>ampubym</i>), 381
<code>InputVenueMessageContent</code>	(клас в <i>aiogram.types.input_venue_message_content</i>), 288		<code>is_anonymous</code>	(<i>aiogram.methods.send_poll.SendPoll</i> <i>ampubym</i>), 415
<code>INR</code>	(<i>aiogram.enums.currency.Currency</i> <i>ampubym</i>), 494		<code>is_anonymous</code>	(<i>aiogram.types.chat_administrator_rights.ChatAdministratorRights</i> <i>ampubym</i>), 53
<code>intensity</code>	(<i>aiogram.types.background_type_pattern.BackgroundTypePattern</i> <i>ampubym</i>), 24		<code>is_anonymous</code>	(<i>aiogram.types.chat_member_administrator.ChatMemberAdministrator</i> <i>ampubym</i>), 105
<code>INTERNAL_PASSPORT</code>	(<i>aiogram.enums.encrypted_passport_element.EncryptedPassportElement</i> <i>ampubym</i>), 496		<code>is_anonymous</code>	(<i>aiogram.types.chat_member_owner.ChatMemberOwner</i> <i>ampubym</i>), 108
<code>invite_link</code>	(<i>aiogram.methods.edit_chat_invite_link.EditChatInviteLink</i> <i>ampubym</i>), 354		<code>is_anonymous</code>	(<i>aiogram.types.poll.Poll</i> <i>ampubym</i>), 234
<code>invite_link</code>	(<i>aiogram.methods.revoke_chat_invite_link.RevokeChatInviteLink</i> <i>ampubym</i>), 387		<code>is_automatic_forward</code>	(<i>aiogram.types.message.Message</i> <i>ampubym</i>), 165
<code>invite_link</code>	(<i>aiogram.types.chat.Chat</i> <i>ampubym</i>), 39		<code>is_automatic_forward</code>	(<i>aiogram.types.message_reaction.SetMessageReaction</i> <i>ampubym</i>), 437
<code>invite_link</code>	(<i>aiogram.types.chat_full_info.ChatFullInfo</i> <i>ampubym</i>), 61		<code>is_blurred</code>	(<i>aiogram.types.background_type_wallpaper.BackgroundTypeWallpaper</i> <i>ampubym</i>), 25
<code>invite_link</code>	(<i>aiogram.types.chat_invite_link.ChatInviteLink</i> <i>ampubym</i>), 62		<code>is_bot</code>	(<i>aiogram.types.user.User</i> <i>ampubym</i>), 231
<code>invite_link</code>	(<i>aiogram.types.chat_join_request.ChatJoinRequest</i> <i>ampubym</i>), 63		<code>is_bot</code>	(<i>aiogram.utils.web_app.WebAppUser</i> <i>ampubym</i>), 596
<code>invite_link</code>	(<i>aiogram.types.chat_member_updated.ChatMemberUpdated</i> <i>ampubym</i>), 111		<code>is_closed</code>	(<i>aiogram.methods.send_poll.SendPoll</i> <i>ampubym</i>), 415
<code>INVOICE</code>	(<i>aiogram.enums.content_type.ContentType</i> <i>ampubym</i>), 492		<code>is_closed</code>	(<i>aiogram.types.poll.Poll</i> <i>ampubym</i>), 221
<code>invoice</code>	(<i>aiogram.types.external_reply_info.ExternalReplyInfo</i> <i>ampubym</i>), 136		<code>is_enabled</code>	(<i>aiogram.types.link_preview_options.LinkPreviewOptions</i> <i>ampubym</i>), 158
<code>invoice</code>	(<i>aiogram.types.message.Message</i> <i>ampubym</i>), 168		<code>is_enabled</code>	(<i>aiogram.types.business_connection.BusinessConnection</i> <i>ampubym</i>), 32
<code>Invoice</code>	(клас в <i>aiogram.types.invoice</i>), 304		<code>is_flexible</code>	(<i>aiogram.methods.create_invoice_link.CreateInvoiceLink</i> <i>ampubym</i>), 478
<code>invoice_payload</code>		(<i>aiogram.methods.send_invoice.SendInvoice</i> <i>ampubym</i>), 481	<code>is_flexible</code>	(<i>aiogram.types.input_invoice_message_content.InputInvoiceMessageContent</i> <i>ampubym</i>), 285

[is_forum \(aiogram.types.chat.Chat ampубym\), 36](#)
[is_forum \(aiogram.types.chat_full_info.ChatFullInfo ampубym\), 60](#)
[is_from_offline \(aiogram.types.message.Message ampубym\), 166](#)
[is_inverted \(aiogram.types.background_type_pattern.BackgroundTypePattern ampубym\), 24](#)
[is_local \(aiogram.client.telegram.TelegramAPIServer ampубym\), 15](#)
[is_manual \(aiogram.types.text_quote.TextQuote ampубym\), 230](#)
[is_member \(aiogram.types.chat_member_restricted.ChatMemberRestricted ampубym\), 109](#)
[is_moving \(aiogram.types.background_type_pattern.BackgroundTypePattern ampубym\), 24](#)
[is_moving \(aiogram.types.background_type_wallpaper.BackgroundTypeWallpaper ampубym\), 25](#)
[is_persistent \(aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup ampубym\), 225](#)
[is_personal \(aiogram.methods.answer_inline_query.AnswerInlineQuery ampубym\), 466](#)
[is_premium \(aiogram.types.user.User ampубym\), 231](#)
[is_premium \(aiogram.utils.web_app.WebAppUser ampубym\), 596](#)
[is_primary \(aiogram.types.chat_invite_link.ChatInviteLink ampубym\), 62](#)
[is_revoked \(aiogram.types.chat_invite_link.ChatInviteLink ampубym\), 63](#)
[is_topic_message \(aiogram.types.message.Message ampубym\), 165](#)
[is_unclaimed \(aiogram.types.chat_boost_source_giveaway.ChatBoostSourceGiveaway ampубym\), 57](#)
[is_video \(aiogram.types.sticker.Sticker ampубym\), 291](#)
[is_video \(aiogram.types.sticker_set.StickerSet ampубym\), 292](#)
[ISK \(aiogram.enums.currency.Currency ampубym\), 494](#)
[ITALIC \(aiogram.enums.message_entity_type.MessageEntityType ampубym\), 499](#)
[Italic \(клас в aiogram.utils.formatting\), 606](#)

J

[JMD \(aiogram.enums.currency.Currency ampубym\), 494](#)
[join_by_request \(aiogram.types.chat.Chat ampубym\), 39](#)
[join_by_request \(aiogram.types.chat_full_info.ChatFullInfo ampубym\), 61](#)
[join_to_send_messages \(aiogram.types.chat.Chat ampубym\), 39](#)

[join_to_send_messages \(aiogram.types.chat_full_info.ChatFullInfo ampубym\), 61](#)
[JPY \(aiogram.enums.currency.Currency ampубym\), 494](#)
K
[KES \(aiogram.enums.currency.Currency ampубym\), 494](#)
[keyboard \(aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup ampубym\), 225](#)
[KeyboardButton \(клас в aiogram.types.keyboard_button\), 153](#)
[KeyboardButtonPollType \(клас в aiogram.types.keyboard_button_poll_type\), 154](#)
[KeyboardButtonPollTypeType \(клас в aiogram.types.keyboard_button_poll_type_type\), 498](#)
[KeyboardButtonRequestChat \(клас в aiogram.types.keyboard_button_request_chat\), 155](#)
[KeyboardButtonRequestUser \(клас в aiogram.types.keyboard_button_request_user\), 156](#)
[KeyboardButtonRequestUsers \(клас в aiogram.types.keyboard_button_request_users\), 157](#)
[KeyBuilder \(клас в aiogram.fsm.storage.base\), 549](#)
[keywords \(aiogram.methods.set_sticker_keywords.SetStickerKeywords ampубym\), 327](#)
[keywords \(aiogram.types.sticker_input_sticker.InputStickerKeywords ampубym\), 289](#)
[KGS \(aiogram.enums.currency.Currency ampубym\), 494](#)
[KICKED \(aiogram.enums.chat_member_status.ChatMemberStatus ampубym\), 491](#)
[KRW \(aiogram.enums.currency.Currency ampубym\), 494](#)
[KZT \(aiogram.enums.currency.Currency ampубym\), 494](#)

L

[label \(aiogram.types.labeled_price.LabeledPrice ampубym\), 304](#)
[LabeledPrice \(клас в aiogram.types.labeled_price\), 304](#)
[language \(aiogram.types.message_entity.MessageEntity ampубym\), 215](#)
[language_code \(aiogram.methods.delete_my_commands.DeleteMyCommands ampубym\), 353](#)
[language_code \(aiogram.methods.get_my_commands.GetMyCommands ampубym\), 370](#)

`language_code` (`aiogram.methods.get_my_description.GetMyDescription` `ampu6ym`), 285
`ampu6ym`), 372 `latitude` (`aiogram.types.input_venue_message_content.InputVenueMessageContent` `ampu6ym`), 372
`language_code` (`aiogram.methods.get_my_name.GetMyName` `ampu6ym`), 288
`ampu6ym`), 373 `latitude` (`aiogram.types.location.Location` `ampu6ym`), 373
`language_code` (`aiogram.methods.get_my_short_description.GetMyShortDescription` `ampu6ym`), 374
`ampu6ym`), 374 `LBP` (`aiogram.enums.currency.Currency` `ampu6ym`), 494
`language_code` (`aiogram.methods.set_my_commands.SetMyCommands` `ampu6ym`), 438
`ampu6ym`), 438 `leave()` (`aiogram.fsm.scene.SceneWizard` `memod`), 568
`language_code` (`aiogram.methods.set_my_description.SetMyDescription` `ampu6ym`), 441
`ampu6ym`), 441 `leave()` (`aiogram.types.chat.Chat` `memod`), 46
`language_code` (`aiogram.methods.set_my_name.SetMyName` `ampu6ym`), 441
`ampu6ym`), 441 `leave_chat` (клас в `aiogram.methods.leave_chat`), 377
`language_code` (`aiogram.methods.set_my_short_description.SetMyShortDescription` `ampu6ym`), 443
`ampu6ym`), 443 `LEFT_CHAT_MEMBER_STATUS` (`aiogram.enums.chat_member_status.ChatMemberStatus` `ampu6ym`), 490
`language_code` (`aiogram.types.user.User` `ampu6ym`), 231
`ampu6ym`), 231 `LEFT_CHAT_MEMBER` (`aiogram.enums.content_type.ContentType` `ampu6ym`), 492
`language_code` (`aiogram.utils.web_app.WebAppUser` `ampu6ym`), 596
`ampu6ym`), 596 `left_chat_member` (`aiogram.types.message.Message` `ampu6ym`), 167
`last_error_date` (`aiogram.types.webhook_info.WebhookInfo` `ampu6ym`), 312
`ampu6ym`), 312 `length` (`aiogram.methods.send_video_note.SendVideoNote` `ampu6ym`), 423
`last_error_message` (`aiogram.types.webhook_info.WebhookInfo` `ampu6ym`), 313
`ampu6ym`), 313 `length` (`aiogram.types.message_entity.MessageEntity` `ampu6ym`), 215
`last_name` (`aiogram.methods.send_contact.SendContact` `ampu6ym`), 396
`ampu6ym`), 396 `length` (`aiogram.types.video_note.VideoNote` `ampu6ym`), 237
`last_name` (`aiogram.types.chat.Chat` `ampu6ym`), 36
`ampu6ym`), 36 `limit` (`aiogram.methods.get_updates.GetUpdates` `ampu6ym`), 484
`last_name` (`aiogram.types.chat_full_info.ChatFullInfo` `ampu6ym`), 60
`ampu6ym`), 60 `limit` (`aiogram.methods.get_user_profile_photos.GetUserProfilePhotos` `ampu6ym`), 376
`last_name` (`aiogram.types.contact.Contact` `ampu6ym`), 133
`ampu6ym`), 133 `link_preview_options` (`aiogram.methods.edit_message_text.EditMessageText` `ampu6ym`), 408
`last_name` (`aiogram.types.inline_query_result_contact.InlineQueryResultContact` `ampu6ym`), 263
`ampu6ym`), 263 `link_preview_options` (`aiogram.methods.send_message.SendMessage` `ampu6ym`), 408
`last_name` (`aiogram.types.input_contact_message_content.InputContactMessageContent` `ampu6ym`), 282
`ampu6ym`), 282 `link_preview_options` (`aiogram.types.external_reply_info.ExternalReplyInfo` `ampu6ym`), 136
`last_name` (`aiogram.types.shared_user.SharedUser` `ampu6ym`), 228
`ampu6ym`), 228 `link_preview_options` (`aiogram.types.input_text_message_content.InputTextMessageContent` `ampu6ym`), 287
`last_name` (`aiogram.types.user.User` `ampu6ym`), 231
`ampu6ym`), 231 `link_preview_options` (`aiogram.types.message.Message` `ampu6ym`), 167
`last_name` (`aiogram.utils.web_app.WebAppUser` `ampu6ym`), 596
`ampu6ym`), 596 `link_preview_options` (`aiogram.types.video_note.VideoNote` `ampu6ym`), 237
`last_synchronization_error_date` (`aiogram.types.webhook_info.WebhookInfo` `ampu6ym`), 313
`ampu6ym`), 313 `link_preview_options` (`aiogram.types.chat_full_info.ChatFullInfo` `ampu6ym`), 62
`latitude` (`aiogram.methods.edit_message_live_location.EditMessageLiveLocation` `ampu6ym`), 456
`ampu6ym`), 456 `linked_chat_id` (`aiogram.types.chat.Chat` `ampu6ym`), 39
`latitude` (`aiogram.methods.send_location.SendLocation` `ampu6ym`), 403
`ampu6ym`), 403 `linked_chat_id` (`aiogram.types.chat_full_info.ChatFullInfo` `ampu6ym`), 62
`latitude` (`aiogram.methods.send_venue.SendVenue` `ampu6ym`), 417
`ampu6ym`), 417 `link_preview_options` (клас в `aiogram.types.link_preview_options`), 158
`latitude` (`aiogram.types.inline_query_result_location.InlineQueryResultLocation` `ampu6ym`), 271
`ampu6ym`), 271 `live_order_id` (`aiogram.methods.edit_message_live_location.EditMessageLiveLocation` `ampu6ym`), 456
`latitude` (`aiogram.types.inline_query_result_venue.InlineQueryResultVenue` `ampu6ym`), 276
`ampu6ym`), 276 `live_order_id` (`aiogram.methods.send_location.SendLocation` `ampu6ym`), 403
`latitude` (`aiogram.types.input_location_message_content.InputLocationMessageContent` `ampu6ym`), 282
`ampu6ym`), 282 `live_order_id` (`aiogram.methods.send_location.SendLocation` `ampu6ym`), 403

<code>ampubym)</code> , 403	<code>live_period(aiogram.types.inline_query_result_location_location_type.LocationType ampubym)</code> , 271	<code>live_period(aiogram.types.input_location_message_content.InputLocationMessageContent ampubym)</code> , 286	<code>live_period(aiogram.types.location.Location ampubym)</code> , 159	<code>LKR(aiogram.enums.currency.Currency ampubym)</code> , 494	<code>LOCATION(aiogram.enums.content_type.ContentType ampubym)</code> , 492	<code>LOCATION(aiogram.enums.inline_query_result_type.InlineQueryResultType ampubym)</code> , 497	<code>location(aiogram.types.business_location.BusinessLocation ampubym)</code> , 33	<code>location(aiogram.types.chat.Chat ampubym)</code> , 39	<code>location(aiogram.types.chat_full_info.ChatFullInfo ampubym)</code> , 62	<code>location(aiogram.types.chat_location.ChatLocation ampubym)</code> , 103	<code>location(aiogram.types.chosen_inline_result.ChosenInlineResult ampubym)</code> , 240	<code>location(aiogram.types.external_reply_info.ExternalReplyInfo ampubym)</code> , 137	<code>location(aiogram.types.inline_query.InlineQuery ampubym)</code> , 241	<code>location(aiogram.types.message.Message ampubym)</code> , 167	<code>location(aiogram.types.venue.Venue ampubym)</code> , 234	<code>Location(клас в aiogram.types.location)</code> , 159	<code>login_url(aiogram.types.inline_keyboard_button.InlineKeyboardButton ampubym)</code> , 144	<code>LoginUrl(клас в aiogram.types.login_url)</code> , 160	<code>LogOut(клас в aiogram.methods.log_out)</code> , 378	<code>longitude(aiogram.methods.edit_message_live_location.EditMessageLiveLocation ampubym)</code> , 456	<code>longitude(aiogram.methods.send_location.SendLocation ampubym)</code> , 403	<code>longitude(aiogram.methods.send_venue.SendVenue ampubym)</code> , 417	<code>longitude(aiogram.types.inline_query_result_location.InlineQueryResultLocation ampubym)</code> , 271	<code>longitude(aiogram.types.inline_query_result_venue.InlineQueryResultVenue ampubym)</code> , 276	<code>longitude(aiogram.types.input_location_message_content.InputLocationMessageContent ampubym)</code> , 285	<code>longitude(aiogram.types.input_venue_message_content.InputVenueMessageContent ampubym)</code> , 288	<code>longitude(aiogram.types.location.Location ampubym)</code> , 159
<code>magic_data.MagicData(клас в aiogram.filters.magic_data)</code> , 524	<code>make_request(aiogram.client.session.base.BaseSession метод)</code> , 15	<code>MARKDOWN(aiogram.enums.parse_mode.ParseMode ampubym)</code> , 500	<code>MARKDOWN_VES(aiogram.enums.parse_mode.ParseMode ampubym)</code> , 500	<code>MASK(aiogram.enums.sticker_type.StickerType ampubym)</code> , 502	<code>mask_position(aiogram.methods.set_sticker_mask_position.SetStickerMaskPosition(клас в aiogram.types.mask_position))</code> , 290	<code>mask_position(aiogram.types.input_sticker.InputSticker ampubym)</code> , 289	<code>mask_position(aiogram.types.sticker.Sticker ampubym)</code> , 291	<code>MaskPosition(клас в aiogram.types.mask_position)</code> , 290	<code>MaskPositionPoint(клас в aiogram.enums.mask_position_point)</code> , 498	<code>max_connections(aiogram.methods.set_webhook.SetWebhook ampubym)</code> , 486	<code>max_connections(aiogram.types.webhook_info.WebhookInfo ampubym)</code> , 313	<code>max_quantity(aiogram.types.keyboard_button_request_users.KeyboardButtonRequestUsers ampubym)</code> , 158	<code>max_reaction_count(aiogram.types.chat_full_info.ChatFullInfo ampubym)</code> , 60	<code>max_tip_amount(aiogram.methods.create_invoice_link.CreateInvoiceLink ampubym)</code> , 477	<code>max_tip_amount(aiogram.methods.send_invoice.SendInvoice ampubym)</code> , 480	<code>max_tip_amount(aiogram.types.input_invoice_message_content.InputInvoiceMessageContent ampubym)</code> , 160	<code>MaybeInaccessibleMessage(клас в aiogram.types.message.MaybeInaccessibleMessage)</code> , 160	<code>md_text(aiogram.types.message.Message property)</code> , 170	<code>MDL(aiogram.enums.currency.Currency ampubym)</code> , 495	<code>media(aiogram.methods.edit_message_media.EditMessageMedia ampubym)</code> , 160							

M

M

`ampubym`), 458
`media` (`aiogram.methods.send_media_group.SendMediaGroup` `ampubym`), 502
`ampubym`), 406
`media` (`aiogram.types.input_media_animation.InputMediaAnimation` `ampubym`), 147
`media` (`aiogram.types.input_media_audio.InputMediaAudio` `ampubym`), 32
`ampubym`), 148
`media` (`aiogram.types.input_media_document.InputMediaDocument` `ampubym`), 35
`ampubym`), 150
`media` (`aiogram.types.input_media_photo.InputMediaPhoto` `ampubym`), 297
`ampubym`), 151
`media` (`aiogram.types.input_media_video.InputMediaVideo` `ampubym`), 297
`ampubym`), 151
`media_group_id` (`aiogram.types.message.Message` `ampubym`), 166
`MediaGroupBuilder` (клас в `aiogram.utils.media_group`), 608
`MEMBER` (`aiogram.enums.chat_member_status.ChatMemberStatus` `ampubym`), 490
`member_limit` (`aiogram.methods.create_chat_invite_link.CreateChatInviteLink` `ampubym`), 346
`member_limit` (`aiogram.methods.edit_chat_invite_link.EditChatInviteLink` `ampubym`), 354
`member_limit` (`aiogram.types.chat_invite_link.ChatInviteLink` `ampubym`), 63
`member_status_changed` (`aiogram.filters.chat_member_updated.ChatMemberUpdated` `ampubym`), 520
`MemoryStorage` (клас в `aiogram.fsm.storage.memory`), 548
`MENTION` (`aiogram.enums.message_entity_type.MessageEntityType` `ampubym`), 499
`mention` (`aiogram.filters.command.CommandObject` `ampubym`), 519
`mention_html()` (`aiogram.types.user.User` `метод`), 232
`mention_markdown()` (`aiogram.types.user.User` `метод`), 232
`mentioned` (`aiogram.filters.command.CommandObject` `property`), 519
`menu_button` (`aiogram.methods.set_chat_menu_button.SetChatMenuButton` `ampubym`), 431
`MenuButton` (клас в `aiogram.types.menu_button`), 161
`MenuButtonCommands` (клас в `aiogram.types.menu_button_commands`), 161
`MenuButtonDefault` (клас в `aiogram.types.menu_button_default`), 162
`MenuButtonType` (клас в `aiogram.enums.menu_button_type`), 499
`MenuButtonWebApp` (клас в `aiogram.types.menu_button_web_app`), 162
`MESSAGE` (`aiogram.enums.update_type.UpdateType` `ampubym`), 578
`message` (`aiogram.handlers.callback_query.CallbackQueryHandler` `ampubym`), 578
`message` (`aiogram.types.business_intro.BusinessIntro` `ampubym`), 32
`message` (`aiogram.types.callback_query.CallbackQuery` `ampubym`), 35
`message` (`aiogram.types.passport_element_error_data_field.PassportElementErrorDataField` `ampubym`), 297
`message` (`aiogram.types.passport_element_error_file.PassportElementErrorFile` `ampubym`), 297
`message` (`aiogram.types.passport_element_error_files.PassportElementErrorFiles` `ampubym`), 298
`message` (`aiogram.types.passport_element_error_front_side.PassportElementErrorFrontSide` `ampubym`), 299
`message` (`aiogram.types.passport_element_error_reverse_side.PassportElementErrorReverseSide` `ampubym`), 300
`message` (`aiogram.types.passport_element_error_selfie.PassportElementErrorSelfie` `ampubym`), 300
`message` (`aiogram.types.passport_element_error_translation_file.PassportElementErrorTranslationFile` `ampubym`), 301
`message` (`aiogram.types.passport_element_error_translation_files.PassportElementErrorTranslationFiles` `ampubym`), 302
`message` (`aiogram.types.passport_element_error_unspecified.PassportElementErrorUnspecified` `ampubym`), 303
`message` (`aiogram.types.update.Update` `ampubym`), 310
`Message` (клас в `aiogram.types.message`), 163
`message_auto_delete_time` (`aiogram.types.chat.Chat` `ampubym`), 40
`message_auto_delete_time` (`aiogram.types.chat_full_info.ChatFullInfo` `ampubym`), 61
`message_auto_delete_time` (`aiogram.types.message_auto_delete_timer_changed.MessageAutoDeleteTimerChanged` `ampubym`), 214
`MESSAGE_AUTO_DELETE_TIMER_CHANGED` (`aiogram.enums.content_type.ContentType` `ampubym`), 492
`message_auto_delete_timer_changed` (`aiogram.types.message.Message` `ampubym`), 168
`message_id` (`aiogram.methods.copy_message.CopyMessage` `ampubym`), 342
`message_id` (`aiogram.methods.delete_message.DeleteMessage` `ampubym`), 452
`message_id` (`aiogram.methods.edit_message_caption.EditMessageCaption` `ampubym`), 454
`message_id` (`aiogram.methods.edit_message_live_location.EditMessageLiveLocation` `ampubym`), 456
`message_id` (`aiogram.methods.edit_message_media.EditMessageMedia` `ampubym`), 458
`message_id` (`aiogram.methods.edit_message_reply_markup.EditMessageReplyMarkup` `ampubym`), 458

<code>message_id</code> (<code>aiogram.methods.edit_message_text.EditMessageText</code>), 287	<code>message_text</code> (<code>aiogram.types.input_text_message_content.InputTextMessageContent</code>), 287
<code>message_id</code> (<code>aiogram.methods.forward_message.ForwardMessage</code>), 340	<code>message_thread_id</code> (<code>aiogram.methods.close_forum_topic.CloseForumTopic</code>), 340
<code>message_id</code> (<code>aiogram.methods.get_game_high_scores.GetGameHighScores</code>), 342	<code>message_thread_id</code> (<code>aiogram.methods.copy_message.CopyMessage</code>), 342
<code>message_id</code> (<code>aiogram.methods.pin_chat_message.PinChatMessage</code>), 342	<code>message_thread_id</code> (<code>aiogram.methods.copy_messages.CopyMessages</code>), 345
<code>message_id</code> (<code>aiogram.methods.set_game_score.SetGameScore</code>), 345	<code>message_thread_id</code> (<code>aiogram.methods.delete_forum_topic.DeleteForumTopic</code>), 350
<code>message_id</code> (<code>aiogram.methods.set_message_reaction.SetMessageReaction</code>), 347	<code>message_thread_id</code> (<code>aiogram.methods.edit_forum_topic.EditForumTopic</code>), 355
<code>message_id</code> (<code>aiogram.methods.stop_message_live_location.StopMessageLiveLocation</code>), 350	<code>message_thread_id</code> (<code>aiogram.methods.forward_message.ForwardMessage</code>), 359
<code>message_id</code> (<code>aiogram.methods.stop_poll.StopPoll</code>), 355	<code>message_thread_id</code> (<code>aiogram.methods.forward_messages.ForwardMessages</code>), 360
<code>message_id</code> (<code>aiogram.methods.unpin_chat_message.UnpinChatMessage</code>), 359	<code>message_thread_id</code> (<code>aiogram.methods.reopen_forum_topic.ReopenForumTopic</code>), 383
<code>message_id</code> (<code>aiogram.types.external_reply_info.ExternalReplyInfo</code>), 359	<code>message_thread_id</code> (<code>aiogram.methods.send_animation.SendAnimation</code>), 389
<code>message_id</code> (<code>aiogram.types.inaccessible_message.InaccessibleMessage</code>), 360	<code>message_thread_id</code> (<code>aiogram.methods.send_chat_action.SendChatAction</code>), 394
<code>message_id</code> (<code>aiogram.types.message.Message</code>), 365	<code>message_thread_id</code> (<code>aiogram.methods.send_contact.SendContact</code>), 396
<code>message_id</code> (<code>aiogram.types.message_id.MessageId</code>), 365	<code>message_thread_id</code> (<code>aiogram.methods.send_dice.SendDice</code>), 398
<code>message_id</code> (<code>aiogram.types.message_origin_channel.MessageOriginChannel</code>), 365	<code>message_thread_id</code> (<code>aiogram.methods.send_document.SendDocument</code>), 400
<code>message_id</code> (<code>aiogram.types.message_reaction_count.MessageReactionCount</code>), 365	<code>message_thread_id</code> (<code>aiogram.methods.send_game.SendGame</code>), 471
<code>message_id</code> (<code>aiogram.types.message_reaction_updated.MessageReactionUpdated</code>), 365	<code>message_thread_id</code> (<code>aiogram.methods.send_invoice.SendInvoice</code>), 480
<code>message_id</code> (<code>aiogram.types.reply_parameters.ReplyParameters</code>), 365	<code>message_thread_id</code> (<code>aiogram.methods.send_location.SendLocation</code>), 403
<code>message_ids</code> (<code>aiogram.methods.copy_messages.CopyMessages</code>), 344	<code>message_thread_id</code> (<code>aiogram.methods.send_message.SendMessage</code>), 392
<code>message_ids</code> (<code>aiogram.methods.delete_messages.DeleteMessages</code>), 396	<code>message_thread_id</code> (<code>aiogram.methods.send_poll.SendPoll</code>), 392
<code>message_ids</code> (<code>aiogram.methods.forward_messages.ForwardMessages</code>), 360	<code>message_thread_id</code> (<code>aiogram.methods.send_poll_answer.SendPollAnswer</code>), 392
<code>message_ids</code> (<code>aiogram.types.business_messages_deleted.BusinessMessagesDeleted</code>), 33	<code>message_thread_id</code> (<code>aiogram.methods.send_voice.SendVoice</code>), 392
<code>MESSAGE_REACTION</code> (<code>aiogram.enums.update_type.UpdateType</code>), 503	<code>message_thread_id</code> (<code>aiogram.methods.send_voice_chat.SendVoiceChat</code>), 392
<code>message_reaction</code> (<code>aiogram.types.update.Update</code>), 311	<code>message_thread_id</code> (<code>aiogram.methods.send_voice_chat.SendVoiceChat</code>), 392
<code>MESSAGE_REACTION_COUNT</code> (<code>aiogram.enums.update_type.UpdateType</code>), 503	<code>message_thread_id</code> (<code>aiogram.methods.send_voice_chat.SendVoiceChat</code>), 392
<code>message_reaction_count</code> (<code>aiogram.types.update.Update</code>), 311	<code>message_thread_id</code> (<code>aiogram.methods.send_voice_chat.SendVoiceChat</code>), 392

<code>aiogram.methods.send_media_group.SendMediaGroup</code> (клас в <code>aiogram.methods</code>), 406	<code>MessageOriginType</code> (клас в <code>aiogram.enums.message_origin_type</code>), 500
<code>message_thread_id</code> (aiogram.methods.send_message.SendMessage <code>aiogram.methods</code>), 408	<code>MessageOriginUser</code> (клас в <code>aiogram.types.message_origin_user</code>), 218
<code>message_thread_id</code> (aiogram.methods.send_photo.SendPhoto <code>aiogram.methods</code>), 411	<code>MessageReactionCountUpdated</code> (клас в <code>aiogram.types.message_reaction_count_updated</code>), 219
<code>message_thread_id</code> (aiogram.methods.send_poll.SendPoll <code>aiogram.methods</code>), 414	<code>MessageReactionUpdated</code> (клас в <code>aiogram.types.message_reaction_updated</code>), 219
<code>message_thread_id</code> (aiogram.methods.send_sticker.SendSticker <code>aiogram.methods</code>), 323	<code>MIGRATE_FROM_CHAT_ID</code> (aiogram.enums.content_type.ContentType <code>aiogram.enums</code>), 492
<code>message_thread_id</code> (aiogram.methods.send_venue.SendVenue <code>aiogram.methods</code>), 417	<code>migrate_from_chat_id</code> (aiogram.types.message.Message <code>aiogram.types</code>), 168
<code>message_thread_id</code> (aiogram.methods.send_video.SendVideo <code>aiogram.methods</code>), 420	<code>MIGRATE_TO_CHAT_ID</code> (aiogram.enums.content_type.ContentType <code>aiogram.enums</code>), 492
<code>message_thread_id</code> (aiogram.methods.send_video_note.SendVideoNote <code>aiogram.methods</code>), 423	<code>migrate_to_chat_id</code> (aiogram.types.message.Message <code>aiogram.types</code>), 168
<code>message_thread_id</code> (aiogram.methods.send_voice.SendVoice <code>aiogram.methods</code>), 426	<code>migrate_to_chat_id</code> (aiogram.types.response_parameters.ResponseParameters <code>aiogram.types</code>), 228
<code>message_thread_id</code> (aiogram.methods.unpin_all_forum_topic_messages.UnpinAllForumTopicMessages <code>aiogram.methods</code>), 448	<code>mime_type</code> (aiogram.types.animation.Animation <code>aiogram.types</code>), 20
<code>message_thread_id</code> (aiogram.types.forum_topic.ForumTopic <code>aiogram.types</code>), 138	<code>mime_type</code> (aiogram.types.audio.Audio <code>aiogram.types</code>), 135
<code>message_thread_id</code> (aiogram.types.message.Message <code>aiogram.types</code>), 165	<code>mime_type</code> (aiogram.types.document.Document <code>aiogram.types</code>), 266
<code>MessageAutoDeleteTimerChanged</code> (клас в <code>aiogram.types.message_auto_delete_timer_changed</code>), 214	<code>mime_type</code> (aiogram.types.inline_query_result_document.InlineQueryResultDocument <code>aiogram.types</code>), 279
<code>MessageEntity</code> (клас в <code>aiogram.types.message_entity</code>), 215	<code>mime_type</code> (aiogram.types.inline_query_result_video.InlineQueryResultVideo <code>aiogram.types</code>), 235
<code>MessageEntityType</code> (клас в <code>aiogram.enums.message_entity_type</code>), 499	<code>mime_type</code> (aiogram.types.video.Video <code>aiogram.types</code>), 238
<code>MessageId</code> (клас в <code>aiogram.types.message_id</code>), 216	<code>MNT</code> (aiogram.enums.currency.Currency <code>aiogram.enums</code>), 495
<code>MessageOrigin</code> (клас в <code>aiogram.types.message_origin</code>), 216	<code>model_computed_fields</code> (aiogram.filters.callback_data.CallbackData <code>aiogram.filters</code>), 525
<code>MessageOriginChannel</code> (клас в <code>aiogram.types.message_origin_channel</code>), 216	<code>model_computed_fields</code> (aiogram.methods.add_sticker_to_set.AddStickerToSet <code>aiogram.methods</code>), 315
<code>MessageOriginChat</code> (клас в <code>aiogram.types.message_origin_chat</code>), 217	<code>model_computed_fields</code> (aiogram.methods.answer_callback_query.AnswerCallbackQuery <code>aiogram.methods</code>), 334
<code>MessageOriginHiddenUser</code> (клас в <code>aiogram.types.message_origin_hidden_user</code>), 218	<code>model_computed_fields</code> (aiogram.methods.answer_inline_query.AnswerInlineQuery <code>aiogram.methods</code>), 467

<code>model_computed_fields</code> <code>ogram.methods.answer_pre_checkout_query.AnswerPreCheckoutQuery</code> <code>ampubym</code>), 474	<code>(ai- model_computed_fields</code> <code>ogram.methods.delete_forum_topic.DeleteForumTopic</code> <code>ampubym</code>), 352	<code>(ai-</code>
<code>model_computed_fields</code> <code>ogram.methods.answer_shipping_query.AnswerShippingQuery</code> <code>ampubym</code>), 475	<code>ogram.methods.delete_message.DeleteMessage</code> <code>ampubym</code>), 452	<code>(ai-</code>
<code>model_computed_fields</code> <code>ogram.methods.answer_web_app_query.AnswerWebAppQuery</code> <code>ampubym</code>), 468	<code>ogram.methods.delete_messages.DeleteMessages</code> <code>ampubym</code>), 453	<code>(ai-</code>
<code>model_computed_fields</code> <code>ogram.methods.approve_chat_join_request.ApproveChatJoinRequest</code> <code>ampubym</code>), 335	<code>ogram.methods.delete_my_commands.DeleteMyCommands</code> <code>ampubym</code>), 353	<code>(ai-</code>
<code>model_computed_fields</code> <code>ogram.methods.ban_chat_member.BanChatMember</code> <code>ampubym</code>), 336	<code>ogram.methods.delete_sticker_from_set.DeleteStickerFromSet</code> <code>ampubym</code>), 317	<code>(ai-</code>
<code>model_computed_fields</code> <code>ogram.methods.ban_chat_sender_chat.BanChatSenderChat</code> <code>ampubym</code>), 338	<code>ogram.methods.delete_sticker_set.DeleteStickerSet</code> <code>ampubym</code>), 318	<code>(ai-</code>
<code>model_computed_fields</code> <code>ogram.methods.close.Close</code> <code>ampubym</code>), 339	<code>ogram.methods.delete_webhook.DeleteWebhook</code> <code>ampubym</code>), 482	<code>(ai-</code>
<code>model_computed_fields</code> <code>ogram.methods.close_forum_topic.CloseForumTopic</code> <code>ampubym</code>), 340	<code>ogram.methods.edit_chat_invite_link.EditChatInviteLink</code> <code>ampubym</code>), 354	<code>(ai-</code>
<code>model_computed_fields</code> <code>ogram.methods.close_general_forum_topic.CloseGeneralForumTopic</code> <code>ampubym</code>), 341	<code>ogram.methods.edit_forum_topic.EditForumTopic</code> <code>ampubym</code>), 355	<code>(ai-</code>
<code>model_computed_fields</code> <code>ogram.methods.copy_message.CopyMessage</code> <code>ampubym</code>), 343	<code>ogram.methods.edit_general_forum_topic.EditGeneralForumTopic</code> <code>ampubym</code>), 356	<code>(ai-</code>
<code>model_computed_fields</code> <code>ogram.methods.copy_messages.CopyMessages</code> <code>ampubym</code>), 345	<code>ogram.methods.edit_message_caption.EditMessageCaption</code> <code>ampubym</code>), 454	<code>(ai-</code>
<code>model_computed_fields</code> <code>ogram.methods.create_chat_invite_link.CreateChatInviteLink</code> <code>ampubym</code>), 346	<code>ogram.methods.edit_message_live_location.EditMessageLiveLocation</code> <code>ampubym</code>), 456	<code>(ai-</code>
<code>model_computed_fields</code> <code>ogram.methods.create_forum_topic.CreateForumTopic</code> <code>ampubym</code>), 347	<code>ogram.methods.edit_message_media.EditMessageMedia</code> <code>ampubym</code>), 458	<code>(ai-</code>
<code>model_computed_fields</code> <code>ogram.methods.create_invoice_link.CreateInvoiceLink</code> <code>ampubym</code>), 478	<code>ogram.methods.edit_message_reply_markup.EditMessageReplyMarkup</code> <code>ampubym</code>), 459	<code>(ai-</code>
<code>model_computed_fields</code> <code>ogram.methods.create_new_sticker_set.CreateNewStickerSet</code> <code>ampubym</code>), 316	<code>ogram.methods.edit_message_text.EditMessageText</code> <code>ampubym</code>), 461	<code>(ai-</code>
<code>model_computed_fields</code> <code>ogram.methods.decline_chat_join_request.DeclineChatJoinRequest</code> <code>ampubym</code>), 348	<code>ogram.methods.export_chat_invite_link.ExportChatInviteLink</code> <code>ampubym</code>), 358	<code>(ai-</code>
<code>model_computed_fields</code> <code>ogram.methods.delete_chat_photo.DeleteChatPhoto</code> <code>ampubym</code>), 349	<code>ogram.methods.forward_message.ForwardMessage</code> <code>ampubym</code>), 359	<code>(ai-</code>
<code>model_computed_fields</code> <code>ogram.methods.delete_chat_sticker_set.DeleteChatStickerSet</code> <code>ampubym</code>), 350	<code>ogram.methods.forward_messages.ForwardMessages</code> <code>ampubym</code>), 360	<code>(ai-</code>

model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_business_connection.GetBusinessConnection	ogram.methods.get_user_chat_boosts.GetUserChatBoosts	
ampubym), 361	ampubym), 375	
model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_chat.GetChat	ogram.methods.get_user_profile_photos.GetUserProfilePhotos	
ampubym), 362	ampubym), 375	
model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_chat_administrators.GetChatAdministrators	ogram.methods.get_webhook_info.GetWebhookInfo	
ampubym), 363	ampubym), 485	
model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_chat_member.GetChatMember	ogram.methods.hide_general_forum_topic.HideGeneralForumTopic	
ampubym), 364	ampubym), 376	
model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_chat_member_count.GetChatMemberCount	ogram.methods.leave_chat.LeaveChat	
ampubym), 365	ampubym), 377	
model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_chat_menu_button.GetChatMenuButton	ogram.methods.log_out.LogOut	
ampubym), 366	ampubym), 378	
model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_custom_emoji_stickers.GetCustomEmojiStickers	ogram.methods.pin_chat_message.PinChatMessage	
ampubym), 319	ampubym), 379	
model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_file.GetFile	ogram.methods.promote_chat_member.PromoteChatMember	
ampubym), 367	ampubym), 382	
model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_forum_topic_icon_stickers.GetForumTopicIconStickers	ogram.methods.reopen_forum_topic.ReopenForumTopic	
ampubym), 368	ampubym), 383	
model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_game_high_scores.GetGameHighScores	ogram.methods.reopen_general_forum_topic.ReopenGeneralForumTopic	
ampubym), 470	ampubym), 384	
model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_me.GetMe	ogram.methods.replace_sticker_in_set.ReplaceStickerInSet	
ampubym), 369	ampubym), 321	
model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_my_commands.GetMyCommands	ogram.methods.restrict_chat_member.RestrictChatMember	
ampubym), 370	ampubym), 385	
model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_my_default_administrator_rights.GetMyDefaultAdministratorRights	ogram.methods.revoke_chat_invite_link.RevokeChatInviteLink	
ampubym), 371	ampubym), 387	
model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_my_description.GetMyDescription	ogram.methods.send_animation.SendAnimation	
ampubym), 372	ampubym), 389	
model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_my_name.GetMyName	ogram.methods.send_audio.SendAudio	
ampubym), 373	ampubym), 392	
model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_my_short_description.GetMyShortDescription	ogram.methods.send_chat_action.SendChatAction	
ampubym), 374	ampubym), 394	
model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_sticker_set.GetStickerSet	ogram.methods.send_contact.SendContact	
ampubym), 320	ampubym), 396	
model_computed_fields	(ai- model_computed_fields	(ai-
ogram.methods.get_updates.GetUpdates	ogram.methods.send_dice.SendDice	
ampubym), 484	ampubym), 398	

model_computed_fields ogram.methods.send_document.SendDocument ampubym), 401	(ai- model_computed_fields ogram.methods.set_chat_sticker_set.SetChatStickerSet ampubym), 434	(ai-
model_computed_fields ogram.methods.send_game.SendGame ampubym), 471	(ai- model_computed_fields ogram.methods.set_chat_title.SetChatTitle ampubym), 435	(ai-
model_computed_fields ogram.methods.send_invoice.SendInvoice ampubym), 480	(ai- model_computed_fields ogram.methods.set_custom_emoji_sticker_set_thumbnail.S ampubym), 325	(ai-
model_computed_fields ogram.methods.send_location.SendLocation ampubym), 403	(ai- model_computed_fields ogram.methods.set_game_score.SetGameScore ampubym), 473	(ai-
model_computed_fields ogram.methods.send_media_group.SendMediaGroup ampubym), 406	(ai- model_computed_fields ogram.methods.set_message_reaction.SetMessageReaction ampubym), 437	(ai-
model_computed_fields ogram.methods.send_message.SendMessage ampubym), 409	(ai- model_computed_fields ogram.methods.set_my_commands.SetMyCommands ampubym), 438	(ai-
model_computed_fields ogram.methods.send_photo.SendPhoto ampubym), 412	(ai- model_computed_fields ogram.methods.set_my_default_administrator_rights.SetM ampubym), 439	(ai-
model_computed_fields ogram.methods.send_poll.SendPoll ampubym), 415	(ai- model_computed_fields ogram.methods.set_my_description.SetMyDescription ampubym), 440	(ai-
model_computed_fields ogram.methods.send_sticker.SendSticker ampubym), 323	(ai- model_computed_fields ogram.methods.set_my_name.SetMyName ampubym), 441	(ai-
model_computed_fields ogram.methods.send_venue.SendVenue ampubym), 418	(ai- model_computed_fields ogram.methods.set_my_short_description.SetMyShortDescr ampubym), 442	(ai-
model_computed_fields ogram.methods.send_video.SendVideo ampubym), 421	(ai- model_computed_fields ogram.methods.set_passport_data_errors.SetPassportDataE ampubym), 488	(ai-
model_computed_fields ogram.methods.send_video_note.SendVideoNote ampubym), 424	(ai- model_computed_fields ogram.methods.set_sticker_emoji_list.SetStickerEmojiList ampubym), 326	(ai-
model_computed_fields ogram.methods.send_voice.SendVoice ampubym), 427	(ai- model_computed_fields ogram.methods.set_sticker_keywords.SetStickerKeywords ampubym), 327	(ai-
model_computed_fields ogram.methods.set_chat_administrator_custom_title.SetChatAdminis ampubym), 428	(ai- model_computed_fields ogram.methods.set_sticker_mask_position.SetStickerMaskP ampubym), 328	(ai-
model_computed_fields ogram.methods.set_chat_description.SetChatDescription ampubym), 429	(ai- model_computed_fields ogram.methods.set_sticker_position_in_set.SetStickerPosit ampubym), 329	(ai-
model_computed_fields ogram.methods.set_chat_menu_button.SetChatMenuButton ampubym), 431	(ai- model_computed_fields ogram.methods.set_sticker_set_thumbnail.SetStickerSetThu ampubym), 330	(ai-
model_computed_fields ogram.methods.set_chat_permissions.SetChatPermissions ampubym), 432	(ai- model_computed_fields ogram.methods.set_sticker_set_title.SetStickerSetTitle ampubym), 332	(ai-
model_computed_fields ogram.methods.set_chat_photo.SetChatPhoto ampubym), 433	(ai- model_computed_fields ogram.methods.set_webhook.SetWebhook ampubym), 486	(ai-

model_computed_fields	(aiogram.methods.stop_message_live_location.StopMessageLiveLocationTypeFill), 463	(aiogram.types.background_type_fill.BackgroundTypeFill), 24
model_computed_fields	(aiogram.methods.stop_poll.StopPollTypePattern), 464	(aiogram.types.background_type_pattern.BackgroundTypePattern), 24
model_computed_fields	(aiogram.methods.unban_chat_member.UnbanChatMemberTypeWallpaper), 444	(aiogram.types.background_type_wallpaper.BackgroundTypeWallpaper), 25
model_computed_fields	(aiogram.methods.unban_chat_sender_chat.UnbanChatSenderChatTypeBirthdate), 445	(aiogram.types.birthdate.Birthdate), 25
model_computed_fields	(aiogram.methods.unhide_general_forum_topic.UnhideGeneralForumTopicTypeBotCommand), 446	(aiogram.types.bot_command.BotCommand), 26
model_computed_fields	(aiogram.methods.unpin_all_chat_messages.UnpinAllChatMessagesTypeBotCommandScope), 447	(aiogram.types.bot_command_scope.BotCommandScope), 26
model_computed_fields	(aiogram.methods.unpin_all_forum_topic_messages.UnpinAllForumTopicMessagesTypeScopeAllChatAdministrators), 448	(aiogram.types.scope_all_chat_administrators.ScopeAllChatAdministrators), 27
model_computed_fields	(aiogram.methods.unpin_all_general_forum_topic_messages.UnpinAllGeneralForumTopicMessagesTypeScopePrivateChats), 449	(aiogram.types.scope_private_chats.ScopePrivateChats), 27
model_computed_fields	(aiogram.methods.unpin_chat_message.UnpinChatMessageTypeBotCommandScopeAllPrivateChats), 450	(aiogram.types.bot_command_scope_all_private_chats.BotCommandScopeAllPrivateChats), 28
model_computed_fields	(aiogram.methods.upload_sticker_file.UploadStickerFileTypeBotCommandScopeChat), 333	(aiogram.types.bot_command_scope_chat.BotCommandScopeChat), 28
model_computed_fields	(aiogram.types.animation.AnimationTypeBotCommandScopeChatAdministrators), 19	(aiogram.types.bot_command_scope_chat_administrators.BotCommandScopeChatAdministrators), 29
model_computed_fields	(aiogram.types.audio.AudioTypeBotCommandScopeChatMember), 20	(aiogram.types.bot_command_scope_chat_member.BotCommandScopeChatMember), 29
model_computed_fields	(aiogram.types.background_fill.BackgroundFillTypeBotCommandScopeDefault), 21	(aiogram.types.bot_command_scope_default.BotCommandScopeDefault), 30
model_computed_fields	(aiogram.types.background_fill_freeform_gradient.BackgroundFillFreeformGradientTypeBotDescription), 21	(aiogram.types.bot_description.BotDescription), 30
model_computed_fields	(aiogram.types.background_fill_gradient.GradientTypeBotName), 22	(aiogram.types.bot_name.BotName), 31
model_computed_fields	(aiogram.types.background_fill_solid.BackgroundFillSolidTypeBotShortDescription), 22	(aiogram.types.bot_short_description.BotShortDescription), 31
model_computed_fields	(aiogram.types.background_type.BackgroundTypeTypeBusinessConnection), 23	(aiogram.types.business_connection.BusinessConnection), 31
model_computed_fields	(aiogram.types.background_type_chat_theme.BackgroundTypeChatThemeTypeBusinessIntro), 23	(aiogram.types.business_intro.BusinessIntro), 32

model_computed_fields	(aiogram.types.business_location.BusinessLocation	aiogram.types.chat_invite_link.ChatInviteLink
ampubym), 32	ampubym), 63	
model_computed_fields	(aiogram.types.business_messages_deleted.BusinessMessagesDeleted	aiogram.types.chat_join_request.ChatJoinRequest
ampubym), 33	ampubym), 97	
model_computed_fields	(aiogram.types.business_opening_hours.BusinessOpeningHours	aiogram.types.chat_location.ChatLocation
ampubym), 33	ampubym), 103	
model_computed_fields	(aiogram.types.business_opening_hours_interval.BusinessOpeningHoursInterval	aiogram.types.chat_member.ChatMember
ampubym), 34	ampubym), 104	
model_computed_fields	(aiogram.types.callback_game.CallbackGame	aiogram.types.chat_member_administrator.ChatMemberAdministrator
ampubym), 313	ampubym), 106	
model_computed_fields	(aiogram.types.callback_query.CallbackQuery	aiogram.types.chat_member_banned.ChatMemberBanned
ampubym), 35	ampubym), 107	
model_computed_fields	(aiogram.types.chat.Chat	aiogram.types.chat_member_left.ChatMemberLeft
ampubym), 40	ampubym), 107	
model_computed_fields	(aiogram.types.chat_administrator_rights.ChatAdministratorRights	aiogram.types.chat_member_member.ChatMemberMember
ampubym), 53	ampubym), 108	
model_computed_fields	(aiogram.types.chat_background.ChatBackground	aiogram.types.chat_member_owner.ChatMemberOwner
ampubym), 54	ampubym), 108	
model_computed_fields	(aiogram.types.chat_boost.ChatBoost	aiogram.types.chat_member_restricted.ChatMemberRestricted
ampubym), 54	ampubym), 109	
model_computed_fields	(aiogram.types.chat_boost_added.ChatBoostAdded	aiogram.types.chat_member_updated.ChatMemberUpdated
ampubym), 55	ampubym), 122	
model_computed_fields	(aiogram.types.chat_boost_removed.ChatBoostRemoved	aiogram.types.chat_permissions.ChatPermissions
ampubym), 55	ampubym), 131	
model_computed_fields	(aiogram.types.chat_boost_source.ChatBoostSource	aiogram.types.chat_photo.ChatPhoto
ampubym), 56	ampubym), 132	
model_computed_fields	(aiogram.types.chat_boost_source_gift_code.ChatBoostSourceGiftCode	aiogram.types.chat_shared.ChatShared
ampubym), 56	ampubym), 133	
model_computed_fields	(aiogram.types.chat_boost_source_giveaway.ChatBoostSourceGiveaway	aiogram.types.chat_shared_chosen_inline_result.ChosenInlineResult
ampubym), 57	ampubym), 240	
model_computed_fields	(aiogram.types.chat_boost_source_premium.ChatBoostSourcePremium	aiogram.types.contact.Contact
ampubym), 57	ampubym), 133	
model_computed_fields	(aiogram.types.chat_boost_updated.ChatBoostUpdated	aiogram.types.dice.Dice
ampubym), 58	ampubym), 134	
model_computed_fields	(aiogram.types.chat_full_info.ChatFullInfo	aiogram.types.document.Document
ampubym), 61	ampubym), 134	
model_computed_fields	(aiogram.types.encrypted_credentials.EncryptedCredentials	

<code>aiogram.types.encrypted_passport_element.EncryptedPassportElement</code>	(aiogram.types.encrypted_passport_element.EncryptedPassportElement), 294	<code>aiogram.types.inaccessible_message.InaccessibleMessage</code>	(aiogram.types.inaccessible_message.InaccessibleMessage), 144
<code>aiogram.types.error_event.ErrorEvent</code>	(aiogram.types.error_event.ErrorEvent), 573	<code>aiogram.types.inline_keyboard_button.InlineKeyboardButton</code>	(aiogram.types.inline_keyboard_button.InlineKeyboardButton), 145
<code>aiogram.types.external_reply_info.ExternalReplyInfo</code>	(aiogram.types.external_reply_info.ExternalReplyInfo), 136	<code>aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup</code>	(aiogram.types.inline_keyboard_markup.InlineKeyboardMarkup), 145
<code>aiogram.types.file.File</code>	(aiogram.types.file.File), 137	<code>aiogram.types.inline_query.InlineQuery</code>	(aiogram.types.inline_query.InlineQuery), 241
<code>aiogram.types.force_reply.ForceReply</code>	(aiogram.types.force_reply.ForceReply), 138	<code>aiogram.types.inline_query_result.InlineQueryResult</code>	(aiogram.types.inline_query_result.InlineQueryResult), 242
<code>aiogram.types.forum_topic.ForumTopic</code>	(aiogram.types.forum_topic.ForumTopic), 139	<code>aiogram.types.inline_query_result_article.InlineQueryResultArticle</code>	(aiogram.types.inline_query_result_article.InlineQueryResultArticle), 243
<code>aiogram.types.forum_topic_closed.ForumTopicClosed</code>	(aiogram.types.forum_topic_closed.ForumTopicClosed), 139	<code>aiogram.types.inline_query_result_audio.InlineQueryResultAudio</code>	(aiogram.types.inline_query_result_audio.InlineQueryResultAudio), 246
<code>aiogram.types.forum_topic_created.ForumTopicCreated</code>	(aiogram.types.forum_topic_created.ForumTopicCreated), 139	<code>aiogram.types.inline_query_result_cached_audio.InlineQueryResultCachedAudio</code>	(aiogram.types.inline_query_result_cached_audio.InlineQueryResultCachedAudio), 248
<code>aiogram.types.forum_topic_edited.ForumTopicEdited</code>	(aiogram.types.forum_topic_edited.ForumTopicEdited), 140	<code>aiogram.types.inline_query_result_cached_document.InlineQueryResultCachedDocument</code>	(aiogram.types.inline_query_result_cached_document.InlineQueryResultCachedDocument), 250
<code>aiogram.types.forum_topic_reopened.ForumTopicReopened</code>	(aiogram.types.forum_topic_reopened.ForumTopicReopened), 140	<code>aiogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif</code>	(aiogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif), 252
<code>aiogram.types.game.Game</code>	(aiogram.types.game.Game), 313	<code>aiogram.types.inline_query_result_cached_mpeg4_gif.InlineQueryResultCachedMpeg4Gif</code>	(aiogram.types.inline_query_result_cached_mpeg4_gif.InlineQueryResultCachedMpeg4Gif), 254
<code>aiogram.types.game_high_score.GameHighScore</code>	(aiogram.types.game_high_score.GameHighScore), 314	<code>aiogram.types.inline_query_result_cached_photo.InlineQueryResultCachedPhoto</code>	(aiogram.types.inline_query_result_cached_photo.InlineQueryResultCachedPhoto), 256
<code>aiogram.types.general_forum_topic_hidden.GeneralForumTopicHidden</code>	(aiogram.types.general_forum_topic_hidden.GeneralForumTopicHidden), 140	<code>aiogram.types.inline_query_result_cached_sticker.InlineQueryResultCachedSticker</code>	(aiogram.types.inline_query_result_cached_sticker.InlineQueryResultCachedSticker), 257
<code>aiogram.types.general_forum_topic_unhidden.GeneralForumTopicUnhidden</code>	(aiogram.types.general_forum_topic_unhidden.GeneralForumTopicUnhidden), 141	<code>aiogram.types.inline_query_result_cached_video.InlineQueryResultCachedVideo</code>	(aiogram.types.inline_query_result_cached_video.InlineQueryResultCachedVideo), 260
<code>aiogram.types.giveaway.Giveaway</code>	(aiogram.types.giveaway.Giveaway), 141	<code>aiogram.types.inline_query_result_cached_voice.InlineQueryResultCachedVoice</code>	(aiogram.types.inline_query_result_cached_voice.InlineQueryResultCachedVoice), 262
<code>aiogram.types.giveaway_completed.GiveawayCompleted</code>	(aiogram.types.giveaway_completed.GiveawayCompleted), 142	<code>aiogram.types.inline_query_result_contact.InlineQueryResultContact</code>	(aiogram.types.inline_query_result_contact.InlineQueryResultContact), 264
<code>aiogram.types.giveaway_created.GiveawayCreated</code>	(aiogram.types.giveaway_created.GiveawayCreated), 142	<code>aiogram.types.inline_query_result_document.InlineQueryResultDocument</code>	(aiogram.types.inline_query_result_document.InlineQueryResultDocument), 266
<code>aiogram.types.giveaway_winners.GiveawayWinners</code>	(aiogram.types.giveaway_winners.GiveawayWinners), 143	<code>aiogram.types.inline_query_result_game.InlineQueryResultGame</code>	(aiogram.types.inline_query_result_game.InlineQueryResultGame), 267
<code>aiogram.types.location.Location</code>	(aiogram.types.location.Location), 144		

<code>ogram.types.inline_query_result_gif.InlineQueryResultGif</code>	<code>ogram.types.input_poll_option.InputPollOption</code>
<code>ampubym), 269</code>	<code>ampubym), 153</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>
<code>ogram.types.inline_query_result_location.InlineQueryResultLocation</code>	<code>ogram.types.input_sticker.InputSticker</code>
<code>ampubym), 271</code>	<code>ampubym), 289</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>
<code>ogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif</code>	<code>ogram.types.text_message_content.InputTextMessageContent</code>
<code>ampubym), 273</code>	<code>ampubym), 287</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>
<code>ogram.types.inline_query_result_photo.InlineQueryResultPhoto</code>	<code>ogram.types.input_venue_message_content.InputVenueMessageContent</code>
<code>ampubym), 275</code>	<code>ampubym), 288</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>
<code>ogram.types.inline_query_result_venue.InlineQueryResultVenue</code>	<code>ogram.types.invoice.Invoice</code>
<code>ampubym), 277</code>	<code>ampubym), 304</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>
<code>ogram.types.inline_query_result_video.InlineQueryResultVideo</code>	<code>ogram.types.keyboard_button.KeyboardButton</code>
<code>ampubym), 279</code>	<code>ampubym), 154</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>
<code>ogram.types.inline_query_result_voice.InlineQueryResultVoice</code>	<code>ogram.types.keyboard_button_poll_type.KeyboardButtonPollType</code>
<code>ampubym), 280</code>	<code>ampubym), 154</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>
<code>ogram.types.inline_query_results_button.InlineQueryResultsButton</code>	<code>ogram.types.keyboard_button_request_chat.KeyboardButtonRequestChat</code>
<code>ampubym), 281</code>	<code>ampubym), 156</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>
<code>ogram.types.input_contact_message_content.InputContactMessageContent</code>	<code>ogram.types.keyboard_button_request_user.KeyboardButtonRequestUser</code>
<code>ampubym), 282</code>	<code>ampubym), 157</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>
<code>ogram.types.input_invoice_message_content.InputInvoiceMessageContent</code>	<code>ogram.types.keyboard_button_request_users.KeyboardButtonRequestUsers</code>
<code>ampubym), 284</code>	<code>ampubym), 158</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>
<code>ogram.types.input_location_message_content.InputLocationMessageContent</code>	<code>ogram.types.labeled_price.LabeledPrice</code>
<code>ampubym), 285</code>	<code>ampubym), 304</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>
<code>ogram.types.input_media.InputMedia</code>	<code>ogram.types.link_preview_options.LinkPreviewOptions</code>
<code>ampubym), 146</code>	<code>ampubym), 159</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>
<code>ogram.types.input_media_animation.InputMediaAnimation</code>	<code>ogram.types.location.Location</code>
<code>ampubym), 148</code>	<code>ampubym), 159</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>
<code>ogram.types.input_media_audio.InputMediaAudio</code>	<code>ogram.types.login_url.LoginUrl</code>
<code>ampubym), 149</code>	<code>ampubym), 160</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>
<code>ogram.types.input_media_document.InputMediaDocument</code>	<code>ogram.types.mask_position.MaskPosition</code>
<code>ampubym), 150</code>	<code>ampubym), 290</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>
<code>ogram.types.input_media_photo.InputMediaPhoto</code>	<code>ogram.types.maybe_inaccessible_message.MaybeInaccessibleMessage</code>
<code>ampubym), 151</code>	<code>ampubym), 160</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>
<code>ogram.types.input_media_video.InputMediaVideo</code>	<code>ogram.types.menu_button.MenuButton</code>
<code>ampubym), 152</code>	<code>ampubym), 161</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>
<code>ogram.types.input_message_content.InputMessageContent</code>	<code>ogram.types.menu_button_commands.MenuButtonCommands</code>
<code>ampubym), 286</code>	<code>ampubym), 161</code>
<code>model_computed_fields (ai- model_computed_fields (ai-</code>	<code>model_computed_fields (ai-</code>

<code>ogram.types.menu_button_default.MenuButtonDefault</code>	<code>ogram.types.passport_element_error_files.PassportElementErrorFiles</code>
<code>ampubym</code>), 162	<code>ampubym</code>), 298
<code>model_computed_fields</code> (ai- <code>ogram.types.menu_button_web_app.MenuButtonWebApp</code>	<code>model_computed_fields</code> (ai- <code>ogram.types.passport_element_error_front_side.PassportElementErrorFrontSide</code>
<code>ampubym</code>), 162	<code>ampubym</code>), 299
<code>model_computed_fields</code> (ai- <code>ogram.types.message.Message</code>	<code>model_computed_fields</code> (ai- <code>ogram.types.passport_element_error_reverse_side.PassportElementErrorReverseSide</code>
<code>ampubym</code>), 169	<code>ampubym</code>), 299
<code>model_computed_fields</code> (ai- <code>ogram.types.message_auto_delete_timer_changed.MessageAutoDeleteTimerChanged</code>	<code>model_computed_fields</code> (ai- <code>ogram.types.passport_element_error_selfie.PassportElementErrorSelfie</code>
<code>ampubym</code>), 214	<code>ampubym</code>), 300
<code>model_computed_fields</code> (ai- <code>ogram.types.message_entity.MessageEntity</code>	<code>model_computed_fields</code> (ai- <code>ogram.types.passport_element_error_translation_file.PassportElementErrorTranslationFile</code>
<code>ampubym</code>), 215	<code>ampubym</code>), 301
<code>model_computed_fields</code> (ai- <code>ogram.types.message_id.MessageId</code>	<code>model_computed_fields</code> (ai- <code>ogram.types.passport_element_error_translation_files.PassportElementErrorTranslationFiles</code>
<code>ampubym</code>), 216	<code>ampubym</code>), 302
<code>model_computed_fields</code> (ai- <code>ogram.types.message_origin.MessageOrigin</code>	<code>model_computed_fields</code> (ai- <code>ogram.types.passport_element_error_unspecified.PassportElementErrorUnspecified</code>
<code>ampubym</code>), 216	<code>ampubym</code>), 303
<code>model_computed_fields</code> (ai- <code>ogram.types.message_origin_channel.MessageOriginChannel</code>	<code>model_computed_fields</code> (ai- <code>ogram.types.passport_file.PassportFile</code>
<code>ampubym</code>), 217	<code>ampubym</code>), 303
<code>model_computed_fields</code> (ai- <code>ogram.types.message_origin_chat.MessageOriginChat</code>	<code>model_computed_fields</code> (ai- <code>ogram.types.photo_size.PhotoSize</code>
<code>ampubym</code>), 217	<code>ampubym</code>), 220
<code>model_computed_fields</code> (ai- <code>ogram.types.message_origin_hidden_user.MessageOriginHiddenUser</code>	<code>model_computed_fields</code> (aiogram.types.poll.PollAnswer)
<code>ampubym</code>), 218	<code>model_computed_fields</code> (ai- <code>ogram.types.poll_answer.PollAnswer</code>
<code>model_computed_fields</code> (ai- <code>ogram.types.message_origin_user.MessageOriginUser</code>	<code>model_computed_fields</code> (ai- <code>ogram.types.poll_option.PollOption</code>
<code>ampubym</code>), 218	<code>ampubym</code>), 222
<code>model_computed_fields</code> (ai- <code>ogram.types.message_reaction_count_updated.MessageReactionCountUpdated</code>	<code>model_computed_fields</code> (ai- <code>ogram.types.pre_checkout_query.PreCheckoutQuery</code>
<code>ampubym</code>), 219	<code>ampubym</code>), 306
<code>model_computed_fields</code> (ai- <code>ogram.types.message_reaction_updated.MessageReactionUpdated</code>	<code>model_computed_fields</code> (ai- <code>ogram.types.proximity_alert_triggered.ProximityAlertTriggered</code>
<code>ampubym</code>), 220	<code>ampubym</code>), 223
<code>model_computed_fields</code> (ai- <code>ogram.types.order_info.OrderInfo</code>	<code>model_computed_fields</code> (ai- <code>ogram.types.reaction_count.ReactionCount</code>
<code>ampubym</code>), 305	<code>ampubym</code>), 223
<code>model_computed_fields</code> (ai- <code>ogram.types.passport_data.PassportData</code>	<code>model_computed_fields</code> (ai- <code>ogram.types.reaction_type.ReactionType</code>
<code>ampubym</code>), 295	<code>ampubym</code>), 224
<code>model_computed_fields</code> (ai- <code>ogram.types.passport_element_error.PassportElementError</code>	<code>model_computed_fields</code> (ai- <code>ogram.types.reaction_type_custom_emoji.ReactionTypeCustomEmoji</code>
<code>ampubym</code>), 296	<code>ampubym</code>), 224
<code>model_computed_fields</code> (ai- <code>ogram.types.passport_element_error_data_field.PassportElementErrorDataField</code>	<code>model_computed_fields</code> (ai- <code>ogram.types.reaction_type_emoji.ReactionTypeEmoji</code>
<code>ampubym</code>), 297	<code>ampubym</code>), 224
<code>model_computed_fields</code> (ai- <code>ogram.types.passport_element_error_file.PassportElementErrorFile</code>	<code>model_computed_fields</code> (ai- <code>ogram.types.reply_keyboard_markup.ReplyKeyboardMarkup</code>
<code>ampubym</code>), 297	<code>ampubym</code>), 224
<code>model_computed_fields</code> (ai- <code>ogram.types.reply_keyboard_markup.ReplyKeyboardMarkup</code>	<code>model_computed_fields</code> (ai- <code>ogram.types.reply_keyboard_markup.ReplyKeyboardMarkup</code>
<code>ampubym</code>), 297	<code>ampubym</code>), 224

<code>model_computed_fields</code>	<code>(aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove)</code> , 226	<code>aiogram.types.user_shared.UserShared</code>	<code>(aiogram.types.user_shared.UserShared)</code> , 233
<code>model_computed_fields</code>	<code>(aiogram.types.reply_parameters.ReplyParameters)</code> , 227	<code>aiogram.types.users_shared.UsersShared</code>	<code>(aiogram.types.users_shared.UsersShared)</code> , 234
<code>model_computed_fields</code>	<code>(aiogram.types.response_parameters.ResponseParameters)</code> , 228	<code>aiogram.types.venue.Venue</code>	<code>(aiogram.types.venue.Venue)</code> , 234
<code>model_computed_fields</code>	<code>(aiogram.types.sent_web_app_message.SentWebAppMessage)</code> , 289	<code>aiogram.types.video.Video</code>	<code>(aiogram.types.video.Video)</code> , 235
<code>model_computed_fields</code>	<code>(aiogram.types.shared_user.SharedUser)</code> , 228	<code>aiogram.types.video_chat_ended.VideoChatEnded</code>	<code>(aiogram.types.video_chat_ended.VideoChatEnded)</code> , 236
<code>model_computed_fields</code>	<code>(aiogram.types.shipping_address.ShippingAddress)</code> , 307	<code>aiogram.types.video_chat_participants_invited.VideoChatParticipantsInvited</code>	<code>(aiogram.types.video_chat_participants_invited.VideoChatParticipantsInvited)</code> , 236
<code>model_computed_fields</code>	<code>(aiogram.types.shipping_option.ShippingOption)</code> , 307	<code>aiogram.types.video_chat_scheduled.VideoChatScheduled</code>	<code>(aiogram.types.video_chat_scheduled.VideoChatScheduled)</code> , 236
<code>model_computed_fields</code>	<code>(aiogram.types.shipping_query.ShippingQuery)</code> , 308	<code>aiogram.types.video_chat_started.VideoChatStarted</code>	<code>(aiogram.types.video_chat_started.VideoChatStarted)</code> , 237
<code>model_computed_fields</code>	<code>(aiogram.types.shipping_query.ShippingQuery)</code> , 308	<code>aiogram.types.video_note.VideoNote</code>	<code>(aiogram.types.video_note.VideoNote)</code> , 237
<code>model_computed_fields</code>	<code>(aiogram.types.sticker.Sticker)</code> , 291	<code>aiogram.types.voice.Voice</code>	<code>(aiogram.types.voice.Voice)</code> , 238
<code>model_computed_fields</code>	<code>(aiogram.types.sticker_set.StickerSet)</code> , 292	<code>aiogram.types.web_app_data.WebAppData</code>	<code>(aiogram.types.web_app_data.WebAppData)</code> , 238
<code>model_computed_fields</code>	<code>(aiogram.types.story.Story)</code> , 229	<code>aiogram.types.web_app_info.WebAppInfo</code>	<code>(aiogram.types.web_app_info.WebAppInfo)</code> , 239
<code>model_computed_fields</code>	<code>(aiogram.types.successful_payment.SuccessfulPayment)</code> , 309	<code>aiogram.types.webhook_info.WebhookInfo</code>	<code>(aiogram.types.webhook_info.WebhookInfo)</code> , 312
<code>model_computed_fields</code>	<code>(aiogram.types.switch_inline_query_chosen_chat.SwitchInlineQueryChosenChat)</code> , 230	<code>aiogram.types.write_access_allowed.WriteAccessAllowed</code>	<code>(aiogram.types.write_access_allowed.WriteAccessAllowed)</code> , 239
<code>model_computed_fields</code>	<code>(aiogram.types.text_quote.TextQuote)</code> , 230	<code>aiogram.types.web_app_chat.WebAppChat</code>	<code>(aiogram.types.web_app_chat.WebAppChat)</code> , 598
<code>model_computed_fields</code>	<code>(aiogram.types.update.Update)</code> , 311	<code>aiogram.types.web_app_init_data.WebAppInitData</code>	<code>(aiogram.types.web_app_init_data.WebAppInitData)</code> , 595
<code>model_computed_fields</code>	<code>(aiogram.types.user.User)</code> , 231	<code>aiogram.types.web_app_user.WebAppUser</code>	<code>(aiogram.types.web_app_user.WebAppUser)</code> , 597
<code>model_computed_fields</code>	<code>(aiogram.types.user_chat_boosts.UserChatBoosts)</code> , 232	<code>aiogram.types.web_app_chat.WebAppChat</code>	<code>(aiogram.types.web_app_chat.WebAppChat)</code> , 598
<code>model_computed_fields</code>	<code>(aiogram.types.user_profile_photos.UserProfilePhotos)</code> , 233	<code>aiogram.types.web_app_init_data.WebAppInitData</code>	<code>(aiogram.types.web_app_init_data.WebAppInitData)</code> , 595
<code>model_computed_fields</code>	<code>(aiogram.types.user_profile_photos.UserProfilePhotos)</code> , 233	<code>aiogram.types.web_app_user.WebAppUser</code>	<code>(aiogram.types.web_app_user.WebAppUser)</code> , 597

`model_fields (aiogram.utils.web_app.WebAppChat
 ampubym), 598`
`ogram.methods.create_invoice_link.CreateInvoiceLink
 memod), 478`

`model_fields (aiogram.utils.web_app.WebAppInitData
 ampubym), 595`
`model_post_init() (ai-
 ogram.methods.create_new_sticker_set.CreateNewStickerSet
 memod), 316`

`model_fields (aiogram.utils.web_app.WebAppUser
 ampubym), 597`
`model_post_init() (ai-
 ogram.methods.decline_chat_join_request.DeclineChatJoin
 memod), 348`

`model_post_init() (ai-
 ogram.methods.add_sticker_to_set.AddStickerToSet
 memod), 315`
`model_post_init() (ai-
 ogram.methods.delete_chat_photo.DeleteChatPhoto
 memod), 349`

`model_post_init() (ai-
 ogram.methods.answer_callback_query.AnswerCallbackQuery
 memod), 334`
`model_post_init() (ai-
 ogram.methods.delete_chat_sticker_set.DeleteChatStickerSet
 memod), 351`

`model_post_init() (ai-
 ogram.methods.answer_inline_query.AnswerInlineQuery
 memod), 467`
`model_post_init() (ai-
 ogram.methods.delete_forum_topic.DeleteForumTopic
 memod), 352`

`model_post_init() (ai-
 ogram.methods.answer_pre_checkout_query.AnswerPreCheckoutQuery
 memod), 474`
`model_post_init() (ai-
 ogram.methods.delete_message.DeleteMessage
 memod), 452`

`model_post_init() (ai-
 ogram.methods.answer_shipping_query.AnswerShippingQuery
 memod), 475`
`model_post_init() (ai-
 ogram.methods.delete_messages.DeleteMessages
 memod), 453`

`model_post_init() (ai-
 ogram.methods.answer_web_app_query.AnswerWebAppQuery
 memod), 468`
`model_post_init() (ai-
 ogram.methods.delete_my_commands.DeleteMyCommands
 memod), 353`

`model_post_init() (ai-
 ogram.methods.approve_chat_join_request.ApproveChatJoinRequest
 memod), 335`
`model_post_init() (ai-
 ogram.methods.delete_sticker_from_set.DeleteStickerFromSet
 memod), 317`

`model_post_init() (ai-
 ogram.methods.ban_chat_member.BanChatMember
 memod), 336`
`model_post_init() (ai-
 ogram.methods.delete_sticker_set.DeleteStickerSet
 memod), 318`

`model_post_init() (ai-
 ogram.methods.ban_chat_sender_chat.BanChatSenderChat
 memod), 338`
`model_post_init() (ai-
 ogram.methods.delete_webhook.DeleteWebhook
 memod), 483`

`model_post_init() (aiogram.methods.close.Close
 memod), 339`
`model_post_init() (ai-
 ogram.methods.close_forum_topic.CloseForumTopic
 memod), 340`

`model_post_init() (ai-
 ogram.methods.close_general_forum_topic.CloseGeneralForumTopic
 memod), 341`
`model_post_init() (ai-
 ogram.methods.edit_chat_invite_link.EditChatInviteLink
 memod), 354`

`model_post_init() (ai-
 ogram.methods.copy_message.CopyMessage
 memod), 343`
`model_post_init() (ai-
 ogram.methods.edit_forum_topic.EditForumTopic
 memod), 355`

`model_post_init() (ai-
 ogram.methods.copy_messages.CopyMessages
 memod), 345`
`model_post_init() (ai-
 ogram.methods.edit_general_forum_topic.EditGeneralForum
 memod), 356`

`model_post_init() (ai-
 ogram.methods.create_chat_invite_link.CreateChatInviteLink
 memod), 346`
`model_post_init() (ai-
 ogram.methods.edit_message_caption.EditMessageCaption
 memod), 454`

`model_post_init() (ai-
 ogram.methods.create_forum_topic.CreateForumTopic
 memod), 347`
`model_post_init() (ai-
 ogram.methods.edit_message_live_location.EditMessageLive
 memod), 456`

`model_post_init() (ai-
 ogram.methods.create_forum_topic.CreateForumTopic
 memod), 347`
`model_post_init() (ai-
 ogram.methods.edit_message_media.EditMessageMedia
 memod), 458`

`model_post_init() (ai-
 ogram.methods.create_forum_topic.CreateForumTopic
 memod), 347`
`model_post_init() (ai-
 ogram.methods.create_forum_topic.CreateForumTopic
 memod), 347`

```

ogram.methods.edit_message_reply_markup.EditMessageReplyMarkup
memod), 459
ogram.methods.get_my_description.GetMyDescription
memod), 372
model_post_init() (ai- model_post_init() (ai-
ogram.methods.edit_message_text.EditMessageText ogram.methods.get_my_name.GetMyName
memod), 461 memod), 373
model_post_init() (ai- model_post_init() (ai-
ogram.methods.export_chat_invite_link.ExportChatInviteLink ogram.methods.get_my_short_description.GetMyShortDesc
memod), 358 memod), 374
model_post_init() (ai- model_post_init() (ai-
ogram.methods.forward_message.ForwardMessage ogram.methods.get_sticker_set.GetStickerSet
memod), 359 memod), 320
model_post_init() (ai- model_post_init() (ai-
ogram.methods.forward_messages.ForwardMessages ogram.methods.get_updates.GetUpdates
memod), 360 memod), 484
model_post_init() (ai- model_post_init() (ai-
ogram.methods.get_business_connection.GetBusinessConnection ogram.methods.get_user_chat_boosts.GetUserChatBoosts
memod), 361 memod), 375
model_post_init() (ai- model_post_init() (ai-
ogram.methods.get_chat.GetChat memod), ogram.methods.get_user_profile_photos.GetUserProfilePhotos
362 memod), 375
model_post_init() (ai- model_post_init() (ai-
ogram.methods.get_chat_administrators.GetChatAdministrators ogram.methods.get_webhook_info.GetWebhookInfo
memod), 363 memod), 485
model_post_init() (ai- model_post_init() (ai-
ogram.methods.get_chat_member.GetChatMember ogram.methods.hide_general_forum_topic.HideGeneralForum
memod), 364 memod), 376
model_post_init() (ai- model_post_init() (ai-
ogram.methods.get_chat_member_count.GetChatMemberCount ogram.methods.leave_chat.LeaveChat
memod), 365 memod), 377
model_post_init() (ai- model_post_init() (ai-
ogram.methods.get_chat_menu_button.GetChatMenuButton ogram.methods.log_out.LogOut memod),
memod), 366 378
model_post_init() (ai- model_post_init() (ai-
ogram.methods.get_custom_emoji_stickers.GetCustomEmojiStickers ogram.methods.pin_chat_message.PinChatMessage
memod), 319 memod), 379
model_post_init() (ai- model_post_init() (ai-
ogram.methods.get_file.GetFile memod), ogram.methods.promote_chat_member.PromoteChatMember
367 memod), 382
model_post_init() (ai- model_post_init() (ai-
ogram.methods.get_forum_topic_icon_stickers.GetForumTopicIconStickers ogram.methods.reopen_forum_topic.ReopenForumTopic
memod), 368 memod), 383
model_post_init() (ai- model_post_init() (ai-
ogram.methods.get_game_high_scores.GetGameHighScores ogram.methods.reopen_general_forum_topic.ReopenGeneral
memod), 470 memod), 384
model_post_init() (ai- model_post_init() (ai-
ogram.methods.get_me.GetMe memod), ogram.methods.replace_sticker_in_set.ReplaceStickerInSet
369 memod), 321
model_post_init() (ai- model_post_init() (ai-
ogram.methods.get_my_commands.GetMyCommands ogram.methods.restrict_chat_member.RestrictChatMember
memod), 370 memod), 386
model_post_init() (ai- model_post_init() (ai-
ogram.methods.get_my_default_administrator_rights.GetMyDefaultAdministratorRights ogram.methods.revoke_chat_invite_link.RevokeChatInviteL
memod), 371 memod), 387
model_post_init() (ai- model_post_init() (ai-

```


<code>ogram.methods.send_animation.SendAnimation</code> <code>memod</code>), 389	<code>ogram.methods.set_chat_administrator_custom_title.SetChatAdministratorCustomTitle</code> <code>memod</code>), 428
<code>model_post_init()</code> <code>ogram.methods.send_audio.SendAudio</code> <code>memod</code>), 392	<code>model_post_init()</code> <code>ogram.methods.set_chat_description.SetChatDescription</code> <code>memod</code>), 430
<code>model_post_init()</code> <code>ogram.methods.send_chat_action.SendChatAction</code> <code>memod</code>), 394	<code>model_post_init()</code> <code>ogram.methods.set_chat_menu_button.SetChatMenuButton</code> <code>memod</code>), 431
<code>model_post_init()</code> <code>ogram.methods.send_contact.SendContact</code> <code>memod</code>), 396	<code>model_post_init()</code> <code>ogram.methods.set_chat_permissions.SetChatPermissions</code> <code>memod</code>), 432
<code>model_post_init()</code> <code>ogram.methods.send_dice.SendDice</code> <code>memod</code>), 398	<code>model_post_init()</code> <code>ogram.methods.set_chat_photo.SetChatPhoto</code> <code>memod</code>), 433
<code>model_post_init()</code> <code>ogram.methods.send_document.SendDocument</code> <code>memod</code>), 401	<code>model_post_init()</code> <code>ogram.methods.set_chat_sticker_set.SetChatStickerSet</code> <code>memod</code>), 434
<code>model_post_init()</code> <code>ogram.methods.send_game.SendGame</code> <code>memod</code>), 471	<code>model_post_init()</code> <code>ogram.methods.set_chat_title.SetChatTitle</code> <code>memod</code>), 435
<code>model_post_init()</code> <code>ogram.methods.send_invoice.SendInvoice</code> <code>memod</code>), 481	<code>model_post_init()</code> <code>ogram.methods.set_custom_emoji_sticker_set_thumbnail.SetCustomEmojiStickerSetThumbnail</code> <code>memod</code>), 325
<code>model_post_init()</code> <code>ogram.methods.send_location.SendLocation</code> <code>memod</code>), 403	<code>model_post_init()</code> <code>ogram.methods.set_game_score.SetGameScore</code> <code>memod</code>), 473
<code>model_post_init()</code> <code>ogram.methods.send_media_group.SendMediaGroup</code> <code>memod</code>), 406	<code>model_post_init()</code> <code>ogram.methods.set_message_reaction.SetMessageReaction</code> <code>memod</code>), 437
<code>model_post_init()</code> <code>ogram.methods.send_message.SendMessage</code> <code>memod</code>), 409	<code>model_post_init()</code> <code>ogram.methods.set_my_commands.SetMyCommands</code> <code>memod</code>), 438
<code>model_post_init()</code> <code>ogram.methods.send_photo.SendPhoto</code> <code>memod</code>), 412	<code>model_post_init()</code> <code>ogram.methods.set_my_default_administrator_rights.SetMyDefaultAdministratorRights</code> <code>memod</code>), 439
<code>model_post_init()</code> <code>ogram.methods.send_poll.SendPoll</code> <code>memod</code>), 415	<code>model_post_init()</code> <code>ogram.methods.set_my_description.SetMyDescription</code> <code>memod</code>), 440
<code>model_post_init()</code> <code>ogram.methods.send_sticker.SendSticker</code> <code>memod</code>), 323	<code>model_post_init()</code> <code>ogram.methods.set_my_name.SetMyName</code> <code>memod</code>), 441
<code>model_post_init()</code> <code>ogram.methods.send_venue.SendVenue</code> <code>memod</code>), 418	<code>model_post_init()</code> <code>ogram.methods.set_my_short_description.SetMyShortDescription</code> <code>memod</code>), 442
<code>model_post_init()</code> <code>ogram.methods.send_video.SendVideo</code> <code>memod</code>), 421	<code>model_post_init()</code> <code>ogram.methods.set_passport_data_errors.SetPassportDataErrors</code> <code>memod</code>), 488
<code>model_post_init()</code> <code>ogram.methods.send_video_note.SendVideoNote</code> <code>memod</code>), 424	<code>model_post_init()</code> <code>ogram.methods.set_sticker_emoji_list.SetStickerEmojiList</code> <code>memod</code>), 326
<code>model_post_init()</code> <code>ogram.methods.send_voice.SendVoice</code> <code>memod</code>), 427	<code>model_post_init()</code> <code>ogram.methods.set_sticker_keywords.SetStickerKeywords</code> <code>memod</code>), 327
<code>model_post_init()</code>	<code>model_post_init()</code>


```

        memod), 30
model_post_init() (aiogram.types.bot_name.BotName memod), 31
model_post_init() (aiogram.types.bot_short_description.BotShortDescription memod), 31
model_post_init() (aiogram.types.business_connection.BusinessConnection memod), 32
model_post_init() (aiogram.types.business_intro.BusinessIntro memod), 32
model_post_init() (aiogram.types.business_location.BusinessLocation memod), 32
model_post_init() (aiogram.types.business_messages_deleted.BusinessMessagesDeleted memod), 33
model_post_init() (aiogram.types.business_opening_hours.BusinessOpeningHours memod), 33
model_post_init() (aiogram.types.business_opening_hours_interval.BusinessOpeningHoursInterval memod), 34
model_post_init() (aiogram.types.callback_game.CallbackGame memod), 313
model_post_init() (aiogram.types.callback_query.CallbackQuery memod), 35
model_post_init() (aiogram.types.chat.Chat memod), 40
model_post_init() (aiogram.types.chat_administrator_rights.ChatAdministratorRights memod), 53
model_post_init() (aiogram.types.chat_background.ChatBackground memod), 54
model_post_init() (aiogram.types.chat_boost.ChatBoost memod), 54
model_post_init() (aiogram.types.chat_boost_added.ChatBoostAdded memod), 55
model_post_init() (aiogram.types.chat_boost_removed.ChatBoostRemoved memod), 55
model_post_init() (aiogram.types.chat_boost_source.ChatBoostSource memod), 56
model_post_init() (aiogram.types.chat_boost_source_gift_code.ChatBoostSourceGiftCode memod), 56
model_post_init() (aiogram.types.chat_boost_source_giveaway.ChatBoostSourceGiveaway memod), 57
model_post_init() (aiogram.types.chat_boost_premium.ChatBoostSourcePremium memod), 57
model_post_init() (aiogram.types.chat_boost_updated.ChatBoostUpdated memod), 58
model_post_init() (aiogram.types.chat_full_info.ChatFullInfo memod), 61
model_post_init() (aiogram.types.chat_invite_link.ChatInviteLink memod), 63
model_post_init() (aiogram.types.chat_join_request.ChatJoinRequest memod), 67
model_post_init() (aiogram.types.chat_location.ChatLocation memod), 104
model_post_init() (aiogram.types.chat_member.ChatMember memod), 104
model_post_init() (aiogram.types.chat_member_administrator.ChatMemberAdministrator memod), 106
model_post_init() (aiogram.types.chat_member_banned.ChatMemberBanned memod), 107
model_post_init() (aiogram.types.chat_member_left.ChatMemberLeft memod), 107
model_post_init() (aiogram.types.chat_member_member.ChatMemberMember memod), 108
model_post_init() (aiogram.types.chat_member_owner.ChatMemberOwner memod), 108
model_post_init() (aiogram.types.chat_member_restricted.ChatMemberRestricted memod), 110
model_post_init() (aiogram.types.chat_member_updated.ChatMemberUpdated memod), 122
model_post_init() (aiogram.types.chat_permissions.ChatPermissions memod), 131
model_post_init() (aiogram.types.chat_photo.ChatPhoto memod), 132
model_post_init() (aiogram.types.chat_shared.ChatShared memod), 133

```

<code>model_post_init()</code> <code>ogram.types.chosen_inline_result.ChosenInlineResult</code> <code>memod)</code> , 240	<code>(ai-</code> <code>ogram.types.giveaway.Giveaway</code> <code>memod)</code> , 240
<code>model_post_init()</code> (<code>aiogram.types.contact.Contact</code> <code>memod)</code> , 133	<code>ogram.types.giveaway_completed.GiveawayCompleted</code> <code>memod)</code> , 142
<code>model_post_init()</code> (<code>aiogram.types.dice.Dice</code> <code>me-</code> <code>mod)</code> , 134	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.giveaway_created.GiveawayCreated</code> <code>memod)</code> , 142
<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.document.Document</code> <code>memod)</code> , 134	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.giveaway_winners.GiveawayWinners</code> <code>memod)</code> , 143
<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.encrypted_credentials.EncryptedCredentials</code> <code>memod)</code> , 293	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.inaccessible_message.InaccessibleMessage</code> <code>memod)</code> , 144
<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.encrypted_passport_element.EncryptedPassportElement</code> <code>memod)</code> , 294	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.inline_keyboard_button.InlineKeyboardButton</code> <code>memod)</code> , 145
<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.error_event.ErrorEvent</code> <code>memod)</code> , 573	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.inline_keyboard_markup.InlineKeyboardMarkup</code> <code>memod)</code> , 145
<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.external_reply_info.ExternalReplyInfo</code> <code>memod)</code> , 136	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.inline_query.InlineQuery</code> <code>memod)</code> , 241
<code>model_post_init()</code> (<code>aiogram.types.file.File</code> <code>memod)</code> , 137	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.inline_query_result.InlineQueryResult</code> <code>memod)</code> , 242
<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.force_reply.ForceReply</code> <code>me-</code> <code>mod)</code> , 138	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.inline_query_result_article.InlineQueryResultArticle</code> <code>memod)</code> , 244
<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.forum_topic.ForumTopic</code> <code>memod)</code> , 139	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.inline_query_result_audio.InlineQueryResultAudio</code> <code>memod)</code> , 246
<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.forum_topic_closed.ForumTopicClosed</code> <code>memod)</code> , 139	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.inline_query_result_cached_audio.InlineQueryResultAudioCached</code> <code>memod)</code> , 248
<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.forum_topic_created.ForumTopicCreated</code> <code>memod)</code> , 139	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.inline_query_result_cached_document.InlineQueryResultDocumentCached</code> <code>memod)</code> , 250
<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.forum_topic_edited.ForumTopicEdited</code> <code>memod)</code> , 140	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.inline_query_result_cached_gif.InlineQueryResultGifCached</code> <code>memod)</code> , 252
<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.forum_topic_reopened.ForumTopicReopened</code> <code>memod)</code> , 140	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.inline_query_result_cached_mpeg4_gif.InlineQueryResultMpeg4GifCached</code> <code>memod)</code> , 254
<code>model_post_init()</code> (<code>aiogram.types.game.Game</code> <code>me-</code> <code>mod)</code> , 314	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.inline_query_result_cached_photo.InlineQueryResultPhotoCached</code> <code>memod)</code> , 256
<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.game_high_score.GameHighScore</code> <code>memod)</code> , 314	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.inline_query_result_cached_sticker.InlineQueryResultStickerCached</code> <code>memod)</code> , 257
<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.general_forum_topic_hidden.GeneralForumTopicHidden</code> <code>memod)</code> , 140	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.inline_query_result_cached_video.InlineQueryResultVideoCached</code> <code>memod)</code> , 260
<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.general_forum_topic_unhidden.GeneralForumTopicUnhidden</code> <code>memod)</code> , 141	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.inline_query_result_cached_video.InlineQueryResultVideoCached</code> <code>memod)</code> , 260
<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.general_forum_topic_unhidden.GeneralForumTopicUnhidden</code> <code>memod)</code> , 141	<code>model_post_init()</code> (<code>ai-</code> <code>ogram.types.inline_query_result_cached_video.InlineQueryResultVideoCached</code> <code>memod)</code> , 260

```

ogram.types.inline_query_result_cached_voice.InlineQueryResultCachedVoice.media_document.InputMediaDocument
memod), 262                                memod), 150
model_post_init()                          (ai- model_post_init()                          (ai-
ogram.types.inline_query_result_contact.InlineQueryResultContact.input_media_photo.InputMediaPhoto
memod), 264                                memod), 151
model_post_init()                          (ai- model_post_init()                          (ai-
ogram.types.inline_query_result_document.InlineQueryResultDocument.input_media_video.InputMediaVideo
memod), 266                                memod), 152
model_post_init()                          (ai- model_post_init()                          (ai-
ogram.types.inline_query_result_game.InlineQueryResultGame.types.input_message_content.InputMessageContent
memod), 267                                memod), 286
model_post_init()                          (ai- model_post_init()                          (ai-
ogram.types.inline_query_result_gif.InlineQueryResultGif.types.input_poll_option.InputPollOption
memod), 269                                memod), 153
model_post_init()                          (ai- model_post_init()                          (ai-
ogram.types.inline_query_result_location.InlineQueryResultLocation.input_sticker.InputSticker
memod), 271                                memod), 289
model_post_init()                          (ai- model_post_init()                          (ai-
ogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif_text_message_content.InputTextMessage
memod), 273                                memod), 287
model_post_init()                          (ai- model_post_init()                          (ai-
ogram.types.inline_query_result_photo.InlineQueryResultPhoto.types.input_venue_message_content.InputVenueMes
memod), 275                                memod), 288
model_post_init()                          (ai- model_post_init() (aiogram.types.invoice.Invoice
ogram.types.inline_query_result_venue.InlineQueryResultVenue) 304
memod), 277                                model_post_init()                          (ai-
model_post_init()                          (ai- ogram.types.keyboard_button.KeyboardButton
ogram.types.inline_query_result_video.InlineQueryResultVideo) 154
memod), 279                                model_post_init()                          (ai-
model_post_init()                          (ai- ogram.types.keyboard_button_poll_type.KeyboardButtonPoll
ogram.types.inline_query_result_voice.InlineQueryResultVoice) 154
memod), 281                                model_post_init()                          (ai-
model_post_init()                          (ai- ogram.types.keyboard_button_request_chat.KeyboardButton
ogram.types.inline_query_results_button.InlineQueryResultsButton) 156
memod), 281                                model_post_init()                          (ai-
model_post_init()                          (ai- ogram.types.keyboard_button_request_user.KeyboardButton
ogram.types.input_contact_message_content.InputContactMessageContent) 157
memod), 282                                model_post_init()                          (ai-
model_post_init()                          (ai- ogram.types.keyboard_button_request_users.KeyboardButton
ogram.types.input_invoice_message_content.InputInvoiceMessageContent) 158
memod), 284                                model_post_init()                          (ai-
model_post_init()                          (ai- ogram.types.labeled_price.LabeledPrice
ogram.types.input_location_message_content.InputLocationMessageContent) 304
memod), 286                                model_post_init()                          (ai-
model_post_init()                          (ai- ogram.types.link_preview_options.LinkPreviewOptions
ogram.types.input_media.InputMedia
memod), 147                                memod), 159
model_post_init()                          (ai- ogram.types.location.Location memod),
ogram.types.input_media_animation.InputMediaAnimation) 159
memod), 148                                model_post_init()                          (ai-
model_post_init()                          (ai- ogram.types.login_url.LoginUrl memod),
ogram.types.input_media_audio.InputMediaAudio 160
memod), 149                                model_post_init()                          (ai-
model_post_init()                          (ai- ogram.types.mask_position.MaskPosition

```


`method`), 290
`model_post_init()` (`aiogram.types.maybe_inaccessible_message.MaybeInaccessibleMessage` `method`), 160
`model_post_init()` (`aiogram.types.menu_button.MenuButton` `method`), 161
`model_post_init()` (`aiogram.types.menu_button_commands.MenuButtonCommands` `method`), 161
`model_post_init()` (`aiogram.types.menu_button_default.MenuButtonDefault` `method`), 162
`model_post_init()` (`aiogram.types.menu_button_web_app.MenuButtonWebApp` `method`), 162
`model_post_init()` (`aiogram.types.message.Message` `method`), 169
`model_post_init()` (`aiogram.types.message_auto_delete_timer_changed.MessageAutoDeleteTimerChanged` `method`), 215
`model_post_init()` (`aiogram.types.message_entity.MessageEntity` `method`), 215
`model_post_init()` (`aiogram.types.message_id.MessageId` `method`), 216
`model_post_init()` (`aiogram.types.message_origin.MessageOrigin` `method`), 216
`model_post_init()` (`aiogram.types.message_origin_channel.MessageOriginChannel` `method`), 217
`model_post_init()` (`aiogram.types.message_origin_chat.MessageOriginChat` `method`), 217
`model_post_init()` (`aiogram.types.message_origin_hidden_user.MessageOriginHiddenUser` `method`), 218
`model_post_init()` (`aiogram.types.message_origin_user.MessageOriginUser` `method`), 218
`model_post_init()` (`aiogram.types.message_reaction_count_updated.MessageReactionCountUpdated` `method`), 219
`model_post_init()` (`aiogram.types.message_reaction_updated.MessageReactionUpdated` `method`), 220
`model_post_init()` (`aiogram.types.order_info.OrderInfo` `method`), 305
`model_post_init()` (`aiogram.types.passport_data.PassportData` `method`), 223
`method`), 295
`model_post_init()` (`aiogram.types.passport_element_error.PassportElementError` `method`), 296
`model_post_init()` (`aiogram.types.passport_element_error_data_field.PassportElementErrorDataField` `method`), 297
`model_post_init()` (`aiogram.types.passport_element_error_file.PassportElementErrorFile` `method`), 297
`model_post_init()` (`aiogram.types.passport_element_error_files.PassportElementErrorFiles` `method`), 298
`model_post_init()` (`aiogram.types.passport_element_error_front_side.PassportElementErrorFrontSide` `method`), 299
`model_post_init()` (`aiogram.types.passport_element_error_reverse_side.PassportElementErrorReverseSide` `method`), 299
`model_post_init()` (`aiogram.types.passport_element_error_selfie.PassportElementErrorSelfie` `method`), 300
`model_post_init()` (`aiogram.types.passport_element_error_translation_file.PassportElementErrorTranslationFile` `method`), 301
`model_post_init()` (`aiogram.types.passport_element_error_translation_files.PassportElementErrorTranslationFiles` `method`), 302
`model_post_init()` (`aiogram.types.passport_element_error_unspecified.PassportElementErrorUnspecified` `method`), 303
`model_post_init()` (`aiogram.types.passport_file.PassportFile` `method`), 303
`aiogram.types.photo_size.PhotoSize` `method`), 220
`aiogram.types.poll.Poll` `method`, 221
`aiogram.types.poll_answer.PollAnswer` `method`), 222
`aiogram.types.poll_option.PollOption` `method`, 222
`aiogram.types.pre_checkout_query.PreCheckoutQuery` `method`, 206
`aiogram.types.proximity_alert_triggered.ProximityAlertTriggered` `method`), 223
`aiogram.types.reaction_count.ReactionCount` `method`, 223

model_post_init()	(aiogram.types.reaction_type.ReactionType memod), 224	model_post_init()	(aiogram.types.user.User memod), 231
model_post_init()	(aiogram.types.reaction_type_custom_emoji.ReactionTypeCustomEmoji memod), 224	model_post_init()	(aiogram.types.user_chat_boosts.UserChatBoosts memod), 230
model_post_init()	(aiogram.types.reaction_type_emoji.ReactionTypeEmoji memod), 225	model_post_init()	(aiogram.types.user_profile_photos.UserProfilePhotos memod), 233
model_post_init()	(aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup memod), 225	model_post_init()	(aiogram.types.user_shared.UserShared memod), 233
model_post_init()	(aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove memod), 226	model_post_init()	(aiogram.types.users_shared.UsersShared memod), 234
model_post_init()	(aiogram.types.reply_parameters.ReplyParameters memod), 227	model_post_init()	(aiogram.types.venue.Venue memod), 234
model_post_init()	(aiogram.types.response_parameters.ResponseParameters memod), 228	model_post_init()	(aiogram.types.video.Video memod), 235
model_post_init()	(aiogram.types.sent_web_app_message.SentWebAppMessage memod), 289	model_post_init()	(aiogram.types.video_chat_ended.VideoChatEnded memod), 236
model_post_init()	(aiogram.types.shared_user.SharedUser memod), 228	model_post_init()	(aiogram.types.video_chat_participants_invited.VideoChatParticipantsInvited memod), 236
model_post_init()	(aiogram.types.shipping_address.ShippingAddress memod), 307	model_post_init()	(aiogram.types.video_chat_scheduled.VideoChatScheduled memod), 236
model_post_init()	(aiogram.types.shipping_option.ShippingOption memod), 307	model_post_init()	(aiogram.types.video_chat_started.VideoChatStarted memod), 237
model_post_init()	(aiogram.types.shipping_query.ShippingQuery memod), 308	model_post_init()	(aiogram.types.video_note.VideoNote memod), 237
model_post_init()	(aiogram.types.sticker.Sticker memod), 291	model_post_init()	(aiogram.types.voice.Voice memod), 238
model_post_init()	(aiogram.types.sticker_set.StickerSet memod), 292	model_post_init()	(aiogram.types.web_app_data.WebAppData memod), 238
model_post_init()	(aiogram.types.story.Story memod), 229	model_post_init()	(aiogram.types.web_app_info.WebAppInfo memod), 239
model_post_init()	(aiogram.types.successful_payment.SuccessfulPayment memod), 309	model_post_init()	(aiogram.types.webhook_info.WebhookInfo memod), 312
model_post_init()	(aiogram.types.switch_inline_query_chosen_chat.SwitchInlineQueryChosenChat memod), 230	model_post_init()	(aiogram.types.write_access_allowed.WriteAccessAllowed memod), 239
model_post_init()	(aiogram.types.text_quote.TextQuote memod), 230	model_post_init()	(aiogram.types.web_app_chat_data.WebAppChatData memod), 598
model_post_init()	(aiogram.types.update.Update memod), 311	model_post_init()	(aiogram.types.web_app_init_data.WebAppInitData memod), 595
		model_post_init()	(aiogram.types.web_app_message.WebAppMessage memod), 595

ogram.utils.web_app.WebAppUser *метод*),
597

module

- `aiogram.dispatcher.flags`, 576
- `aiogram.enums.bot_command_scope_type`, 489
- `aiogram.enums.chat_action`, 489
- `aiogram.enums.chat_boost_source_type`, 490
- `aiogram.enums.chat_member_status`, 490
- `aiogram.enums.chat_type`, 491
- `aiogram.enums.content_type`, 491
- `aiogram.enums.currency`, 493
- `aiogram.enums.dice_emoji`, 496
- `aiogram.enums.encrypted_passport_element`, 496
- `aiogram.enums.inline_query_result_type`, 497
- `aiogram.enums.input_media_type`, 498
- `aiogram.enums.keyboard_button_poll_type_type`, 498
- `aiogram.enums.mask_position_point`, 498
- `aiogram.enums.menu_button_type`, 499
- `aiogram.enums.message_entity_type`, 499
- `aiogram.enums.message_origin_type`, 500
- `aiogram.enums.parse_mode`, 500
- `aiogram.enums.passport_element_error_type`, 500
- `aiogram.enums.poll_type`, 501
- `aiogram.enums.reaction_type_type`, 501
- `aiogram.enums.sticker_format`, 501
- `aiogram.enums.sticker_type`, 502
- `aiogram.enums.topic_icon_color`, 502
- `aiogram.enums.update_type`, 502
- `aiogram.exceptions`, 574
- `aiogram.handlers.callback_query`, 578
- `aiogram.methods.add_sticker_to_set`, 315
- `aiogram.methods.answer_callback_query`, 333
- `aiogram.methods.answer_inline_query`, 465
- `aiogram.methods.answer_pre_checkout_query`, 474
- `aiogram.methods.answer_shipping_query`, 475
- `aiogram.methods.answer_web_app_query`, 468
- `aiogram.methods.approve_chat_join_request`, 335
- `aiogram.methods.ban_chat_member`, 336
- `aiogram.methods.ban_chat_sender_chat`, 337
- `aiogram.methods.close`, 339
- `aiogram.methods.close_forum_topic`, 340
- `aiogram.methods.close_general_forum_topic`, 341
- `aiogram.methods.copy_message`, 342
- `aiogram.methods.copy_messages`, 344
- `aiogram.methods.create_chat_invite_link`, 346
- `aiogram.methods.create_forum_topic`, 347
- `aiogram.methods.create_invoice_link`, 476
- `aiogram.methods.create_new_sticker_set`, 316
- `aiogram.methods.decline_chat_join_request`, 348
- `aiogram.methods.delete_chat_photo`, 349
- `aiogram.methods.delete_chat_sticker_set`, 350
- `aiogram.methods.delete_forum_topic`, 351
- `aiogram.methods.delete_message`, 451
- `aiogram.methods.delete_messages`, 453
- `aiogram.methods.delete_my_commands`, 352
- `aiogram.methods.delete_sticker_from_set`, 317
- `aiogram.methods.delete_sticker_set`, 318
- `aiogram.methods.delete_webhook`, 482
- `aiogram.methods.edit_chat_invite_link`, 354
- `aiogram.methods.edit_forum_topic`, 355
- `aiogram.methods.edit_general_forum_topic`, 356
- `aiogram.methods.edit_message_caption`, 454
- `aiogram.methods.edit_message_live_location`, 455
- `aiogram.methods.edit_message_media`, 458
- `aiogram.methods.edit_message_reply_markup`, 459
- `aiogram.methods.edit_message_text`, 460
- `aiogram.methods.export_chat_invite_link`, 357
- `aiogram.methods.forward_message`, 358
- `aiogram.methods.forward_messages`, 360
- `aiogram.methods.get_business_connection`, 361
- `aiogram.methods.get_chat`, 362
- `aiogram.methods.get_chat_administrators`, 363
- `aiogram.methods.get_chat_member`, 364
- `aiogram.methods.get_chat_member_count`, 365
- `aiogram.methods.get_chat_menu_button`, 366
- `aiogram.methods.get_custom_emoji_stickers`, 319
- `aiogram.methods.get_file`, 367
- `aiogram.methods.get_forum_topic_icon_stickers`, 368
- `aiogram.methods.get_game_high_scores`, 469
- `aiogram.methods.get_me`, 369
- `aiogram.methods.get_my_commands`, 370
- `aiogram.methods.get_my_default_administrator_rights`, 371

[aiogram.methods.get_my_description](#), 372
[aiogram.methods.get_my_name](#), 373
[aiogram.methods.get_my_short_description](#), 374
[aiogram.methods.get_sticker_set](#), 320
[aiogram.methods.get_updates](#), 483
[aiogram.methods.get_user_chat_boosts](#), 374
[aiogram.methods.get_user_profile_photos](#), 375
[aiogram.methods.get_webhook_info](#), 485
[aiogram.methods.hide_general_forum_topic](#), 376
[aiogram.methods.leave_chat](#), 377
[aiogram.methods.log_out](#), 378
[aiogram.methods.pin_chat_message](#), 379
[aiogram.methods.promote_chat_member](#), 380
[aiogram.methods.reopen_forum_topic](#), 383
[aiogram.methods.reopen_general_forum_topic](#), 384
[aiogram.methods.replace_sticker_in_set](#), 321
[aiogram.methods.restrict_chat_member](#), 385
[aiogram.methods.revoke_chat_invite_link](#), 387
[aiogram.methods.send_animation](#), 388
[aiogram.methods.send_audio](#), 391
[aiogram.methods.send_chat_action](#), 394
[aiogram.methods.send_contact](#), 395
[aiogram.methods.send_dice](#), 397
[aiogram.methods.send_document](#), 400
[aiogram.methods.send_game](#), 470
[aiogram.methods.send_invoice](#), 479
[aiogram.methods.send_location](#), 402
[aiogram.methods.send_media_group](#), 405
[aiogram.methods.send_message](#), 408
[aiogram.methods.send_photo](#), 411
[aiogram.methods.send_poll](#), 413
[aiogram.methods.send_sticker](#), 322
[aiogram.methods.send_venue](#), 417
[aiogram.methods.send_video](#), 420
[aiogram.methods.send_video_note](#), 423
[aiogram.methods.send_voice](#), 426
[aiogram.methods.set_chat_administrator_custom_title](#), 428
[aiogram.methods.set_chat_description](#), 429
[aiogram.methods.set_chat_menu_button](#), 430
[aiogram.methods.set_chat_permissions](#), 432
[aiogram.methods.set_chat_photo](#), 433
[aiogram.methods.set_chat_sticker_set](#), 434
[aiogram.methods.set_chat_title](#), 435
[aiogram.methods.set_custom_emoji_sticker_set_thumbnail](#), 324
[aiogram.methods.set_game_score](#), 472
[aiogram.methods.set_message_reaction](#), 436
[aiogram.methods.set_my_commands](#), 438
[aiogram.methods.set_my_default_administrator_rights](#), 439
[aiogram.methods.set_my_description](#), 440
[aiogram.methods.set_my_name](#), 441
[aiogram.methods.set_my_short_description](#), 442
[aiogram.methods.set_passport_data_errors](#), 487
[aiogram.methods.set_sticker_emoji_list](#), 326
[aiogram.methods.set_sticker_keywords](#), 327
[aiogram.methods.set_sticker_mask_position](#), 328
[aiogram.methods.set_sticker_position_in_set](#), 329
[aiogram.methods.set_sticker_set_thumbnail](#), 330
[aiogram.methods.set_sticker_set_title](#), 331
[aiogram.methods.set_webhook](#), 485
[aiogram.methods.stop_message_live_location](#), 462
[aiogram.methods.stop_poll](#), 464
[aiogram.methods.unban_chat_member](#), 443
[aiogram.methods.unban_chat_sender_chat](#), 445
[aiogram.methods.unhide_general_forum_topic](#), 446
[aiogram.methods.unpin_all_chat_messages](#), 447
[aiogram.methods.unpin_all_forum_topic_messages](#), 448
[aiogram.methods.unpin_all_general_forum_topic_messages](#), 449
[aiogram.methods.unpin_chat_message](#), 450
[aiogram.methods.upload_sticker_file](#), 332
[aiogram.types.animation](#), 19
[aiogram.types.audio](#), 20
[aiogram.types.background_fill](#), 21
[aiogram.types.background_fill_freeform_gradient](#), 21
[aiogram.types.background_fill_gradient](#), 22
[aiogram.types.background_fill_solid](#), 22
[aiogram.types.background_type](#), 23
[aiogram.types.background_type_chat_theme](#), 23
[aiogram.types.background_type_fill](#), 23
[aiogram.types.background_type_pattern](#), 24
[aiogram.types.background_type_wallpaper](#), 25
[aiogram.types.birthdate](#), 25
[aiogram.types.bot_command](#), 26

aiogram.types.bot_command_scope, 26	aiogram.types.chat_member_updated, 110
aiogram.types.bot_command_scope_all_chat_administrators, 27	aiogram.types.chat_permissions, 130
aiogram.types.bot_command_scope_all_group_chats, 27	aiogram.types.chat_photo, 132
aiogram.types.bot_command_scope_all_private_chats, 28	aiogram.types.chat_shared, 132
aiogram.types.bot_command_scope_chat, 28	aiogram.types.chosen_inline_result, 240
aiogram.types.bot_command_scope_chat_administrator, 29	aiogram.types.contact, 133
aiogram.types.bot_command_scope_chat_member, 29	aiogram.types.dice, 134
aiogram.types.bot_command_scope_default, 30	aiogram.types.document, 134
aiogram.types.bot_description, 30	aiogram.types.encrypted_credentials, 293
aiogram.types.bot_name, 31	aiogram.types.encrypted_passport_element, 294
aiogram.types.bot_short_description, 31	aiogram.types.error_event, 573
aiogram.types.business_connection, 31	aiogram.types.external_reply_info, 135
aiogram.types.business_intro, 32	aiogram.types.file, 137
aiogram.types.business_location, 32	aiogram.types.force_reply, 138
aiogram.types.business_messages_deleted, 33	aiogram.types.forum_topic, 138
aiogram.types.business_opening_hours, 33	aiogram.types.forum_topic_closed, 139
aiogram.types.business_opening_hours_interval, 34	aiogram.types.forum_topic_created, 139
aiogram.types.callback_game, 313	aiogram.types.forum_topic_edited, 140
aiogram.types.callback_query, 34	aiogram.types.forum_topic_reopened, 140
aiogram.types.chat, 36	aiogram.types.game, 313
aiogram.types.chat_administrator_rights, 52	aiogram.types.game_high_score, 314
aiogram.types.chat_background, 54	aiogram.types.general_forum_topic_hidden, 140
aiogram.types.chat_boost, 54	aiogram.types.general_forum_topic_unhidden, 141
aiogram.types.chat_boost_added, 55	aiogram.types.giveaway, 141
aiogram.types.chat_boost_removed, 55	aiogram.types.giveaway_completed, 142
aiogram.types.chat_boost_source, 56	aiogram.types.giveaway_created, 142
aiogram.types.chat_boost_source_gift_code, 56	aiogram.types.giveaway_winners, 142
aiogram.types.chat_boost_source_giveaway, 56	aiogram.types.inaccessible_message, 143
aiogram.types.chat_boost_source_premium, 57	aiogram.types.inline_keyboard_button, 144
aiogram.types.chat_boost_updated, 58	aiogram.types.inline_keyboard_markup, 145
aiogram.types.chat_full_info, 58	aiogram.types.inline_query, 240
aiogram.types.chat_invite_link, 62	aiogram.types.inline_query_result, 242
aiogram.types.chat_join_request, 63	aiogram.types.inline_query_result_article, 243
aiogram.types.chat_location, 103	aiogram.types.inline_query_result_audio, 244
aiogram.types.chat_member, 104	aiogram.types.inline_query_result_cached_audio, 246
aiogram.types.chat_member_administrator, 105	aiogram.types.inline_query_result_cached_document, 248
aiogram.types.chat_member_banned, 107	aiogram.types.inline_query_result_cached_gif, 250
aiogram.types.chat_member_left, 107	aiogram.types.inline_query_result_cached_mpeg4_gif, 252
aiogram.types.chat_member_member, 108	aiogram.types.inline_query_result_cached_photo, 254
aiogram.types.chat_member_owner, 108	aiogram.types.inline_query_result_cached_sticker, 256
aiogram.types.chat_member_restricted, 109	aiogram.types.inline_query_result_cached_video, 258

[aiogram.types.inline_query_result_cached_voice](#), 260
[aiogram.types.inline_query_result_contact](#), 263
[aiogram.types.inline_query_result_document](#), 264
[aiogram.types.inline_query_result_game](#), 267
[aiogram.types.inline_query_result_gif](#), 267
[aiogram.types.inline_query_result_location](#), 270
[aiogram.types.inline_query_result_mpeg4_gif](#), 272
[aiogram.types.inline_query_result_photo](#), 274
[aiogram.types.inline_query_result_venue](#), 275
[aiogram.types.inline_query_result_video](#), 277
[aiogram.types.inline_query_result_voice](#), 280
[aiogram.types.inline_query_results_button](#), 281
[aiogram.types.input_contact_message_content](#), 282
[aiogram.types.input_file](#), 146
[aiogram.types.input_invoice_message_content](#), 282
[aiogram.types.input_location_message_content](#), 285
[aiogram.types.input_media](#), 146
[aiogram.types.input_media_animation](#), 147
[aiogram.types.input_media_audio](#), 148
[aiogram.types.input_media_document](#), 149
[aiogram.types.input_media_photo](#), 150
[aiogram.types.input_media_video](#), 151
[aiogram.types.input_message_content](#), 286
[aiogram.types.input_poll_option](#), 152
[aiogram.types.input_sticker](#), 289
[aiogram.types.input_text_message_content](#), 286
[aiogram.types.input_venue_message_content](#), 288
[aiogram.types.invoice](#), 304
[aiogram.types.keyboard_button](#), 153
[aiogram.types.keyboard_button_poll_type](#), 154
[aiogram.types.keyboard_button_request_chat](#), 155
[aiogram.types.keyboard_button_request_user](#), 156
[aiogram.types.keyboard_button_request_users](#), 157
[aiogram.types.labeled_price](#), 304
[aiogram.types.link_preview_options](#), 158
[aiogram.types.location](#), 159
[aiogram.types.login_url](#), 160
[aiogram.types.mask_position](#), 290
[aiogram.types.maybe_inaccessible_message](#), 160
[aiogram.types.menu_button](#), 161
[aiogram.types.menu_button_commands](#), 161
[aiogram.types.menu_button_default](#), 162
[aiogram.types.menu_button_web_app](#), 162
[aiogram.types.message](#), 163
[aiogram.types.message_auto_delete_timer_changed](#), 214
[aiogram.types.message_entity](#), 215
[aiogram.types.message_id](#), 216
[aiogram.types.message_origin](#), 216
[aiogram.types.message_origin_channel](#), 216
[aiogram.types.message_origin_chat](#), 217
[aiogram.types.message_origin_hidden_user](#), 218
[aiogram.types.message_origin_user](#), 218
[aiogram.types.message_reaction_count_updated](#), 219
[aiogram.types.message_reaction_updated](#), 219
[aiogram.types.order_info](#), 305
[aiogram.types.passport_data](#), 295
[aiogram.types.passport_element_error](#), 296
[aiogram.types.passport_element_error_data_field](#), 296
[aiogram.types.passport_element_error_file](#), 297
[aiogram.types.passport_element_error_files](#), 298
[aiogram.types.passport_element_error_front_side](#), 298
[aiogram.types.passport_element_error_reverse_side](#), 299
[aiogram.types.passport_element_error_selfie](#), 300
[aiogram.types.passport_element_error_translation_file](#), 300
[aiogram.types.passport_element_error_translation_files](#), 302
[aiogram.types.passport_element_error_unspecified](#), 302
[aiogram.types.passport_file](#), 303
[aiogram.types.photo_size](#), 220
[aiogram.types.poll](#), 221
[aiogram.types.poll_answer](#), 222
[aiogram.types.poll_option](#), 222
[aiogram.types.pre_checkout_query](#), 305

aiogram.types.proximity_alert_triggered, 223
 aiogram.types.reaction_count, 223
 aiogram.types.reaction_type, 224
 aiogram.types.reaction_type_custom_emoji, 224
 aiogram.types.reaction_type_emoji, 224
 aiogram.types.reply_keyboard_markup, 225
 aiogram.types.reply_keyboard_remove, 226
 aiogram.types.reply_parameters, 226
 aiogram.types.response_parameters, 228
 aiogram.types.sent_web_app_message, 289
 aiogram.types.shared_user, 228
 aiogram.types.shipping_address, 307
 aiogram.types.shipping_option, 307
 aiogram.types.shipping_query, 308
 aiogram.types.sticker, 290
 aiogram.types.sticker_set, 292
 aiogram.types.story, 229
 aiogram.types.successful_payment, 309
 aiogram.types.switch_inline_query_chosen_chat, 229
 aiogram.types.text_quote, 230
 aiogram.types.update, 309
 aiogram.types.user, 231
 aiogram.types.user_chat_boosts, 232
 aiogram.types.user_profile_photos, 233
 aiogram.types.user_shared, 233
 aiogram.types.users_shared, 234
 aiogram.types.venue, 234
 aiogram.types.video, 235
 aiogram.types.video_chat_ended, 236
 aiogram.types.video_chat_participants_invited, 236
 aiogram.types.video_chat_scheduled, 236
 aiogram.types.video_chat_started, 237
 aiogram.types.video_note, 237
 aiogram.types.voice, 238
 aiogram.types.web_app_data, 238
 aiogram.types.web_app_info, 239
 aiogram.types.webhook_info, 312
 aiogram.types.write_access_allowed, 239
 month (aiogram.types.birthdate.Birthdate ampубym), 25
 MOUTH (aiogram.enums.mask_position_point.MaskPositionPoint ampубym), 498
 mpeg4_duration (aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif ampубym), 273
 mpeg4_file_id (aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif ampубym), 254
 MPEG4_GIF (aiogram.enums.inline_query_result_type.InlineQueryResultType ampубym), 497
 mpeg4_height (aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif ampубym), 273
 mpeg4_url (aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif ampубym), 273
 mpeg4_width (aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif ampубym), 273
 MUR (aiogram.enums.currency.Currency ampубym), 495
 MVR (aiogram.enums.currency.Currency ampубym), 495
 MXN (aiogram.enums.currency.Currency ampубym), 495
 MY_CHAT_MEMBER (aiogram.enums.update_type.UpdateType ampубym), 503
 my_chat_member (aiogram.types.update.UpdateType ampубym), 311
 MYR (aiogram.enums.currency.Currency ampубym), 495
 MZN (aiogram.enums.currency.Currency ampубym), 495
 N
 name (aiogram.methods.add_sticker_to_set.AddStickerToSet ampубym), 315
 name (aiogram.methods.create_chat_invite_link.CreateChatInviteLink ampубym), 346
 name (aiogram.methods.create_forum_topic.CreateForumTopic ampубym), 347
 name (aiogram.methods.create_new_sticker_set.CreateNewStickerSet ampубym), 316
 name (aiogram.methods.delete_sticker_set.DeleteStickerSet ampубym), 318
 name (aiogram.methods.edit_chat_invite_link.EditChatInviteLink ampубym), 354
 name (aiogram.methods.edit_forum_topic.EditForumTopic ampубym), 355
 name (aiogram.methods.edit_general_forum_topic.EditGeneralForumTopic ampубym), 357
 name (aiogram.methods.get_sticker_set.GetStickerSet ampубym), 320
 name (aiogram.methods.replace_sticker_in_set.ReplaceStickerInSet ampубym), 321
 name (aiogram.methods.set_custom_emoji_sticker_set_thumbnail.SetCustomEmojiStickerSetThumbnail ampубym), 325
 name (aiogram.methods.set_my_name.SetMyName ampубym), 441
 name (aiogram.methods.set_sticker_set_thumbnail.SetStickerSetThumbnail ampубym), 330
 name (aiogram.methods.set_sticker_set_title.SetStickerSetTitle ampубym), 331
 name (aiogram.methods.set_bot_name.BotName ampубym), 31

name (*aiogram.types.chat_invite_link.ChatInviteLink* *ampu6ym*), 167
name (*aiogram.types.forum_topic.ForumTopic* *ampu6ym*), 138
name (*aiogram.types.forum_topic_created.ForumTopicCreated* *ampu6ym*), 139
name (*aiogram.types.forum_topic_edited.ForumTopicEdited* *ampu6ym*), 140
name (*aiogram.types.order_info.OrderInfo* *ampu6ym*), 305
name (*aiogram.types.sticker_set.StickerSet* *ampu6ym*), 292
need_email (*aiogram.methods.create_invoice_link.CreateInvoiceLink* *ampu6ym*), 478
need_email (*aiogram.methods.send_invoice.SendInvoice* *ampu6ym*), 481
need_email (*aiogram.types.input_invoice_message_content.InputInvoiceMessageContent* *ampu6ym*), 285
need_name (*aiogram.methods.create_invoice_link.CreateInvoiceLink* *ampu6ym*), 478
need_name (*aiogram.methods.send_invoice.SendInvoice* *ampu6ym*), 481
need_name (*aiogram.types.input_invoice_message_content.InputInvoiceMessageContent* *ampu6ym*), 284
need_phone_number (*aiogram.methods.create_invoice_link.CreateInvoiceLink* *ampu6ym*), 478
need_phone_number (*aiogram.methods.send_invoice.SendInvoice* *ampu6ym*), 481
need_phone_number (*aiogram.types.input_invoice_message_content.InputInvoiceMessageContent* *ampu6ym*), 285
need_shipping_address (*aiogram.methods.create_invoice_link.CreateInvoiceLink* *ampu6ym*), 478
need_shipping_address (*aiogram.methods.send_invoice.SendInvoice* *ampu6ym*), 481
need_shipping_address (*aiogram.types.input_invoice_message_content.InputInvoiceMessageContent* *ampu6ym*), 285
needs_repainting (*aiogram.methods.create_new_sticker_set.CreateNewStickerSet* *ampu6ym*), 316
needs_repainting (*aiogram.types.sticker.Sticker* *ampu6ym*), 291
new_chat_member (*aiogram.types.chat_member_updated.ChatMemberUpdated* *ampu6ym*), 111
NEW_CHAT_MEMBERS (*aiogram.enums.content_type.ContentType* *ampu6ym*), 492
new_chat_members (*aiogram.types.message.Message* *ampu6ym*), 167
NEW_CHAT_PHOTO (*aiogram.enums.content_type.ContentType* *ampu6ym*), 492
new_chat_photo (*aiogram.types.message.Message* *ampu6ym*), 167
NEW_CHAT_TITLE (*aiogram.enums.content_type.ContentType* *ampu6ym*), 492
new_chat_title (*aiogram.types.message.Message* *ampu6ym*), 167
new_reaction (*aiogram.types.message_reaction_updated.MessageReactionUpdated* *ampu6ym*), 220
next_offset (*aiogram.methods.answer_inline_query.AnswerInlineQuery* *ampu6ym*), 467
NGN (*aiogram.enums.currency.Currency* *ampu6ym*), 495
NIO (*aiogram.enums.currency.Currency* *ampu6ym*), 495
NOK (*aiogram.enums.currency.Currency* *ampu6ym*), 495
NPR (*aiogram.enums.currency.Currency* *ampu6ym*), 495
NZD (*aiogram.enums.currency.Currency* *ampu6ym*), 495
offset (*aiogram.methods.get_updates.GetUpdates* *ampu6ym*), 483
offset (*aiogram.methods.get_user_profile_photos.GetUserProfilePhotos* *ampu6ym*), 375
offset (*aiogram.methods.answer_inline_query.InlineQuery* *ampu6ym*), 240
offset (*aiogram.types.message_entity.MessageEntity* *ampu6ym*), 215
ok (*aiogram.methods.answer_pre_checkout_query.AnswerPreCheckoutQuery* *ampu6ym*), 474
ok (*aiogram.methods.answer_shipping_query.AnswerShippingQuery* *ampu6ym*), 475
old_chat_member (*aiogram.types.chat_member_updated.ChatMemberUpdated* *ampu6ym*), 111
old_reaction (*aiogram.types.message_reaction_updated.MessageReactionUpdated* *ampu6ym*), 220
old_sticker (*aiogram.methods.replace_sticker_in_set.ReplaceStickerInSet* *ampu6ym*), 321
one_time_keyboard (*aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup* *ampu6ym*), 225
only_if_banned (*aiogram.methods.unban_chat_member.UnbanChatMember* *ampu6ym*), 444
only_new_members (*aiogram.types.giveaway.Giveaway* *ampu6ym*), 444

[141](#)
[only_new_members](#) ([aiogram.types.giveaway_winners.GiveawayWinner](#) [ampubym](#)), [143](#)
[open_period](#) ([aiogram.methods.send_poll.SendPoll](#) [ampubym](#)), [415](#)
[open_period](#) ([aiogram.types.poll.Poll](#) [ampubym](#)), [221](#)
[opening_hours](#) ([aiogram.types.business_opening_hours.BusinessOpeningHours](#) [ampubym](#)), [33](#)
[opening_minute](#) ([aiogram.types.business_opening_hours_interval.BusinessOpeningHoursInterval](#) [ampubym](#)), [34](#)
[option_ids](#) ([aiogram.types.poll_answer.PollAnswer](#) [ampubym](#)), [222](#)
[options](#) ([aiogram.methods.send_poll.SendPoll](#) [ampubym](#)), [414](#)
[options](#) ([aiogram.types.poll.Poll](#) [ampubym](#)), [221](#)
[order_info](#) ([aiogram.types.pre_checkout_query.PreCheckoutQuery](#) [ampubym](#)), [306](#)
[order_info](#) ([aiogram.types.successful_payment.SuccessfulPayment](#) [ampubym](#)), [309](#)
[OrderInfo](#) (клас в [aiogram.types.order_info](#)), [305](#)
[origin](#) ([aiogram.types.external_reply_info.ExternalReplyInfo](#) [ampubym](#)), [135](#)

P

[PAB](#) ([aiogram.enums.currency.Currency](#) [ampubym](#)), [495](#)
[pack\(\)](#) ([aiogram.filters.callback_data.CallbackData](#) [memod](#)), [525](#)
[parse_mode](#) ([aiogram.methods.copy_message.CopyMessage](#) [ampubym](#)), [343](#)
[parse_mode](#) ([aiogram.methods.edit_message_caption.EditMessageCaption](#) [ampubym](#)), [454](#)
[parse_mode](#) ([aiogram.methods.edit_message_text.EditMessageText](#) [ampubym](#)), [461](#)
[parse_mode](#) ([aiogram.methods.send_animation.SendAnimation](#) [ampubym](#)), [389](#)
[parse_mode](#) ([aiogram.methods.send_audio.SendAudio](#) [ampubym](#)), [392](#)
[parse_mode](#) ([aiogram.methods.send_document.SendDocument](#) [ampubym](#)), [401](#)
[parse_mode](#) ([aiogram.methods.send_message.SendMessage](#) [ampubym](#)), [408](#)
[parse_mode](#) ([aiogram.methods.send_photo.SendPhoto](#) [ampubym](#)), [411](#)
[parse_mode](#) ([aiogram.methods.send_video.SendVideo](#) [ampubym](#)), [421](#)
[parse_mode](#) ([aiogram.methods.send_voice.SendVoice](#) [ampubym](#)), [426](#)
[parse_mode](#) ([aiogram.types.inline_query_result_audio.InlineQueryResultAudio](#) [ampubym](#)), [245](#)
[parse_mode](#) ([aiogram.types.inline_query_result_cached_audio.InlineQueryResultCachedAudio](#) [ampubym](#)), [248](#)
[parse_mode](#) ([aiogram.types.inline_query_result_cached_document.InlineQueryResultCachedDocument](#) [ampubym](#)), [250](#)
[parse_mode](#) ([aiogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif](#) [ampubym](#)), [252](#)
[parse_mode](#) ([aiogram.types.inline_query_result_cached_mpeg4_gif.InlineQueryResultCachedMpeg4Gif](#) [ampubym](#)), [254](#)
[parse_mode](#) ([aiogram.types.inline_query_result_cached_photo.InlineQueryResultCachedPhoto](#) [ampubym](#)), [256](#)
[parse_mode](#) ([aiogram.types.inline_query_result_cached_video.InlineQueryResultCachedVideo](#) [ampubym](#)), [261](#)
[parse_mode](#) ([aiogram.types.inline_query_result_cached_voice.InlineQueryResultCachedVoice](#) [ampubym](#)), [262](#)
[parse_mode](#) ([aiogram.types.inline_query_result_document.InlineQueryResultDocument](#) [ampubym](#)), [266](#)
[parse_mode](#) ([aiogram.types.inline_query_result_gif.InlineQueryResultGif](#) [ampubym](#)), [269](#)
[parse_mode](#) ([aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif](#) [ampubym](#)), [273](#)
[parse_mode](#) ([aiogram.types.inline_query_result_photo.InlineQueryResultPhoto](#) [ampubym](#)), [275](#)
[parse_mode](#) ([aiogram.types.inline_query_result_video.InlineQueryResultVideo](#) [ampubym](#)), [279](#)
[parse_mode](#) ([aiogram.types.inline_query_result_voice.InlineQueryResultVoice](#) [ampubym](#)), [281](#)
[parse_mode](#) ([aiogram.types.input_media_animation.InputMediaAnimation](#) [ampubym](#)), [147](#)
[parse_mode](#) ([aiogram.types.input_media_audio.InputMediaAudio](#) [ampubym](#)), [149](#)
[parse_mode](#) ([aiogram.types.input_media_document.InputMediaDocument](#) [ampubym](#)), [150](#)
[parse_mode](#) ([aiogram.types.input_media_photo.InputMediaPhoto](#) [ampubym](#)), [151](#)
[parse_mode](#) ([aiogram.types.input_media_video.InputMediaVideo](#) [ampubym](#)), [152](#)
[parse_mode](#) ([aiogram.types.input_text_message_content.InputTextMessageContent](#) [ampubym](#)), [287](#)
[parse_webapp_init_data\(\)](#) (в модулі [aiogram.utils.web_app](#)), [594](#)
[ParseMode](#) (клас в [aiogram.enums.parse_mode](#)), [500](#)
[PASSPORT](#) ([aiogram.enums.encrypted_passport_element.EncryptedPassportElement](#) [ampubym](#)), [496](#)
[PASSPORT_DATA](#) ([aiogram.enums.content_type.ContentType](#) [ampubym](#)), [492](#)
[passport_data](#) ([aiogram.types.message.Message](#) [ampubym](#)), [168](#)
[PASSPORT_REGISTRATION](#) ([aiogram.enums.encrypted_passport_element.EncryptedPassportElement](#) [ampubym](#)), [497](#)
[PassportData](#) (клас в [aiogram.types.passport_data](#)), [295](#)
[PassportElementError](#) (клас в [aiogram.types.passport_element_error](#)), [295](#)

296		<i>ampubym</i>), 392
PassportElementErrorDataField (клас в aiogram.types.passport_element_error_data_field), 296	performer (aiogram.types.audio.Audio <i>ampubym</i>), 20	
PassportElementErrorFile (клас в aiogram.types.passport_element_error_file), 297	performer (aiogram.types.inline_query_result_audio.InlineQueryResultAudio <i>ampubym</i>), 246	
PassportElementErrorFiles (клас в aiogram.types.passport_element_error_files), 298	performer (aiogram.types.input_media_audio.InputMediaAudio <i>ampubym</i>), 149	
PassportElementErrorFrontSide (клас в aiogram.types.passport_element_error_front_side), 298	permissions (aiogram.methods.restrict_chat_member.RestrictChatMember <i>ampubym</i>), 385	
PassportElementErrorReverseSide (клас в aiogram.types.passport_element_error_reverse_side), 299	permissions (aiogram.methods.set_chat_permissions.SetChatPermissions <i>ampubym</i>), 432	
PassportElementErrorSelfie (клас в aiogram.types.passport_element_error_selfie), 300	permissions (aiogram.types.chat.Chat <i>ampubym</i>), 40	
PassportElementErrorTranslationFile (клас в aiogram.types.passport_element_error_translation_file), 300	permissions (aiogram.types.chat_full_info.ChatFullInfo <i>ampubym</i>), 61	
PassportElementErrorTranslationFiles (клас в aiogram.types.passport_element_error_translation_files), 302	personal_chat (aiogram.types.chat.Chat <i>ampubym</i>), 40	
PassportElementErrorType (клас в aiogram.enums.passport_element_error_type), 500	personal_chat (aiogram.types.chat_full_info.ChatFullInfo <i>ampubym</i>), 60	
PassportElementErrorUnspecified (клас в aiogram.types.passport_element_error_unspecified), 302	PERSONAL_DETAILS (aiogram.enums.encrypted_passport_element.EncryptedPassportElement <i>ampubym</i>), 496	
PassportFile (клас в aiogram.types.passport_file), 303	PHONE_NUMBER (aiogram.enums.encrypted_passport_element.EncryptedPassportElement <i>ampubym</i>), 497	
pattern (aiogram.filters.exception.ExceptionMessageFilter <i>ampubym</i>), 528	PHONE_NUMBER (aiogram.enums.message_entity_type.MessageEntityType <i>ampubym</i>), 499	
pay (aiogram.types.inline_keyboard_button.InlineKeyboardButton <i>ampubym</i>), 145	phone_number (aiogram.methods.send_contact.SendContact <i>ampubym</i>), 396	
payload (aiogram.methods.create_invoice_link.CreateInvoiceLink <i>ampubym</i>), 477	phone_number (aiogram.types.contact.Contact <i>ampubym</i>), 133	
payload (aiogram.methods.send_invoice.SendInvoice <i>ampubym</i>), 480	phone_number (aiogram.types.encrypted_passport_element.EncryptedPassportElement <i>ampubym</i>), 294	
payload (aiogram.types.input_invoice_message_content.InputInvoiceMessageContent <i>ampubym</i>), 284	phone_number (aiogram.types.inline_query_result_contact.InlineQueryResultContact <i>ampubym</i>), 263	
PEN (aiogram.enums.currency.Currency <i>ampubym</i>), 495	phone_number (aiogram.types.input_contact_message_content.InputContactMessageContent <i>ampubym</i>), 282	
pending_join_request_count (aiogram.types.chat_invite_link.ChatInviteLink <i>ampubym</i>), 63	phone_number (aiogram.types.order_info.OrderInfo <i>ampubym</i>), 305	
pending_update_count (aiogram.types.webhook_info.WebhookInfo <i>ampubym</i>), 312	PhoneNumber (клас в aiogram.utils.formatting), 606	
performer (aiogram.methods.send_audio.SendAudio <i>ampubym</i>), 133	PHOTO (aiogram.enums.content_type.ContentType <i>ampubym</i>), 491	
	PHOTO (aiogram.enums.inline_query_result_type.InlineQueryResultType <i>ampubym</i>), 497	
	PHOTO (aiogram.enums.input_media_type.InputMediaType <i>ampubym</i>), 498	
	photo (aiogram.methods.send_photo.SendPhoto <i>ampubym</i>), 411	
	photo (aiogram.methods.set_chat_photo.SetChatPhoto <i>ampubym</i>), 433	
	photo (aiogram.types.chat.Chat <i>ampubym</i>), 40	
	photo (aiogram.types.chat_full_info.ChatFullInfo <i>ampubym</i>), 60	
	photo (aiogram.types.chat_shared.ChatShared <i>ampubym</i>), 133	

photo (aiogram.types.external_reply_info.ExternalReplyInfo ampубym), 136	ogram.enums.content_type.ContentType ampубym), 492
photo (aiogram.types.game.Game ampубym), 313	pinned_message (aiogram.types.chat.Chat ampубym), 40
photo (aiogram.types.message.Message ampубym), 166	pinned_message (aiogram.types.chat_full_info.ChatFullInfo ampубym), 61
photo (aiogram.types.shared_user.SharedUser ampубym), 229	pinned_message (aiogram.types.chat_full_info.ChatFullInfo ampубym), 168
photo_file_id (aiogram.types.inline_query_result_content_type.Message ampубym), 256	PKR (aiogram.enums.currency.Currency ampубym), 495
photo_height (aiogram.methods.create_invoice_link.CreateInvoiceLink ampубym), 478	PLN (aiogram.enums.currency.Currency ampубym), 495
photo_height (aiogram.methods.send_invoice.SendInvoice ampубym), 481	point (aiogram.types.update.Update ampубym), 290
photo_height (aiogram.types.inline_query_result_photo.InlineQueryResultPhoto ampубym), 275	point (aiogram.types.update.Update ampубym), 290
photo_height (aiogram.types.input_invoice_message_content_type.MessageContent ampубym), 284	poll (aiogram.types.external_reply_info.ExternalReplyInfo ampубym), 137
photo_size (aiogram.methods.create_invoice_link.CreateInvoiceLink ampубym), 478	poll (aiogram.types.messages_group.Message ampубym), 167
photo_size (aiogram.methods.send_invoice.SendInvoice ampубym), 480	poll (aiogram.types.update.Update ampубym), 311
photo_size (aiogram.types.input_invoice_message_content_type.MessageContent ampубym), 284	POLL (клас в aiogram.types.poll), 221
photo_url (aiogram.methods.create_invoice_link.CreateInvoiceLink ampубym), 478	POLL_ANSWER (aiogram.enums.update_type.UpdateType ampубym), 503
photo_url (aiogram.methods.send_invoice.SendInvoice ampубym), 480	POLL_ANSWER (клас в aiogram.types.poll_answer), 222
photo_url (aiogram.types.inline_query_result_photo.InlineQueryResultPhoto ampубym), 274	POLL_OPTION (клас в aiogram.types.poll_option), 222
photo_url (aiogram.types.input_invoice_message_content_type.MessageContent ampубym), 284	POLL_TYPE (клас в aiogram.enums.poll_type), 501
photo_url (aiogram.types.input_invoice_message_content_type.MessageContent ampубym), 284	position (aiogram.methods.set_sticker_position_in_set.SetStickerPosition ampубym), 329
photo_url (aiogram.types.input_invoice_message_content_type.MessageContent ampубym), 284	position (aiogram.types.game_high_score.GameHighScore ampубym), 314
photo_url (aiogram.types.input_invoice_message_content_type.MessageContent ampубym), 284	position (aiogram.types.text_quote.TextQuote ampубym), 314
photo_width (aiogram.methods.create_invoice_link.CreateInvoiceLink ampубym), 478	post_code (aiogram.types.shipping_address.ShippingAddress ampубym), 314
photo_width (aiogram.methods.send_invoice.SendInvoice ampубym), 480	PRE (aiogram.enums.message_entity_type.MessageEntityType ampубym), 499
photo_width (aiogram.types.inline_query_result_photo.InlineQueryResultPhoto ampубym), 275	Pre (клас в aiogram.utils.formatting), 607
photo_width (aiogram.types.input_invoice_message_content_type.MessageContent ampубym), 284	PRE_CHECKOUT_QUERY (клас в aiogram.enums.update_type.UpdateType ampубym), 503
photos (aiogram.types.user_profile_photos.UserProfilePhotos ampубym), 233	pre_checkout_query (aiogram.types.update.Update ampубym), 311
PhotoSize (клас в aiogram.types.photo_size), 220	pre_checkout_query_id (aiogram.methods.answer_pre_checkout_query.AnswerPreCheckoutQuery ampубym), 474
PHP (aiogram.enums.currency.Currency ampубym), 495	PreCheckoutQuery (клас в aiogram.enums.update_type.UpdateType ampубym), 503
pin() (aiogram.types.message.Message метод), 213	
pin_message() (aiogram.types.chat.Chat метод), 47	
PinChatMessage (клас в aiogram.methods.pin_chat_message), 379	
PINNED_MESSAGE (клас в aiogram.enums.chat_type.ChatType ampубym), 40	

ogram.types.pre_checkout_query), 305
 prefer_large_media (aiogram.types.link_preview_options.LinkPreviewOptions), 159
 prefer_small_media (aiogram.types.link_preview_options.LinkPreviewOptions), 158
 prefix (aiogram.filters.command.CommandObject), 519
 PREMIUM (aiogram.enums.chat_boost_source_type.ChatBoostSourceType), 490
 premium_animation (aiogram.types.sticker.Sticker), 291
 premium_subscription_month_count (aiogram.types.giveaway.Giveaway), 141
 premium_subscription_month_count (aiogram.types.giveaway_winners.GiveawayWinners), 143
 prepare_value() (aiogram.client.session.base.BaseSession), 16
 prices (aiogram.methods.create_invoice_link.CreateInvoiceLink), 477
 prices (aiogram.methods.send_invoice.SendInvoice), 480
 prices (aiogram.types.input_invoice_message_content.InputInvoiceMessageContent), 284
 prices (aiogram.types.shipping_option.ShippingOption), 307
 PRIVATE (aiogram.enums.chat_type.ChatType), 491
 prize_description (aiogram.types.giveaway.Giveaway), 141
 prize_description (aiogram.types.giveaway_winners.GiveawayWinners), 143
 profileAccentColorId (aiogram.types.chat.Chat), 40
 profileAccentColorId (aiogram.types.chat_full_info.ChatFullInfo), 60
 profileBackgroundCustomEmojiId (aiogram.types.chat.Chat), 40
 profileBackgroundCustomEmojiId (aiogram.types.chat_full_info.ChatFullInfo), 60
 promote() (aiogram.types.chat.Chat), 48
 PromoteChatMember (aiogram.methods.promote_chat_member), 380
 protect_content (aiogram.methods.copy_message.CopyMessage), 343
 protect_content (aiogram.methods.forward_messages.ForwardMessages), 345
 protect_content (aiogram.methods.forward_message.ForwardMessage), 359
 protect_content (aiogram.methods.forward_messages.ForwardMessages), 361
 protect_content (aiogram.methods.send_animation.SendAnimation), 389
 protect_content (aiogram.methods.send_audio.SendAudio), 392
 protect_content (aiogram.methods.send_contact.SendContact), 396
 protect_content (aiogram.methods.send_dice.SendDice), 398
 protect_content (aiogram.methods.send_document.SendDocument), 401
 protect_content (aiogram.methods.send_game.SendGame), 471
 protect_content (aiogram.methods.send_invoice.SendInvoice), 481
 protect_content (aiogram.methods.send_location.SendLocation), 404
 protect_content (aiogram.methods.send_media_group.SendMediaGroup), 406
 protect_content (aiogram.methods.send_message.SendMessage), 409
 protect_content (aiogram.methods.send_photo.SendPhoto), 412
 protect_content (aiogram.methods.send_poll.SendPoll), 415
 protect_content (aiogram.methods.send_sticker.SendSticker), 323
 protect_content (aiogram.methods.send_venue.SendVenue), 418
 protect_content (aiogram.methods.send_video.SendVideo), 418

<code>ampubym</code>), 421	Q
<code>protect_content</code> (<code>aiogram.methods.send_video_note.SendVideoNote</code> <code>ampubym</code>), 424	<code>QAR</code> (<code>aiogram.enums.currency.Currency</code> <code>ampubym</code>), 495
<code>protect_content</code> (<code>aiogram.methods.send_voice.SendVoice</code> <code>ampubym</code>), 427	<code>query</code> (<code>aiogram.types.chosen_inline_result.ChosenInlineResult</code> <code>ampubym</code>), 240
<code>provider_data</code> (<code>aiogram.methods.create_invoice_link.CreateInvoiceLink</code> <code>ampubym</code>), 477	<code>query</code> (<code>aiogram.types.inline_query.InlineQuery</code> <code>ampubym</code>), 240
<code>provider_data</code> (<code>aiogram.methods.send_invoice.SendInvoice</code> <code>ampubym</code>), 480	<code>query</code> (<code>aiogram.types.switch_inline_query_chosen_chat.SwitchInlineQueryChosenChat</code> <code>ampubym</code>), 229
<code>provider_data</code> (<code>aiogram.types.input_invoice_message_content.InputInvoiceMessageContent</code> <code>ampubym</code>), 284	<code>query_id</code> (<code>aiogram.utils.web_app.WebAppInitData</code> <code>ampubym</code>), 595
<code>provider_payment_charge_id</code> (<code>aiogram.types.successful_payment.SuccessfulPayment</code> <code>ampubym</code>), 309	<code>question</code> (<code>aiogram.methods.send_poll.SendPoll</code> <code>ampubym</code>), 414
<code>provider_token</code> (<code>aiogram.methods.create_invoice_link.CreateInvoiceLink</code> <code>ampubym</code>), 477	<code>question</code> (<code>aiogram.types.poll.Poll</code> <code>ampubym</code>), 221
<code>provider_token</code> (<code>aiogram.methods.send_invoice.SendInvoice</code> <code>ampubym</code>), 480	<code>question_entities</code> (<code>aiogram.methods.send_poll.SendPoll</code> <code>ampubym</code>), 415
<code>provider_token</code> (<code>aiogram.types.input_invoice_message_content.InputInvoiceMessageContent</code> <code>ampubym</code>), 284	<code>question_entities</code> (<code>aiogram.types.poll.Poll</code> <code>ampubym</code>), 221
<code>proximity_alert_radius</code> (<code>aiogram.methods.edit_message_live_location.EditMessageLiveLocation</code> <code>ampubym</code>), 457	<code>question_parse_mode</code> (<code>aiogram.methods.send_poll.SendPoll</code> <code>ampubym</code>), 414
<code>proximity_alert_radius</code> (<code>aiogram.methods.send_location.SendLocation</code> <code>ampubym</code>), 404	<code>QUIZ</code> (<code>aiogram.enums.keyboard_button_poll_type_type.KeyboardButtonPollType</code> <code>ampubym</code>), 498
<code>proximity_alert_radius</code> (<code>aiogram.types.inline_query_result_location.InlineQueryResultLocation</code> <code>ampubym</code>), 271	<code>QUIZ</code> (<code>aiogram.enums.poll_type.PollType</code> <code>ampubym</code>), 501
<code>proximity_alert_radius</code> (<code>aiogram.types.input_location_message_content.InputLocationMessageContent</code> <code>ampubym</code>), 286	<code>quote</code> (<code>aiogram.types.message.Message</code> <code>ampubym</code>), 166
<code>proximity_alert_radius</code> (<code>aiogram.types.location.Location</code> <code>ampubym</code>), 159	<code>quote</code> (<code>aiogram.types.reply_parameters.ReplyParameters</code> <code>ampubym</code>), 227
<code>PROXIMITY_ALERT_TRIGGERED</code> (<code>aiogram.enums.content_type.ContentType</code> <code>ampubym</code>), 492	<code>quote_entities</code> (<code>aiogram.types.reply_parameters.ReplyParameters</code> <code>ampubym</code>), 227
<code>proximity_alert_triggered</code> (<code>aiogram.types.message.Message</code> <code>ampubym</code>), 169	<code>quote_parse_mode</code> (<code>aiogram.types.reply_parameters.ReplyParameters</code> <code>ampubym</code>), 227
<code>ProximityAlertTriggered</code> (клас в <code>aiogram.types.proximity_alert_triggered</code>), 223	<code>quote_position</code> (<code>aiogram.types.reply_parameters.ReplyParameters</code> <code>ampubym</code>), 227
<code>PYG</code> (<code>aiogram.enums.currency.Currency</code> <code>ampubym</code>), 495	R
<code>Python Enhancement Proposals</code> PEP 484, 3 PEP 492, 3	<code>react()</code> (<code>aiogram.types.message.Message</code> <code>метод</code>), 214
	<code>reaction</code> (<code>aiogram.methods.set_message_reaction.SetMessageReaction</code> <code>ampubym</code>), 437
	<code>ReactionCount</code> (клас в <code>aiogram.types.reaction_count</code>), 223
	<code>reactions</code> (<code>aiogram.types.message_reaction_count_updated.MessageReactionCountUpdated</code> <code>ampubym</code>), 219
	<code>ReactionType</code> (клас в <code>aiogram.types.reaction_type</code>), 224
	<code>ReactionTypeCustomEmoji</code> (клас в <code>aiogram.types.reaction_type_custom_emoji</code>), 224

ReactionTypeEmoji (класс в aiogram.types.reaction_type_emoji), 224	remove_keyboard (aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove ampuбym), 226
ReactionTypeType (класс в aiogram.enums.reaction_type_type), 501	REMOVED_CHAT_BOOST (aiogram.enums.update_type.UpdateType ampuбym), 503
read() (aiogram.types.input_file.BufferedInputFile memod), 146	removed_chat_boost (aiogram.types.update.Update ampuбym), 312
read() (aiogram.types.input_file.FSInputFile memod), 146	render() (aiogram.utils.formatting.Text memod), 605
read() (aiogram.types.input_file.InputFile memod), 146	RENTAL_AGREEMENT (aiogram.enums.encrypted_passport_element.EncryptedPasspo ampuбym), 497
receiver (aiogram.utils.web_app.WebAppInitData ampuбym), 595	ReopenForumTopic (класс в aiogram.methods.reopen_forum_topic), 383
RECORD_VIDEO (aiogram.enums.chat_action.ChatAction ampuбym), 490	ReopenGeneralForumTopic (класс в aiogram.methods.reopen_general_forum_topic), 384
record_video() (aiogram.utils.chat_action.ChatActionSender class method), 592	ReplaceStickerInSet (класс в aiogram.methods.replace_sticker_in_set), 321
RECORD_VIDEO_NOTE (aiogram.enums.chat_action.ChatAction ampuбym), 490	reply() (aiogram.types.message.Message memod), 188
record_video_note() (aiogram.utils.chat_action.ChatActionSender class method), 592	reply_animation() (aiogram.types.message.Message memod), 170
RECORD_VOICE (aiogram.enums.chat_action.ChatAction ampuбym), 490	reply_audio() (aiogram.types.message.Message memod), 173
record_voice() (aiogram.utils.chat_action.ChatActionSender class method), 592	reply_contact() (aiogram.types.message.Message memod), 176
RED (aiogram.enums.topic_icon_color.TopicIconColor ampuбym), 502	reply_dice() (aiogram.types.message.Message memod), 195
RedisStorage (класс в aiogram.fsm.storage.redis), 548	reply_document() (aiogram.types.message.Message memod), 177
regex_match (aiogram.filters.command.CommandObject ampuбym), 519	reply_game() (aiogram.types.message.Message memod), 180
register() (aiogram.fsm.scene.SceneRegistry memod), 565	reply_help() (aiogram.types.message.Message memod), 181
register() (aiogram.webhook.aiohttp_server.BaseRequestHandler memod), 532	reply_location() (aiogram.types.message.Message memod), 185
register() (aiogram.webhook.aiohttp_server.SimpleRequestHandler memod), 533	reply_markup (aiogram.methods.copy_message.CopyMessage ampuбym), 343
register() (aiogram.webhook.aiohttp_server.TokenBaseRequestHandler memod), 534	reply_keyboard_button_poll_type_edit_message_caption.EditMessage ampuбym), 455
REGULAR (aiogram.enums.keyboard_button_poll_type_type ampuбym), 498	reply_markup (aiogram.methods.edit_message_live_location.EditMe ampuбym), 457
REGULAR (aiogram.enums.poll_type.PollType ampuбym), 501	reply_markup (aiogram.methods.edit_message_media.EditMessageM ampuбym), 458
REGULAR (aiogram.enums.sticker_type.StickerType ampuбym), 502	reply_markup (aiogram.methods.edit_message_reply_markup.EditM ampuбym), 460
remove_caption (aiogram.methods.copy_messages.CopyMessages ampuбym), 345	reply_markup (aiogram.methods.edit_message_text.EditMessageText ampuбym), 461
remove_date (aiogram.types.chat_boost_removed.ChatBoostRemoved ampuбym), 461	

`reply_markup(aiogram.methods.send_animation.SendAnimationReplyMarkup(aiogram.types.inline_query_result_cached_voice.InlineQueryResultCachedVoice), 390`
`reply_markup(aiogram.methods.send_audio.SendAudioReplyMarkup(aiogram.types.inline_query_result_contact.InlineQueryResultContact), 392`
`reply_markup(aiogram.methods.send_contact.SendContactReplyMarkup(aiogram.types.inline_query_result_document.InlineQueryResultDocument), 396`
`reply_markup(aiogram.methods.send_dice.SendDiceReplyMarkup(aiogram.types.inline_query_result_game.InlineQueryResultGame), 398`
`reply_markup(aiogram.methods.send_document.SendDocumentReplyMarkup(aiogram.types.inline_query_result_gif.InlineQueryResultGif), 401`
`reply_markup(aiogram.methods.send_game.SendGameReplyMarkup(aiogram.types.inline_query_result_location.InlineQueryResultLocation), 471`
`reply_markup(aiogram.methods.send_invoice.SendInvoiceReplyMarkup(aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif), 481`
`reply_markup(aiogram.methods.send_location.SendLocationReplyMarkup(aiogram.types.inline_query_result_photo.InlineQueryResultPhoto), 404`
`reply_markup(aiogram.methods.send_message.SendMessageReplyMarkup(aiogram.types.inline_query_result_venue.InlineQueryResultVenue), 409`
`reply_markup(aiogram.methods.send_photo.SendPhotoReplyMarkup(aiogram.types.inline_query_result_video.InlineQueryResultVideo), 412`
`reply_markup(aiogram.methods.send_poll.SendPollReplyMarkup(aiogram.types.inline_query_result_voice.InlineQueryResultVoice), 415`
`reply_markup(aiogram.methods.send_sticker.SendStickerReplyMarkup(aiogram.types.message.Message), 323`
`reply_markup(aiogram.methods.send_venue.SendVenueReplyMediaGroup(aiogram.types.message.Message, 418`
`reply_markup(aiogram.methods.send_video.SendVideoReplyParameters(aiogram.types.message.Message, 421`
`reply_markup(aiogram.methods.send_video_note.SendVideoNoteCopyMessage(aiogram.methods.copy_message.CopyMessage, 424`
`reply_markup(aiogram.methods.send_voice.SendVoiceReplyParameters(aiogram.methods.send_animation.SendAnimation, 427`
`reply_markup(aiogram.methods.stop_message_live_location.StopMessageLiveLocationReplyParameters(aiogram.methods.send_audio.SendAudio, 463`
`reply_markup(aiogram.methods.stop_poll.StopPollReplyParameters(aiogram.methods.send_audio.SendAudio, 464`
`reply_markup(aiogram.types.inline_query_result_article.InlineQueryResultArticle, 243`
`reply_markup(aiogram.types.inline_query_result_audio.InlineQueryResultAudio, 246`
`reply_markup(aiogram.types.inline_query_result_cached_audio.InlineQueryResultCachedAudio, 248`
`reply_markup(aiogram.types.inline_query_result_cached_document.InlineQueryResultCachedDocument, 250`
`reply_markup(aiogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif, 252`
`reply_markup(aiogram.types.inline_query_result_cached_mpeg4_gif.InlineQueryResultCachedMpeg4Gif, 254`
`reply_markup(aiogram.types.inline_query_result_cached_photo.InlineQueryResultCachedPhoto, 256`
`reply_markup(aiogram.types.inline_query_result_cached_sticker.InlineQueryResultCachedSticker, 258`
`reply_markup(aiogram.types.inline_query_result_cached_video.InlineQueryResultCachedVideo, 260`

<code>reply_parameters</code> (<i>aiogram.methods.send_media_group.SendMessageGroupParameters</i>), 406	<code>ogram.methods.send_game.SendGameParameters), 471</code>
<code>reply_parameters</code> (<i>aiogram.methods.send_message.SendMessageParameters</i>), 409	<code>reply_to_message_id</code> (<i>aiogram.methods.send_invoice.SendInvoiceParameters</i>), 481
<code>reply_parameters</code> (<i>aiogram.methods.send_photo.SendPhotoParameters</i>), 412	<code>reply_to_message_id</code> (<i>aiogram.methods.send_location.SendLocationParameters</i>), 404
<code>reply_parameters</code> (<i>aiogram.methods.send_poll.SendPollParameters</i>), 415	<code>reply_to_message_id</code> (<i>aiogram.methods.send_media_group.SendMessageGroupParameters</i>), 406
<code>reply_parameters</code> (<i>aiogram.methods.send_sticker.SendStickerParameters</i>), 323	<code>reply_to_message_id</code> (<i>aiogram.methods.send_message.SendMessageParameters</i>), 409
<code>reply_parameters</code> (<i>aiogram.methods.send_venue.SendVenueParameters</i>), 418	<code>reply_to_message_id</code> (<i>aiogram.methods.send_photo.SendPhotoParameters</i>), 412
<code>reply_parameters</code> (<i>aiogram.methods.send_video.SendVideoParameters</i>), 421	<code>reply_to_message_id</code> (<i>aiogram.methods.send_poll.SendPollParameters</i>), 416
<code>reply_parameters</code> (<i>aiogram.methods.send_video_note.SendVideoNoteParameters</i>), 424	<code>reply_to_message_id</code> (<i>aiogram.methods.send_sticker.SendStickerParameters</i>), 323
<code>reply_parameters</code> (<i>aiogram.methods.send_voice.SendVoiceParameters</i>), 427	<code>reply_to_message_id</code> (<i>aiogram.methods.send_venue.SendVenueParameters</i>), 418
<code>reply_photo()</code> (<i>aiogram.types.message.Message</i> метод), 190	<code>reply_to_message_id</code> (<i>aiogram.methods.send_video.SendVideoParameters</i>), 421
<code>reply_poll()</code> (<i>aiogram.types.message.Message</i> метод), 192	<code>reply_to_message_id</code> (<i>aiogram.methods.send_video_note.SendVideoNoteParameters</i>), 424
<code>reply_sticker()</code> (<i>aiogram.types.message.Message</i> метод), 196	<code>reply_to_message_id</code> (<i>aiogram.methods.send_voice.SendVoiceParameters</i>), 427
<code>reply_to_message</code> (<i>aiogram.types.message.Message</i> атрибут), 165	<code>reply_to_story</code> (<i>aiogram.types.message.Message</i> атрибут), 166
<code>reply_to_message_id</code> (<i>aiogram.methods.copy_message.CopyMessageParameters</i>), 343	<code>reply_venue()</code> (<i>aiogram.types.message.Message</i> метод), 198
<code>reply_to_message_id</code> (<i>aiogram.methods.send_animation.SendAnimationParameters</i>), 390	<code>reply_video()</code> (<i>aiogram.types.message.Message</i> метод), 200
<code>reply_to_message_id</code> (<i>aiogram.methods.send_audio.SendAudioParameters</i>), 393	<code>reply_video_note()</code> (<i>aiogram.types.message.Message</i> метод), 203
<code>reply_to_message_id</code> (<i>aiogram.methods.send_contact.SendContactParameters</i>), 396	<code>reply_voice()</code> (<i>aiogram.types.message.Message</i> метод), 205
<code>reply_to_message_id</code> (<i>aiogram.methods.send_dice.SendDiceParameters</i>), 399	<code>ReplyKeyboardBuilder</code> (клас в <i>aiogram.utils.keyboard</i>), 585
<code>reply_to_message_id</code> (<i>aiogram.methods.send_document.SendDocumentParameters</i>), 401	<code>ReplyKeyboardMarkup</code> (клас в <i>aiogram.types.reply_keyboard_markup</i>), 225
<code>reply_to_message_id</code> (<i>aiogram.methods.send_game.SendGameParameters</i>), 471	<code>ReplyKeyboardRemove</code> (клас в <i>aiogram.types.reply_keyboard_remove</i>), 226

ReplyParameters	(класс в aiogram.types.reply_parameters), 226	resolve_bot() (aiogram.webhook.aiohttp_server.BaseRequestHandler мемод), 533
request_chat	(aiogram.types.keyboard_button.KeyboardButton мемод), 153	resolve_bot() (aiogram.webhook.aiohttp_server.SimpleRequestHandler мемод), 533
request_contact	(aiogram.types.keyboard_button.KeyboardButton мемод), 153	resolve_bot() (aiogram.webhook.aiohttp_server.TokenBasedRequestHandler мемод), 534
request_id	(aiogram.types.chat_shared.ChatShared мемод), 132	resolve_used_update_types() (aiogram.dispatcher.router.Router мемод), 508
request_id	(aiogram.types.keyboard_button_request_ampubym), 155	ResponseParameters (класс в aiogram.types.response_parameters), 228
request_id	(aiogram.types.keyboard_button_request_ampubym), 156	RestrictingTelegramRequestUser (класс в aiogram.types.chat.Chat мемод), 49
request_id	(aiogram.types.keyboard_button_request_ampubym), 157	RestrictingTelegramRequestUser (класс в aiogram.methods.restrict_chat_member), 385
request_id	(aiogram.types.user_shared.UserShared мемод), 233	RESTRICTED (aiogram.enums.chat_member_status.ChatMemberStatus мемод), 490
request_id	(aiogram.types.users_shared.UsersShared мемод), 234	result (aiogram.methods.answer_web_app_query.AnswerWebAppQuery мемод), 468
request_location	(aiogram.types.keyboard_button.KeyboardButton мемод), 154	result_id (aiogram.types.chosen_inline_result.ChosenInlineResult мемод), 240
request_name	(aiogram.types.keyboard_button_request_ampubym), 158	results (aiogram.methods.get_users_inline_query.AnswerInlineQuery мемод), 466
request_photo	(aiogram.types.keyboard_button_request_ampubym), 156	setUp (aiogram.dispatcher.router.Router мемод), 568
request_photo	(aiogram.types.keyboard_button_request_ampubym), 158	setUp (aiogram.dispatcher.router.Router мемод), 568
request_poll	(aiogram.types.keyboard_button.KeyboardButton мемод), 154	RESPONSE (aiogram.enums.passport_element_error_type.PassportElementErrorType мемод), 500
request_title	(aiogram.types.keyboard_button_request_ampubym), 156	reverse_keyboard_button_type (aiogram.methods.get_passport_element.EncryptedPassportElement мемод), 295
request_user	(aiogram.types.keyboard_button.KeyboardButton мемод), 154	revoke_invite_link() (aiogram.types.chat.Chat мемод), 42
request_username	(aiogram.types.keyboard_button_request_chat.KeyboardButton мемод), 156	revoke_messages (aiogram.methods.revoke_chat_member.BanChatMember мемод), 337
request_username	(aiogram.types.keyboard_button_request_users.KeyboardButton мемод), 158	RevokeChatInviteLink (класс в aiogram.methods.set_my_default_administrator_rights.SetMyDefaultAdministratorRights мемод), 439
request_users	(aiogram.types.keyboard_button.KeyboardButton мемод), 153	RON (aiogram.enums.currency.Currency мемод), 495
request_write_access	(aiogram.types.login_url.LoginUrl мемод), 160	ROSE (aiogram.enums.topic_icon_color.TopicIconColor мемод), 502
reset_data_on_enter	(aiogram.fsm.scene.SceneConfig мемод), 566	rotation_angle (aiogram.types.background_fill_gradient.BackgroundFillGradient мемод), 22
reset_history_on_enter	(aiogram.fsm.scene.SceneConfig мемод), 566	Router (класс в aiogram.dispatcher.router), 508
resize_keyboard	(aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup мемод), 225	row() (aiogram.utils.keyboard.InlineKeyboardBuilder мемод), 584
		row() (aiogram.utils.keyboard.ReplyKeyboardBuilder мемод), 586

RSD (<i>aiogram.enums.currency.Currency</i> <i>ampubym</i>), 495	send_email_to_provider (<i>aiogram.types.input_invoice_message_content.InputInvoiceMessageContent</i> <i>ampubym</i>), 285
RUB (<i>aiogram.enums.currency.Currency</i> <i>ampubym</i>), 495	send_phone_number_to_provider (<i>aiogram.methods.create_invoice_link.CreateInvoiceLink</i> <i>ampubym</i>), 478
run_polling() (<i>aiogram.dispatcher.dispatcher.Dispatcher</i> <i>memod</i>), 514	send_phone_number_to_provider (<i>aiogram.methods.send_invoice.SendInvoice</i> <i>ampubym</i>), 481
S	send_phone_number_to_provider (<i>aiogram.types.input_invoice_message_content.InputInvoiceMessageContent</i> <i>ampubym</i>), 285
safe_parse_webapp_init_data() (в модулі <i>aiogram.utils.web_app</i>), 594	SendAnimation (клас в <i>aiogram.methods.send_animation</i>), 388
SAR (<i>aiogram.enums.currency.Currency</i> <i>ampubym</i>), 495	SendAudio (клас в <i>aiogram.methods.send_audio</i>), 391
scale (<i>aiogram.types.mask_position.MaskPosition</i> <i>ampubym</i>), 290	SendChatAction (клас в <i>aiogram.methods.send_chat_action</i>), 394
Scene (клас в <i>aiogram.fsm.scene</i>), 564	SendContact (клас в <i>aiogram.methods.send_contact</i>), 395
SceneConfig (клас в <i>aiogram.fsm.scene</i>), 566	SendDice (клас в <i>aiogram.methods.send_dice</i>), 397
SceneException, 574	SendDocument (клас в <i>aiogram.methods.send_document</i>), 400
SceneRegistry (клас в <i>aiogram.fsm.scene</i>), 565	SENDER (<i>aiogram.enums.chat_type.ChatType</i> <i>ampubym</i>), 491
ScenesManager (клас в <i>aiogram.fsm.scene</i>), 566	sender_boost_count (<i>aiogram.types.message.Message</i> <i>ampubym</i>), 165
SceneWizard (клас в <i>aiogram.fsm.scene</i>), 567	sender_business_bot (<i>aiogram.types.message.Message</i> <i>ampubym</i>), 165
scope (<i>aiogram.methods.delete_my_commands.DeleteMyCommands</i> <i>ampubym</i>), 353	sender_chat (<i>aiogram.types.message.Message</i> <i>ampubym</i>), 165
scope (<i>aiogram.methods.get_my_commands.GetMyCommands</i> <i>ampubym</i>), 370	sender_chat (<i>aiogram.types.message_origin_chat.MessageOriginChat</i> <i>ampubym</i>), 217
scope (<i>aiogram.methods.set_my_commands.SetMyCommands</i> <i>ampubym</i>), 438	sender_chat_id (<i>aiogram.methods.ban_chat_sender_chat.BanChatSenderChat</i> <i>ampubym</i>), 338
score (<i>aiogram.methods.set_game_score.SetGameScore</i> <i>ampubym</i>), 473	sender_chat_id (<i>aiogram.methods.unban_chat_sender_chat.UnbanChatSenderChat</i> <i>ampubym</i>), 445
score (<i>aiogram.types.game_high_score.GameHighScore</i> <i>ampubym</i>), 314	sender_user (<i>aiogram.types.message_origin_user.MessageOriginUser</i> <i>ampubym</i>), 218
secret (<i>aiogram.types.encrypted_credentials.EncryptedCredentials</i> <i>ampubym</i>), 293	sender_user_name (<i>aiogram.types.message_origin_hidden_user.MessageOriginHiddenUser</i> <i>ampubym</i>), 218
secret_token (<i>aiogram.methods.set_webhook.SetWebhook</i> <i>ampubym</i>), 486	SendGame (клас в <i>aiogram.methods.send_game</i>), 470
SEK (<i>aiogram.enums.currency.Currency</i> <i>ampubym</i>), 495	SendInvoice (клас в <i>aiogram.methods.send_invoice</i>), 479
selective (<i>aiogram.types.force_reply.ForceReply</i> <i>ampubym</i>), 138	SendLocation (клас в <i>aiogram.methods.send_location</i>), 402
selective (<i>aiogram.types.reply_keyboard_markup.ReplyKeyboardMarkup</i> <i>ampubym</i>), 226	SendMediaGroup (клас в <i>aiogram.methods.send_media_group</i>), 405
selective (<i>aiogram.types.reply_keyboard_remove.ReplyKeyboardRemove</i> <i>ampubym</i>), 226	SendMessage (клас в <i>aiogram.methods.send_message</i>), 479
SELFIE (<i>aiogram.enums.passport_element_error_type.PassportElementErrorType</i> <i>ampubym</i>), 500	
selfie (<i>aiogram.types.encrypted_passport_element.EncryptedPassportElement</i> <i>ampubym</i>), 295	
send_copy() (<i>aiogram.types.message.Message</i> <i>memod</i>), 207	
send_email_to_provider (<i>aiogram.methods.create_invoice_link.CreateInvoiceLink</i> <i>ampubym</i>), 478	
send_email_to_provider (<i>aiogram.methods.send_invoice.SendInvoice</i> <i>ampubym</i>), 481	

`ogram.methods.send_message`), 408
`SendPhoto` (клас в `aiogram.methods.send_photo`), 411
`SendPoll` (клас в `aiogram.methods.send_poll`), 413
`SendSticker` (клас в `aiogram.methods.send_sticker`), 322
`SendVenue` (клас в `aiogram.methods.send_venue`), 417
`SendVideo` (клас в `aiogram.methods.send_video`), 420
`SendVideoNote` (клас в `aiogram.methods.send_video_note`), 423
`SendVoice` (клас в `aiogram.methods.send_voice`), 426
`SentWebAppMessage` (клас в `aiogram.types.sent_web_app_message`), 289
`set_administrator_custom_title()` (`aiogram.types.chat.Chat` метод), 47
`set_data()` (`aiogram.fsm.scene.SceneWizard` метод), 568
`set_data()` (`aiogram.fsm.storage.base.BaseStorage` метод), 550
`set_description()` (`aiogram.types.chat.Chat` метод), 50
`set_locale()` (`aiogram.utils.i18n.middleware.FSMI18nMiddleware` метод), 589
`set_name` (`aiogram.types.sticker.Sticker` атрибут), 291
`set_permissions()` (`aiogram.types.chat.Chat` метод), 47
`set_photo()` (`aiogram.types.chat.Chat` метод), 51
`set_position_in_set()` (`aiogram.types.sticker.Sticker` метод), 291
`set_state()` (`aiogram.fsm.storage.base.BaseStorage` метод), 550
`set_sticker_set()` (`aiogram.types.chat.Chat` метод), 45
`set_title()` (`aiogram.types.chat.Chat` метод), 51
`SetChatAdministratorCustomTitle` (клас в `aiogram.methods.set_chat_administrator_custom_title`), 329
`SetChatDescription` (клас в `aiogram.methods.set_chat_description`), 429
`SetChatMenuButton` (клас в `aiogram.methods.set_chat_menu_button`), 430
`SetChatPermissions` (клас в `aiogram.methods.set_chat_permissions`), 432
`SetChatPhoto` (клас в `aiogram.methods.set_chat_photo`), 433
`SetChatStickerSet` (клас в `aiogram.methods.set_chat_sticker_set`), 434
`SetChatTitle` (клас в `aiogram.methods.set_chat_title`), 435
`SetCustomEmojiStickerSetThumbnail` (клас в `aiogram.methods.set_custom_emoji_sticker_set_thumbnail`), 324
`SetGameScore` (клас в `aiogram.methods.set_game_score`), 472
`SetMessageReaction` (клас в `aiogram.methods.set_message_reaction`), 436
`SetMyCommands` (клас в `aiogram.methods.set_my_commands`), 438
`SetMyDefaultAdministratorRights` (клас в `aiogram.methods.set_my_default_administrator_rights`), 439
`SetMyDescription` (клас в `aiogram.methods.set_my_description`), 440
`SetMyName` (клас в `aiogram.methods.set_my_name`), 441
`SetMyShortDescription` (клас в `aiogram.methods.set_my_short_description`), 442
`SetPassportDataErrors` (клас в `aiogram.methods.set_passport_data_errors`), 487
`SetStickerEmojiList` (клас в `aiogram.methods.set_sticker_emoji_list`), 326
`SetStickerKeywords` (клас в `aiogram.methods.set_sticker_keywords`), 327
`SetStickerMaskPosition` (клас в `aiogram.methods.set_sticker_mask_position`), 328
`SetStickerPositionInSet` (клас в `aiogram.methods.set_sticker_position_in_set`), 329
`SetStickerSetThumbnail` (клас в `aiogram.methods.set_sticker_set_thumbnail`), 330
`SetStickerSetTitle` (клас в `aiogram.methods.set_sticker_set_title`), 331
`setup()` (`aiogram.utils.i18n.middleware.I18nMiddleware` метод), 589
`SetWebhook` (клас в `aiogram.methods.set_webhook`), 485
`SGD` (`aiogram.enums.currency.Currency` атрибут), 495
`SharedUser` (клас в `aiogram.types.shared_user`), 228

shifted_id (aiogram.types.chat.Chat property), 41	ogram.types.chat_full_info.ChatFullInfo
shipping_address (aiogram.types.order_info.OrderInfo ampuбym), 305	ampuбym), 61
shipping_address (aiogram.types.shipping_query.ShippingQuery ampuбym), 308	small_file_id (aiogram.types.chat_photo.ChatPhoto ampuбym), 132
shipping_option_id (aiogram.types.pre_checkout_query.PreCheckoutQuery ampuбym), 306	small_file_unique_id (aiogram.types.chat_photo.ChatPhoto ampuбym), 132
shipping_option_id (aiogram.types.successful_payment.SuccessfulPayment ampuбym), 309	source (aiogram.types.chat_boost.ChatBoost ampuбym), 54
shipping_options (aiogram.methods.answer_shipping_query.AnswerShippingQuery ampuбym), 475	source (aiogram.types.chat_boost_removed.ChatBoostRemoved ampuбym), 55
SHIPPING_QUERY (aiogram.enums.update_type.UpdateType ampuбym), 503	source (aiogram.types.chat_boost_source_gift_code.ChatBoostSourceGiftCode ampuбym), 56
shipping_query (aiogram.types.update.Update ampuбym), 311	source (aiogram.types.chat_boost_source_giveaway.ChatBoostSourceGiveaway ampuбym), 57
shipping_query_id (aiogram.methods.answer_shipping_query.AnswerShippingQuery ampuбym), 475	source (aiogram.types.chat_boost_source_premium.ChatBoostSourcePremium ampuбym), 57
ShippingAddress (класс в aiogram.types.shipping_address), 307	source (aiogram.types.passport_element_error_data_field.PassportElementErrorDataField ampuбym), 296
ShippingOption (класс в aiogram.types.shipping_option), 307	source (aiogram.types.passport_element_error_file.PassportElementErrorFile ampuбym), 297
ShippingQuery (класс в aiogram.types.shipping_query), 308	source (aiogram.types.passport_element_error_files.PassportElementErrorFiles ampuбym), 298
short_description (aiogram.methods.set_my_short_description.SetMyShortDescription ampuбym), 442	source (aiogram.types.passport_element_error_front_side.PassportElementErrorFrontSide ampuбym), 299
short_description (aiogram.types.bot_short_description.BotShortDescription ampuбym), 31	source (aiogram.types.passport_element_error_reverse_side.PassportElementErrorReverseSide ampuбym), 299
show_above_text (aiogram.types.link_preview_options.LinkPreviewOptions ampuбym), 159	source (aiogram.types.passport_element_error_selfie.PassportElementErrorSelfie ampuбym), 300
show_alert (aiogram.methods.answer_callback_query.AnswerCallbackQuery ampuбym), 334	source (aiogram.types.passport_element_error_translation_file.PassportElementErrorTranslationFile ampuбym), 301
show_alert (aiogram.utils.callback_answer.CallbackAnswer property), 601	source (aiogram.types.passport_element_error_translation_files.PassportElementErrorTranslationFiles ampuбym), 302
SimpleMiddleware (класс в aiogram.utils.middleware), 588	source (aiogram.types.passport_element_error_unspecified.PassportElementErrorUnspecified ampuбym), 303
SimpleRequestHandler (класс в aiogram.webhook.aihttp_server), 533	SPOILER (aiogram.enums.message_entity_type.MessageEntityType ampuбym), 499
SLOT_MACHINE (aiogram.enums.dice_emoji.DiceEmoji ampuбym), 496	Spoilers (класс в aiogram.utils.formatting), 607
SLOT_MACHINE (aiogram.types.dice.DiceEmoji ampuбym), 134	start_date (aiogram.types.video_chat_scheduled.VideoChatScheduled ampuбym), 236
slow_mode_delay (aiogram.types.chat.Chat ampuбym), 40	start_param (aiogram.utils.web_app.WebAppInitData ampuбym), 596
slow_mode_delay (aiogram.fsm.scene.SceneConfig ampuбym), 515	start_parameter (aiogram.methods.send_invoice.SendInvoice ampuбym), 480
	start_parameter (aiogram.types.inline_query_results_button.InlineQueryResultButton ampuбym), 281
	start_parameter (aiogram.types.invoice.Invoice ampuбym), 304
	start_polling() (aiogram.dispatcher.dispatcher.Dispatcher memod), 515
	state (aiogram.fsm.scene.SceneConfig ampuбym), 515

566
state (aiogram.types.shipping_address.ShippingAddress
ampubym), 307
STATIC (aiogram.enums.sticker_format.StickerFormat
ampubym), 501
status (aiogram.types.chat_member_administrator.ChatMemberAdministrator
ampubym), 105
status (aiogram.types.chat_member_banned.ChatMemberBanned
ampubym), 107
status (aiogram.types.chat_member_left.ChatMemberLeft
ampubym), 107
status (aiogram.types.chat_member_member.ChatMemberMember
ampubym), 108
status (aiogram.types.chat_member_owner.ChatMemberOwner
ampubym), 108
status (aiogram.types.chat_member_restricted.ChatMemberRestricted
ampubym), 109
STICKER (aiogram.enums.content_type.ContentType
ampubym), 491
STICKER (aiogram.enums.inline_query_result_type.InlineQueryResultType
ampubym), 497
sticker (aiogram.methods.add_sticker_to_set.AddStickerToSet
ampubym), 315
sticker (aiogram.methods.delete_sticker_from_set.DeleteStickerFromSet
ampubym), 317
sticker (aiogram.methods.replace_sticker_in_set.ReplaceStickerInSet
ampubym), 321
sticker (aiogram.methods.send_sticker.SendSticker
ampubym), 323
sticker (aiogram.methods.set_sticker_emoji_list.SetStickerEmojiList
ampubym), 326
sticker (aiogram.methods.set_sticker_keywords.SetStickerKeywords
ampubym), 327
sticker (aiogram.methods.set_sticker_mask_position.SetStickerMaskPosition
ampubym), 328
sticker (aiogram.methods.set_sticker_position_in_set.SetStickerPositionInSet
ampubym), 329
sticker (aiogram.methods.upload_sticker_file.UploadStickerFile
ampubym), 333
sticker (aiogram.types.business_intro.BusinessIntro
ampubym), 32
sticker (aiogram.types.external_reply_info.ExternalReplyInfo
ampubym), 136
sticker (aiogram.types.input_sticker.InputSticker
ampubym), 289
sticker (aiogram.types.message.Message
ampubym), 166
Sticker (класс в aiogram.types.sticker), 290
sticker_file_id (aiogram.types.inline_query_result_cached_sticker.InlineQueryResultCachedSticker
ampubym), 257
sticker_format (aiogram.methods.create_new_sticker_set.CreateNewStickerSet
ampubym), 316
sticker_format (aiogram.enums.sticker_format), 501
stickers (aiogram.methods.create_new_sticker_set.CreateNewStickerSet
ampubym), 316
sticker_set (aiogram.types.sticker_set.StickerSet
ampubym), 292
sticker_set_name (aiogram.types.chat.Chat
ampubym), 41
sticker_set_name (aiogram.types.sticker_set.StickerSet
ampubym), 292
sticker_set_name (класс в aiogram.enums.sticker_type), 292
sticker_type (класс в aiogram.enums.sticker_type), 292
stop_live_location() (aiogram.types.message.Message
метод), 211
stop_polling() (aiogram.dispatcher.dispatcher.Dispatcher
метод), 515
stop_message_live_location (класс в aiogram.methods.stop_message_live_location), 462
stop_polling (aiogram.methods.stop_poll), 464
STORY (aiogram.enums.content_type.ContentType
ampubym), 136
story (aiogram.types.external_reply_info.ExternalReplyInfo
ampubym), 136
story (aiogram.types.message.Message
ampubym), 166
Story (класс в aiogram.types.story), 229
story_content() (aiogram.client.session.base.BaseSession
метод), 16
street_line1 (aiogram.types.shipping_address.ShippingAddress
ampubym), 307
street_line2 (aiogram.types.shipping_address.ShippingAddress
ampubym), 307
STRIKETHROUGH (aiogram.enums.message_entity_type.MessageEntityType
ampubym), 199
Strikethrough (класс в aiogram.utils.formatting), 607
SUCCESSFUL_PAYMENT (aiogram.enums.content_type.ContentType
ampubym), 316

`ampubym)`, 492
`successful_payment` (aiogram.types.message.Message `ampubym)`, 168
`SuccessfulPayment` (клас в aiogram.types.successful_payment), 309
`suggested_tip_amounts` (aiogram.methods.create_invoice_link.CreateInvoiceLink `ampubym)`, 477
`suggested_tip_amounts` (aiogram.methods.send_invoice.SendInvoice `ampubym)`, 480
`suggested_tip_amounts` (aiogram.types.input_invoice_message_content.InputInvoiceMessageContent `ampubym)`, 284
`SUPERGROUP` (aiogram.enums.chat_type.ChatType `ampubym)`, 491
`SUPERGROUP_CHAT_CREATED` (aiogram.enums.content_type.ContentType `ampubym)`, 492
`supergroup_chat_created` (aiogram.types.message.Message `ampubym)`, 168
`supports_inline_queries` (aiogram.types.user.User `ampubym)`, 231
`supports_streaming` (aiogram.methods.send_video.SendVideo `ampubym)`, 421
`supports_streaming` (aiogram.types.input_media_video.InputMediaVideo `ampubym)`, 152
`switch_inline_query` (aiogram.types.inline_keyboard_button.InlineKeyboardButton `ampubym)`, 145
`switch_inline_query_chosen_chat` (aiogram.types.inline_keyboard_button.InlineKeyboardButton `ampubym)`, 145
`switch_inline_query_current_chat` (aiogram.types.inline_keyboard_button.InlineKeyboardButton `ampubym)`, 145
`switch_pm_parameter` (aiogram.methods.answer_inline_query.AnswerInlineQuery `ampubym)`, 467
`switch_pm_text` (aiogram.methods.answer_inline_query.AnswerInlineQuery `ampubym)`, 467
`SwitchInlineQueryChosenChat` (клас в aiogram.types.switch_inline_query_chosen_chat), 229

T
`telegram_payment_charge_id` (aiogram.types.successful_payment.SuccessfulPayment `ampubym)`, 309
`TelegramAPIError`, 574
`TelegramAPIServer` (клас в aiogram.client.telegram), 14
`TelegramBadRequest`, 574
`TelegramConflictError`, 574
`TelegramEntityTooLarge`, 574
`TelegramForbiddenError`, 574
`TelegramMigrateToChat`, 574
`TelegramNetworkError`, 574
`TelegramNotFound`, 574
`TelegramRetryAfter`, 574
`TelegramServerError`, 574
`TelegramUnauthorizedError`, 574
TEMPORARY REGISTRATION (aiogram.enums.encrypted_passport_element.EncryptedPassportElement `ampubym)`, 497
TEXT (aiogram.enums.content_type.ContentType `ampubym)`, 491
text (aiogram.filters.command.CommandObject `property)`, 519
text (aiogram.methods.answer_callback_query.AnswerCallbackQuery `ampubym)`, 334
text (aiogram.methods.edit_message_text.EditMessageText `ampubym)`, 461
text (aiogram.methods.send_message.SendMessage `ampubym)`, 408
text (aiogram.types.game.Game `ampubym)`, 314
text (aiogram.types.inline_keyboard_button.InlineKeyboardButton `ampubym)`, 144
text (aiogram.types.inline_query_results_button.InlineQueryResultsButton `ampubym)`, 281
text (aiogram.types.input_poll_option.InputPollOption `ampubym)`, 153
text (aiogram.types.keyboard_button.KeyboardButton `ampubym)`, 153
text (aiogram.types.menu_button.MenuButton `ampubym)`, 161
text (aiogram.types.menu_button_web_app.MenuButtonWebApp `ampubym)`, 162
text (aiogram.types.message.Message `ampubym)`, 166
text (aiogram.types.poll_option.PollOption `ampubym)`, 222
text (aiogram.types.text_quote.TextQuote `ampubym)`, 230
text (aiogram.utils.callback_answer.CallbackAnswer `property)`, 601
Text (клас в aiogram.utils.formatting), 605
text_entities (aiogram.types.game.Game `ampubym)`, 314
text_entities (aiogram.types.input_poll_option.InputPollOption `ampubym)`, 153
text_entities (aiogram.types.poll_option.PollOption `ampubym)`, 222

TEXT_LINK (aiogram.enums.message_entity_type.MessageEntityTypeLink), 266	thumb_height (aiogram.types.inline_query_result_document.InlineQueryResultDocument), 271
TEXT_MENTION (aiogram.enums.message_entity_type.MessageEntityTypeMention), 266	thumb_height (aiogram.types.inline_query_result_location.InlineQueryResultLocation), 271
text_parse_mode (aiogram.types.input_poll_option.InputPollOption), 153	thumb_height (aiogram.types.inline_query_result_venue.InlineQueryResultVenue), 277
TextLink (класс в aiogram.utils.formatting), 607	thumb_mime_type (aiogram.types.inline_query_result_gif.InlineQueryResultGif), 269
TextMention (класс в aiogram.utils.formatting), 607	thumb_name (aiogram.types.background_type_chat_theme.ChatTheme), 23
TextQuote (класс в aiogram.types.text_quote), 230	thumb_name (aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif), 273
THB (aiogram.enums.currency.Currency), 495	thumb_name (aiogram.types.inline_query_result_article.InlineQueryResultArticle), 244
theme_name (aiogram.types.background_type_chat_theme.ChatTheme), 23	thumb_name (aiogram.types.inline_query_result_contact.InlineQueryResultContact), 264
thumbnail (aiogram.methods.send_animation.SendAnimation), 389	thumb_name (aiogram.types.inline_query_result_document.InlineQueryResultDocument), 266
thumbnail (aiogram.methods.send_audio.SendAudio), 392	thumb_name (aiogram.types.inline_query_result_gif.InlineQueryResultGif), 268
thumbnail (aiogram.methods.send_document.SendDocument), 400	thumb_name (aiogram.types.inline_query_result_location.InlineQueryResultLocation), 271
thumbnail (aiogram.methods.send_video.SendVideo), 421	thumb_name (aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif), 273
thumbnail (aiogram.methods.send_video_note.SendVideoNote), 423	thumb_name (aiogram.types.inline_query_result_photo.InlineQueryResultPhoto), 274
thumbnail (aiogram.methods.set_sticker_set_thumbnail.SetStickerSetThumbnail), 330	thumb_name (aiogram.types.inline_query_result_venue.InlineQueryResultVenue), 277
thumbnail (aiogram.types.animation.Animation), 20	thumb_name (aiogram.types.inline_query_result_video.InlineQueryResultVideo), 279
thumbnail (aiogram.types.audio.Audio), 21	thumb_width (aiogram.types.inline_query_result_article.InlineQueryResultArticle), 244
thumbnail (aiogram.types.document.Document), 134	thumb_width (aiogram.types.inline_query_result_contact.InlineQueryResultContact), 264
thumbnail (aiogram.types.input_media_animation.InputMediaAnimation), 147	thumb_width (aiogram.types.inline_query_result_document.InlineQueryResultDocument), 266
thumbnail (aiogram.types.input_media_audio.InputMediaAudio), 149	thumb_width (aiogram.types.inline_query_result_location.InlineQueryResultLocation), 271
thumbnail (aiogram.types.input_media_document.InputMediaDocument), 150	thumb_width (aiogram.types.inline_query_result_venue.InlineQueryResultVenue), 277
thumbnail (aiogram.types.input_media_video.InputMediaVideo), 152	thumb_width (aiogram.types.inline_query_result_video.InlineQueryResultVideo), 279
thumbnail (aiogram.types.sticker.Sticker), 291	thumb_width (aiogram.types.business_opening_hours.BusinessOpeningHours), 33
thumbnail (aiogram.types.sticker_set.StickerSet), 292	timeout (aiogram.methods.get_updates.GetUpdates), 184
thumbnail (aiogram.types.video.Video), 235	title (aiogram.methods.create_invoice_link.CreateInvoiceLink), 184
thumbnail (aiogram.types.video_note.VideoNote), 237	
thumbnail_height (aiogram.types.inline_query_result_article.InlineQueryResultArticle), 244	
thumbnail_height (aiogram.types.inline_query_result_contact.InlineQueryResultContact), 264	

<code>ampubym)</code> , 477	<code>title (aiogram.types.input_invoice_message_content.InputInvoiceMessageContent</code> <code>ampubym)</code> , 284
<code>title (aiogram.methods.create_new_sticker_set.CreateNewStickerSet</code> <code>ampubym)</code> , 316	<code>title (aiogram.types.input_media_audio.InputMediaAudio</code> <code>ampubym)</code> , 149
<code>title (aiogram.methods.send_audio.SendAudio</code> <code>ampubym)</code> , 392	<code>title (aiogram.types.input_venue_message_content.InputVenueMessageContent</code> <code>ampubym)</code> , 288
<code>title (aiogram.methods.send_invoice.SendInvoice</code> <code>ampubym)</code> , 480	<code>title (aiogram.types.invoice.Invoice</code> <code>ampubym)</code> , 304
<code>title (aiogram.methods.send_venue.SendVenue</code> <code>ampubym)</code> , 417	<code>title (aiogram.types.shipping_option.ShippingOption</code> <code>ampubym)</code> , 307
<code>title (aiogram.methods.set_chat_title.SetChatTitle</code> <code>ampubym)</code> , 435	<code>title (aiogram.types.sticker_set.StickerSet</code> <code>ampubym)</code> , 292
<code>title (aiogram.methods.set_sticker_set_title.SetStickerSetTitle</code> <code>ampubym)</code> , 332	<code>title (aiogram.types.venue.Venue</code> <code>ampubym)</code> , 234
<code>title (aiogram.types.audio.Audio</code> <code>ampubym)</code> , 20	<code>title (aiogram.utils.web_app.WebAppChat</code> <code>ampubym)</code> , 597
<code>title (aiogram.types.business_intro.BusinessIntro</code> <code>ampubym)</code> , 32	TJS (<code>aiogram.enums.currency.Currency</code> <code>ampubym)</code> , 495
<code>title (aiogram.types.chat.Chat</code> <code>ampubym)</code> , 36	TokenBasedRequestHandler (клас в <code>aiogram.webhook.aihttp_server</code>), 533
<code>title (aiogram.types.chat_full_info.ChatFullInfo</code> <code>ampubym)</code> , 60	<code>top_color (aiogram.types.background_fill_gradient.BackgroundFillGradient</code> <code>ampubym)</code> , 22
<code>title (aiogram.types.chat_shared.ChatShared</code> <code>ampubym)</code> , 133	TopicIconColor (клас в <code>aiogram.enums.topic_icon_color</code>), 502
<code>title (aiogram.types.game.Game</code> <code>ampubym)</code> , 313	<code>total_amount (aiogram.types.invoice.Invoice</code> <code>ampubym)</code> , 304
<code>title (aiogram.types.inline_query_result_article.InlineQueryResultArticle</code> <code>ampubym)</code> , 243	<code>total_amount (aiogram.types.pre_checkout_query.PreCheckoutQuery</code> <code>ampubym)</code> , 306
<code>title (aiogram.types.inline_query_result_audio.InlineQueryResultAudio</code> <code>ampubym)</code> , 245	<code>total_amount (aiogram.types.successful_payment.SuccessfulPayment</code> <code>ampubym)</code> , 309
<code>title (aiogram.types.inline_query_result_cached_document.InlineQueryResultCachedDocument</code> <code>ampubym)</code> , 250	<code>total_reaction_count (aiogram.types.reaction_count.ReactionCount</code> <code>ampubym)</code> , 223
<code>title (aiogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif</code> <code>ampubym)</code> , 252	<code>total_reaction_count (aiogram.types.reaction_count.ReactionCount</code> <code>ampubym)</code> , 223
<code>title (aiogram.types.inline_query_result_cached_mpeg4_gif.InlineQueryResultCachedMpeg4Gif</code> <code>ampubym)</code> , 254	<code>total_reaction_count (aiogram.types.reaction_count.ReactionCount</code> <code>ampubym)</code> , 223
<code>title (aiogram.types.inline_query_result_cached_photo.InlineQueryResultCachedPhoto</code> <code>ampubym)</code> , 256	<code>total_reaction_count (aiogram.types.reaction_count.ReactionCount</code> <code>ampubym)</code> , 221
<code>title (aiogram.types.inline_query_result_cached_video.InlineQueryResultCachedVideo</code> <code>ampubym)</code> , 260	<code>total_reaction_count (aiogram.types.reaction_count.ReactionCount</code> <code>ampubym)</code> , 295
<code>title (aiogram.types.inline_query_result_cached_voice.InlineQueryResultCachedVoice</code> <code>ampubym)</code> , 262	TRANSLATION_FILE (клас в <code>aiogram.enums.passport_element_error_type.PassportElementErrorType</code>), 501
<code>title (aiogram.types.inline_query_result_document.InlineQueryResultDocument</code> <code>ampubym)</code> , 266	TRANSLATION_FILES (клас в <code>aiogram.enums.passport_element_error_type.PassportElementErrorType</code>), 501
<code>title (aiogram.types.inline_query_result_gif.InlineQueryResultGif</code> <code>ampubym)</code> , 269	<code>types.proximity_alert_triggered.ProximityAlertTriggered</code> <code>ampubym)</code> , 223
<code>title (aiogram.types.inline_query_result_location.InlineQueryResultLocation</code> <code>ampubym)</code> , 271	<code>types.proximity_alert_triggered.ProximityAlertTriggered</code> <code>ampubym)</code> , 223
<code>title (aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif</code> <code>ampubym)</code> , 273	<code>types.proximity_alert_triggered.ProximityAlertTriggered</code> <code>ampubym)</code> , 223
<code>title (aiogram.types.inline_query_result_photo.InlineQueryResultPhoto</code> <code>ampubym)</code> , 275	<code>types.proximity_alert_triggered.ProximityAlertTriggered</code> <code>ampubym)</code> , 223
<code>title (aiogram.types.inline_query_result_venue.InlineQueryResultVenue</code> <code>ampubym)</code> , 276	<code>types.proximity_alert_triggered.ProximityAlertTriggered</code> <code>ampubym)</code> , 223
<code>title (aiogram.types.inline_query_result_video.InlineQueryResultVideo</code> <code>ampubym)</code> , 279	<code>types.proximity_alert_triggered.ProximityAlertTriggered</code> <code>ampubym)</code> , 223
<code>title (aiogram.types.inline_query_result_voice.InlineQueryResultVoice</code> <code>ampubym)</code> , 280	<code>types.proximity_alert_triggered.ProximityAlertTriggered</code> <code>ampubym)</code> , 223

type (aiogram.types.background_fill_gradient.BackgroundFillGradient ampuбym), 263
 type (aiogram.types.background_fill_solid.BackgroundFillSolid ampuбym), 265
 type (aiogram.types.background_type_chat_theme.BackgroundTypeChatTheme ampuбym), 267
 type (aiogram.types.background_type_fill.BackgroundTypeFill ampuбym), 268
 type (aiogram.types.background_type_pattern.BackgroundTypePattern ampuбym), 270
 type (aiogram.types.background_type_wallpaper.BackgroundTypeWallpaper ampuбym), 273
 type (aiogram.types.bot_command_scope_all_chat_administrators.BotCommandScopeAllChatAdministrators ampuбym), 274
 type (aiogram.types.bot_command_scope_all_group_chats.BotCommandScopeAllGroupChats ampuбym), 276
 type (aiogram.types.bot_command_scope_all_private_chats.BotCommandScopeAllPrivateChats ampuбym), 278
 type (aiogram.types.bot_command_scope_chat.BotCommandScopeChat ampuбym), 280
 type (aiogram.types.bot_command_scope_chat_administrators.BotCommandScopeChatAdministrators ampuбym), 29
 type (aiogram.types.bot_command_scope_chat_member.BotCommandScopeChatMember ampuбym), 29
 type (aiogram.types.bot_command_scope_default.BotCommandScopeDefault ampuбym), 30
 type (aiogram.types.chat.Chat ampuбym), 36
 type (aiogram.types.chat_background.ChatBackground ampuбym), 54
 type (aiogram.types.chat_full_info.ChatFullInfo ampuбym), 59
 type (aiogram.types.encrypted_passport_element.EncryptedPassportElement ampuбym), 294
 type (aiogram.types.inline_query_result_article.InlineQueryResultArticle ampuбym), 243
 type (aiogram.types.inline_query_result_audio.InlineQueryResultAudio ampuбym), 245
 type (aiogram.types.inline_query_result_cached_audio.InlineQueryResultCachedAudio ampuбym), 247
 type (aiogram.types.inline_query_result_cached_document.InlineQueryResultCachedDocument ampuбym), 250
 type (aiogram.types.inline_query_result_cached_gif.InlineQueryResultCachedGif ampuбym), 251
 type (aiogram.types.inline_query_result_cached_mpeg4_gif.InlineQueryResultCachedMpeg4Gif ampuбym), 254
 type (aiogram.types.inline_query_result_cached_photo.InlineQueryResultCachedPhoto ampuбym), 256
 type (aiogram.types.inline_query_result_cached_sticker.InlineQueryResultCachedSticker ampuбym), 257
 type (aiogram.types.inline_query_result_cached_video.InlineQueryResultCachedVideo ampuбym), 260
 type (aiogram.types.inline_query_result_cached_voice.InlineQueryResultCachedVoice ampuбym), 261
 type (aiogram.types.inline_query_result_contact.InlineQueryResultContact ampuбym), 262
 type (aiogram.types.inline_query_result_document.InlineQueryResultDocument ampuбym), 263
 type (aiogram.types.inline_query_result_game.InlineQueryResultGame ampuбym), 267
 type (aiogram.types.inline_query_result_gif.InlineQueryResultGif ampuбym), 268
 type (aiogram.types.inline_query_result_location.InlineQueryResultLocation ampuбym), 270
 type (aiogram.types.inline_query_result_mpeg4_gif.InlineQueryResultMpeg4Gif ampuбym), 273
 type (aiogram.types.inline_query_result_photo.InlineQueryResultPhoto ampuбym), 274
 type (aiogram.types.inline_query_result_venue.InlineQueryResultVenue ampuбym), 276
 type (aiogram.types.inline_query_result_video.InlineQueryResultVideo ampuбym), 278
 type (aiogram.types.inline_query_result_voice.InlineQueryResultVoice ampuбym), 280
 type (aiogram.types.input_media_animation.InputMediaAnimation ampuбym), 281
 type (aiogram.types.input_media_audio.InputMediaAudio ampuбym), 282
 type (aiogram.types.input_media_document.InputMediaDocument ampuбym), 283
 type (aiogram.types.input_media_photo.InputMediaPhoto ampuбym), 284
 type (aiogram.types.input_media_video.InputMediaVideo ampuбym), 285
 type (aiogram.types.keyboard_button_poll_type.KeyboardButtonPollType ampuбym), 286
 type (aiogram.types.menu_button.MenuButton ampuбym), 287
 type (aiogram.types.menu_button_commands.MenuButtonCommands ampuбym), 288
 type (aiogram.types.menu_button_default.MenuButtonDefault ampuбym), 289
 type (aiogram.types.menu_button_web_app.MenuButtonWebApp ampuбym), 290
 type (aiogram.types.message.Message ampuбym), 291
 type (aiogram.types.message_origin_channel.MessageOriginChannel ampuбym), 292
 type (aiogram.types.message_origin_chat.MessageOriginChat ampuбym), 293
 type (aiogram.types.message_origin_hidden_user.MessageOriginHiddenUser ampuбym), 294
 type (aiogram.types.message_origin_user.MessageOriginUser ampuбym), 295
 type (aiogram.types.passport_element_error_data_field.PassportElementErrorDataField ampuбym), 296
 type (aiogram.types.passport_element_error_file.PassportElementErrorFile ampuбym), 297
 type (aiogram.types.passport_element_error_files.PassportElementErrorFiles ampuбym), 298

<code>ampubym</code>), 298	<code>UnhideGeneralForumTopic</code> (клас в <code>aiogram.methods.unhide_general_forum_topic</code>), 446
<code>type (aiogram.types.passport_element_error_front_side.PassportElementErrorFrontSide ampubym)</code> , 299	<code>UNKNOWN</code> (<code>aiogram.enums.chat_action.ChatAction</code>), 490
<code>type (aiogram.types.passport_element_error_reverse_side.PassportElementErrorReverseSide ampubym)</code> , 299	<code>UNKNOWN</code> (<code>aiogram.enums.chat_action.ChatAction</code>), 491
<code>type (aiogram.types.passport_element_error_selfie.PassportElementErrorSelfie ampubym)</code> , 300	<code>support_callback_data</code> (<code>aiogram.methods.support_callback_data.CallbackData</code> class method), 525
<code>type (aiogram.types.passport_element_error_translation_file.PassportElementErrorTranslationFile ampubym)</code> , 301	<code>upload_file</code> (<code>aiogram.methods.upload_file.UploadFile</code> class method), 213
<code>type (aiogram.types.passport_element_error_translation_file.PassportElementErrorTranslationFile ampubym)</code> , 302	<code>unpin_all_general_forum_topic_messages</code> (<code>aiogram.types.chat.Chat</code> method), 52
<code>type (aiogram.types.passport_element_error_unspecified_file.PassportElementErrorUnspecifiedFile ampubym)</code> , 303	<code>unpin_all_forum_topic_messages</code> (<code>aiogram.types.chat.Chat</code> method), 46
<code>type (aiogram.types.poll.Poll ampubym)</code> , 221	<code>unpin_message</code> (<code>aiogram.types.chat.Chat</code> method), 46
<code>type (aiogram.types.reaction_count.ReactionCount ampubym)</code> , 223	<code>UnpinAllChatMessages</code> (клас в <code>aiogram.methods.unpin_all_chat_messages</code>), 447
<code>type (aiogram.types.reaction_type_custom_emoji.ReactionTypeCustomEmoji ampubym)</code> , 224	<code>UnpinAllForumTopicMessages</code> (клас в <code>aiogram.methods.unpin_all_forum_topic_messages</code>), 448
<code>type (aiogram.types.reaction_type_emoji.ReactionTypeEmoji ampubym)</code> , 224	<code>UnpinAllGeneralForumTopicMessages</code> (клас в <code>aiogram.methods.unpin_all_general_forum_topic_messages</code>), 449
<code>type (aiogram.types.sticker.Sticker ampubym)</code> , 290	<code>UnpinChatMessage</code> (клас в <code>aiogram.methods.unpin_chat_message</code>), 450
<code>type (aiogram.utils.web_app.WebAppChat ampubym)</code> , 597	<code>unrestrict_boost_count</code> (<code>aiogram.types.chat.Chat</code> ampubym), 41
<code>TYPING</code> (<code>aiogram.enums.chat_action.ChatAction</code> ampubym), 490	<code>unrestrict_boost_count</code> (<code>aiogram.types.chat_full_info.ChatFullInfo</code> ampubym), 61
<code>typing</code> (<code>aiogram.utils.chat_action.ChatActionSender</code> class method), 592	<code>UNSPECIFIED</code> (<code>aiogram.enums.passport_element_error_type.PassportElementErrorType</code> ampubym), 501
<code>TZS</code> (<code>aiogram.enums.currency.Currency</code> ampubym), 495	<code>UnsupportedKeywordArgument</code> , 574
U	<code>until_date</code> (<code>aiogram.methods.ban_chat_member.BanChatMember</code> ampubym), 336
<code>UAH</code> (<code>aiogram.enums.currency.Currency</code> ampubym), 496	<code>until_date</code> (<code>aiogram.methods.restrict_chat_member.RestrictChatMember</code> ampubym), 386
<code>UGX</code> (<code>aiogram.enums.currency.Currency</code> ampubym), 496	<code>until_date</code> (<code>aiogram.types.chat_member_banned.ChatMemberBanned</code> ampubym), 107
<code>unban</code> (<code>aiogram.types.chat.Chat</code> method), 50	<code>until_date</code> (<code>aiogram.types.chat_member_restricted.ChatMemberRestricted</code> ampubym), 110
<code>unban_sender_chat</code> (<code>aiogram.types.chat.Chat</code> method), 41	<code>update</code> (<code>aiogram.types.error_event.ErrorEvent</code> ampubym), 573
<code>UnbanChatMember</code> (клас в <code>aiogram.methods.unban_chat_member</code>), 443	<code>update</code> (клас в <code>aiogram.types.update</code>), 309
<code>UnbanChatSenderChat</code> (клас в <code>aiogram.methods.unban_chat_sender_chat</code>), 445	<code>update_data</code> (<code>aiogram.fsm.scene.SceneWizard</code> method), 568
<code>unclaimed_prize_count</code> (<code>aiogram.types.giveaway_completed.GiveawayCompleted</code> ampubym), 142	<code>update_data</code> (<code>aiogram.fsm.storage.base.BaseStorage</code> method), 550
<code>unclaimed_prize_count</code> (<code>aiogram.types.giveaway_winners.GiveawayWinners</code> ampubym), 143	<code>update_notification_flags</code> (<code>aiogram.filters.base.Filter</code> method), 528
<code>UNDERLINE</code> (<code>aiogram.enums.message_entity_type.MessageEntityType</code> ampubym), 499	<code>update_id</code> (<code>aiogram.types.update.Update</code> ampubym), 310
<code>Underline</code> (клас в <code>aiogram.utils.formatting</code>), 607	

UpdateType (клас в aiogram.enums.update_type), 502	property), 601
UpdateTypeLookupError, 312	Url (клас в aiogram.utils.formatting), 606
UPLOAD_DOCUMENT (aiogram.enums.chat_action.ChatAction ampuбym), 490	URLInputFile (клас в aiogram.types.input_file), 146, 507
upload_document() (aiogram.utils.chat_action.ChatActionSender class method), 592	USD (aiogram.enums.currency.Currency ampuбym), 496
UPLOAD_PHOTO (aiogram.enums.chat_action.ChatAction ampuбym), 490	use_independent_chat_permissions (aiogram.methods.restrict_chat_member.RestrictChatMember ampuбym), 386
upload_photo() (aiogram.utils.chat_action.ChatActionSender class method), 592	use_independent_chat_permissions (aiogram.methods.set_chat_permissions.SetChatPermissions ampuбym), 432
UPLOAD_VIDEO (aiogram.enums.chat_action.ChatAction ampuбym), 490	USER (aiogram.enums.message_origin_type.MessageOriginType ampuбym), 500
upload_video() (aiogram.utils.chat_action.ChatActionSender class method), 592	user (aiogram.types.business_connection.BusinessConnection ampuбym), 31
UPLOAD_VIDEO_NOTE (aiogram.enums.chat_action.ChatAction ampuбym), 490	user (aiogram.types.chat_boost_source_gift_code.ChatBoostSourceG ampuбym), 56
upload_video_note() (aiogram.utils.chat_action.ChatActionSender class method), 592	user (aiogram.types.chat_boost_source_giveaway.ChatBoostSourceG ampuбym), 57
	user (aiogram.types.chat_boost_source_premium.ChatBoostSourcePr ampuбym), 58
	user (aiogram.types.chat_member_administrator.ChatMemberAdmin ampuбym), 105
	user (aiogram.types.chat_member_banned.ChatMemberBanned ampuбym), 107
	user (aiogram.types.chat_member_left.ChatMemberLeft ampuбym), 107
	user (aiogram.types.chat_member_member.ChatMemberMember ampuбym), 108
	user (aiogram.types.chat_member_owner.ChatMemberOwner ampuбym), 108
UploadStickerFile (клас в aiogram.methods.upload_sticker_file), 332	user (aiogram.types.chat_member_restricted.ChatMemberRestricted ampuбym), 109
URL (aiogram.enums.message_entity_type.MessageEntity ampuбym), 499	user (aiogram.types.game_high_score.GameHighScore ampuбym), 314
url (aiogram.methods.answer_callback_query.CallbackQuery ampuбym), 334	user (aiogram.types.message_entity.MessageEntity ampuбym), 215
url (aiogram.methods.set_webhook.SetWebhook ampuбym), 486	user (aiogram.types.message_reaction_updated.MessageReactionUpd ampuбym), 220
url (aiogram.types.inline_keyboard_button.InlineKeyboardButton ampuбym), 144	user (aiogram.types.poll_answer.PollAnswer ampuбym), 222
url (aiogram.types.inline_query_result_article.InlineQueryResultArticle ampuбym), 243	user (aiogram.utils.web_app.WebAppInitData ampuбym), 595
url (aiogram.types.link_preview_options.LinkPreviewOptions ampuбym), 158	User (клас в aiogram.types.user), 231
url (aiogram.types.login_url.LoginUrl ampuбym), 160	user_administrator_rights (aiogram.types.keyboard_button_request_chat.KeyboardButton ampuбym), 156
url (aiogram.types.message_entity.MessageEntity ampuбym), 215	user_chat_id (aiogram.types.business_connection.BusinessConnectio ampuбym), 31
url (aiogram.types.user.User property), 232	user_chat_id (aiogram.types.chat_join_request.ChatJoinRequest ampuбym), 63
url (aiogram.types.web_app_info.WebAppInfo ampuбym), 239	user_id (aiogram.methods.add_sticker_to_set.AddStickerToSet ampuбym), 315
url (aiogram.types.webhook_info.WebhookInfo ampuбym), 312	
url (aiogram.utils.callback_answer.CallbackAnswer	

<code>user_id</code> (<code>aiogram.methods.approve_chat_join_request</code> . <code>ApproveChatJoinRequest</code> <code>amplib</code>), 335	<code>USERS_SHARED</code> (<code>aiogram.enums.content_type.ContentType</code> <code>amplib</code>), 493
<code>user_id</code> (<code>aiogram.methods.ban_chat_member</code> . <code>BanChatMember</code> <code>amplib</code>), 336	<code>UserShared</code> (<code>aiogram.types.message.Message</code> <code>amplib</code>), 170
<code>user_id</code> (<code>aiogram.methods.create_new_sticker_set</code> . <code>CreateNewStickerSet</code> <code>amplib</code>), 316	<code>UserChatBoosts</code> (клас в <code>aiogram.types.user_chat_boosts</code>), 232
<code>user_id</code> (<code>aiogram.methods.decline_chat_join_request</code> . <code>DeclineChatJoinRequest</code> <code>amplib</code>), 348	<code>DeclineChatJoinRequest</code> (<code>aiogram.types.chat.Chat</code> <code>amplib</code>), 36
<code>user_id</code> (<code>aiogram.methods.get_chat_member</code> . <code>GetChatMember</code> <code>amplib</code>), 364	<code>username</code> (<code>aiogram.types.chat_full_info.ChatFullInfo</code> <code>amplib</code>), 60
<code>user_id</code> (<code>aiogram.methods.get_game_high_scores</code> . <code>GetGameHighScores</code> <code>amplib</code>), 470	<code>username</code> (<code>aiogram.types.chat_shared.ChatShared</code> <code>amplib</code>), 133
<code>user_id</code> (<code>aiogram.methods.get_user_chat_boosts</code> . <code>GetUserChatBoosts</code> <code>amplib</code>), 375	<code>username</code> (<code>aiogram.types.shared_user.SharedUser</code> <code>amplib</code>), 228
<code>user_id</code> (<code>aiogram.methods.get_user_profile_photos</code> . <code>GetUserProfilePhotos</code> <code>amplib</code>), 375	<code>username</code> (<code>aiogram.types.user.User</code> <code>amplib</code>), 231
<code>user_id</code> (<code>aiogram.methods.promote_chat_member</code> . <code>PromoteChatMember</code> <code>amplib</code>), 381	<code>user_id</code> (<code>aiogram.types.user_chat_boosts</code> <code>amplib</code>), 597
<code>user_id</code> (<code>aiogram.methods.replace_sticker_in_set</code> . <code>ReplaceStickerInSet</code> <code>amplib</code>), 321	<code>user_id</code> (<code>aiogram.types.chat_full_info.ChatFullInfo</code> <code>amplib</code>), 596
<code>user_id</code> (<code>aiogram.methods.restrict_chat_member</code> . <code>RestrictChatMember</code> <code>amplib</code>), 385	<code>UserProfilePhotos</code> (клас в <code>aiogram.types.user_profile_photos</code>), 233
<code>user_id</code> (<code>aiogram.methods.set_chat_administrator_custom_title</code> . <code>SetChatAdministratorCustomTitle</code> <code>amplib</code>), 428	<code>UsersShared</code> (<code>aiogram.types.users_shared.UsersShared</code> <code>amplib</code>), 234
<code>user_id</code> (<code>aiogram.methods.set_game_score</code> . <code>SetGameScore</code> <code>amplib</code>), 472	<code>UsersShared</code> (клас в <code>aiogram.types.users_shared</code>), 236
<code>user_id</code> (<code>aiogram.methods.set_passport_data_errors</code> . <code>SetPassportDataErrors</code> <code>amplib</code>), 488	<code>USERS_SHARED</code> (<code>aiogram.enums.content_type.ContentType</code> <code>amplib</code>), 492
<code>user_id</code> (<code>aiogram.methods.set_sticker_set_thumbnail</code> . <code>SetStickerSetThumbnail</code> <code>amplib</code>), 330	<code>UsersSharedDataError</code> (<code>aiogram.types.message.Message</code> <code>amplib</code>), 168
<code>user_id</code> (<code>aiogram.methods.unban_chat_member</code> . <code>UnbanChatMember</code> <code>amplib</code>), 443	<code>UserShared</code> (<code>aiogram.types.user_shared</code>), 233
<code>user_id</code> (<code>aiogram.methods.upload_sticker_file</code> . <code>UploadStickerFile</code> <code>amplib</code>), 332	<code>UsersShared</code> (клас в <code>aiogram.types.users_shared</code>), 234
<code>user_id</code> (<code>aiogram.types.bot_command_scope_chat_member</code> . <code>BotCommandScopeChatMember</code> <code>amplib</code>), 30	<code>UTILITY_BILL</code> (<code>aiogram.enums.encrypted_passport_element.EncryptedPassportElement</code> <code>amplib</code>), 497
<code>user_id</code> (<code>aiogram.types.contact</code> . <code>Contact</code> <code>amplib</code>), 133	<code>UYU</code> (<code>aiogram.enums.currency.Currency</code> <code>amplib</code>), 496
<code>user_id</code> (<code>aiogram.types.shared_user.SharedUser</code> <code>amplib</code>), 228	<code>UZS</code> (<code>aiogram.enums.currency.Currency</code> <code>amplib</code>), 496
<code>user_id</code> (<code>aiogram.types.user_shared.UserShared</code> <code>amplib</code>), 233	V
<code>user_ids</code> (<code>aiogram.types.users_shared.UsersShared</code> <code>amplib</code>), 234	<code>value</code> (<code>aiogram.types.dice.Dice</code> <code>amplib</code>), 134
<code>user_is_bot</code> (<code>aiogram.types.keyboard_button_request_user.KeyboardButtonRequestUser</code> <code>amplib</code>), 157	<code>vcard</code> (<code>aiogram.methods.send_contact.SendContact</code> <code>amplib</code>), 396
<code>user_is_bot</code> (<code>aiogram.types.keyboard_button_request_user.KeyboardButtonRequestUser</code> <code>amplib</code>), 157	<code>vcard</code> (<code>aiogram.types.contact.Contact</code> <code>amplib</code>), 133
<code>user_is_premium</code> (<code>aiogram.types.keyboard_button_request_user.KeyboardButtonRequestUser</code> <code>amplib</code>), 157	<code>vcard</code> (<code>aiogram.types.keyboard_button_request_user.KeyboardButtonRequestUser</code> <code>amplib</code>), 263
<code>user_is_premium</code> (<code>aiogram.types.keyboard_button_request_users.KeyboardButtonRequestUsers</code> <code>amplib</code>), 157	<code>vcard</code> (<code>aiogram.types.keyboard_button_request_users.KeyboardButtonRequestUsers</code> <code>amplib</code>), 282
	<code>VENUE</code> (<code>aiogram.enums.content_type.ContentType</code> <code>amplib</code>), 497
	<code>VENUE</code> (<code>aiogram.enums.inline_query_result_type.InlineQueryResultType</code> <code>amplib</code>), 497
	<code>VENUE</code> (<code>aiogram.enums.inline_query_result_type.InlineQueryResultType</code> <code>amplib</code>), 137

venue (*aiogram.types.message.Message* *ampubym*), 167

Venue (клас в *aiogram.types.venue*), 234

via_bot (*aiogram.types.message.Message* *ampubym*), 166

via_chat_folder_invite_link (*aiogram.types.chat_member_updated.ChatMemberUpdated* *ampubym*), 111

via_join_request (*aiogram.types.chat_member_updated.ChatMemberUpdated* *ampubym*), 111

VIDEO (*aiogram.enums.content_type.ContentType* *ampubym*), 491

VIDEO (*aiogram.enums.inline_query_result_type.InlineQueryResultType* *ampubym*), 497

VIDEO (*aiogram.enums.input_media_type.InputMediaType* *ampubym*), 498

VIDEO (*aiogram.enums.sticker_format.StickerFormat* *ampubym*), 501

video (*aiogram.methods.send_video.SendVideo* *ampubym*), 420

video (*aiogram.types.external_reply_info.ExternalReplyInfo* *ampubym*), 136

video (*aiogram.types.message.Message* *ampubym*), 167

Video (клас в *aiogram.types.video*), 235

VIDEO_CHAT_ENDED (*aiogram.enums.content_type.ContentType* *ampubym*), 493

video_chat_ended (*aiogram.types.message.Message* *ampubym*), 169

VIDEO_CHAT_PARTICIPANTS_INVITED (*aiogram.enums.content_type.ContentType* *ampubym*), 493

video_chat_participants_invited (*aiogram.types.message.Message* *ampubym*), 169

VIDEO_CHAT_SCHEDULED (*aiogram.enums.content_type.ContentType* *ampubym*), 493

video_chat_scheduled (*aiogram.types.message.Message* *ampubym*), 169

VIDEO_CHAT_STARTED (*aiogram.enums.content_type.ContentType* *ampubym*), 493

video_chat_started (*aiogram.types.message.Message* *ampubym*), 169

video_duration (*aiogram.types.inline_query_result_video.InlineQueryResultVideo* *ampubym*), 279

video_file_id (*aiogram.types.inline_query_result_cached_voice.InlineQueryResultCachedVoice* *ampubym*), 260

video_height (*aiogram.types.inline_query_result_video.InlineQueryResultVideo* *ampubym*), 279

VIDEO_NOTE (*aiogram.enums.content_type.ContentType* *ampubym*), 491

video_note (*aiogram.methods.send_video_note.SendVideoNote* *ampubym*), 423

video_note (*aiogram.types.external_reply_info.ExternalReplyInfo* *ampubym*), 136

video_note (*aiogram.types.message.Message* *ampubym*), 167

video_url (*aiogram.types.inline_query_result_video.InlineQueryResultVideo* *ampubym*), 278

video_width (*aiogram.types.inline_query_result_video.InlineQueryResultVideo* *ampubym*), 279

VideoChatEnded (клас в *aiogram.types.video_chat_ended*), 236

VideoChatParticipantsInvited (клас в *aiogram.types.video_chat_participants_invited*), 236

VideoChatScheduled (клас в *aiogram.types.video_chat_scheduled*), 236

VideoChatStarted (клас в *aiogram.types.video_chat_started*), 237

VideoNote (клас в *aiogram.types.video_note*), 237

VIOLET (*aiogram.enums.topic_icon_color.TopicIconColor* *ampubym*), 502

VND (*aiogram.enums.currency.Currency* *ampubym*), 496

VOICE (*aiogram.enums.content_type.ContentType* *ampubym*), 491

VOICE (*aiogram.enums.inline_query_result_type.InlineQueryResultType* *ampubym*), 497

voice (*aiogram.methods.send_voice.SendVoice* *ampubym*), 426

voice (*aiogram.types.external_reply_info.ExternalReplyInfo* *ampubym*), 136

voice (*aiogram.types.message.Message* *ampubym*), 167

Voice (клас в *aiogram.types.voice*), 238

voice_duration (*aiogram.types.inline_query_result_voice.InlineQueryResultVoice* *ampubym*), 281

voice_file_id (*aiogram.types.inline_query_result_cached_voice.InlineQueryResultCachedVoice* *ampubym*), 262

voice_url (*aiogram.types.inline_query_result_voice.InlineQueryResultVoice* *ampubym*), 280

voter_chat (*aiogram.types.poll_answer.PollAnswer* *ampubym*), 222

voter_count (*aiogram.types.poll_option.PollOption* *ampubym*), 222

was_refunded (*aiogram.types.giveaway_winners.GiveawayWinners* *ampubym*), 143

watcher (*aiogram.types.proximity_alert_triggered.ProximityAlertTriggered* ampубym), 223
 WEB_APP (*aiogram.enums.menu_button_type.MenuButtonType* ampубym), 499
 web_app (*aiogram.types.inline_keyboard_button.InlineKeyboardButton* ampубym), 144
 web_app (*aiogram.types.inline_query_results_button.InlineQueryResultsButton* ampубym), 281
 web_app (*aiogram.types.keyboard_button.KeyboardButton* ampубym), 154
 web_app (*aiogram.types.menu_button.MenuButton* ampубym), 161
 web_app (*aiogram.types.menu_button_web_app.MenuButtonWebApp* ampубym), 162
 WEB_APP_DATA (*aiogram.enums.content_type.ContentType* ampубym), 493
 web_app_data (*aiogram.types.message.Message* ampубym), 170
 web_app_name (*aiogram.types.write_access_allowed.WriteAccessAllowed* ampубym), 239
 web_app_query_id (*aiogram.methods.answer_web_app_query.AnswerWebAppQuery* ampубym), 468
 WebAppChat (клас в *aiogram.utils.web_app*), 597
 WebAppData (клас в *aiogram.types.web_app_data*), 238
 WebAppInfo (клас в *aiogram.types.web_app_info*), 239
 WebAppInitData (клас в *aiogram.utils.web_app*), 595
 WebAppUser (клас в *aiogram.utils.web_app*), 596
 WebhookInfo (клас в *aiogram.types.webhook_info*), 312
 width (*aiogram.methods.send_animation.SendAnimation* ampубym), 389
 width (*aiogram.methods.send_video.SendVideo* ampубym), 420
 width (*aiogram.types.animation.Animation* ampубym), 19
 width (*aiogram.types.input_media_animation.InputMediaAnimation* ampубym), 148
 width (*aiogram.types.input_media_video.InputMediaVideo* ampубym), 152
 width (*aiogram.types.photo_size.PhotoSize* ampубym), 220
 width (*aiogram.types.sticker.Sticker* ampубym), 290
 width (*aiogram.types.video.Video* ampубym), 235
 winner_count (*aiogram.types.giveaway.Giveaway* ampубym), 141
 winner_count (*aiogram.types.giveaway_completed.GiveawayCompleted* ampубym), 142
 winner_count (*aiogram.types.giveaway_winners.GiveawayWinners* ampубym), 143
 winners (*aiogram.types.giveaway_winners.GiveawayWinners* ampубym), 143
 winners_selection_date (*aiogram.types.giveaway.Giveaway* ampубym), 141
 winners_selection_date (*aiogram.types.giveaway_winners.GiveawayWinners* ampубym), 143
 wrap_local_file (*aiogram.client.telegram.TelegramAPIServer* ampубym), 15
 WRITE_ACCESS_ALLOWED (*aiogram.enums.content_type.ContentType* ampубym), 492
 write_access_allowed (*aiogram.types.message.Message* ampубym), 168
 WriteAccessAllowed (клас в *aiogram.types.write_access_allowed*), 239
 x_shift (*aiogram.types.mask_position.MaskPosition* ampубym), 290
 y_shift (*aiogram.types.mask_position.MaskPosition* ampубym), 290
 year (*aiogram.types.birthdate.Birthdate* ampубym), 26
 YELLOW (*aiogram.enums.topic_icon_color.TopicIconColor* ampубym), 502
 YER (*aiogram.enums.currency.Currency* ampубym), 496
 ZAR (*aiogram.enums.currency.Currency* ampубym), 496