python-telegram-bot Documentation

Release 20.0a3

Leandro Toledo

Aug 27, 2022

REFERENCE

python-telegram-bot

We have made you a wrapper you can't refuse

We have a vibrant community of developers helping each other in our Telegram group. Join us!

Stay tuned for library updates and new releases on our Telegram Channel.

This library provides a pure Python, asynchronous interface for the Telegram Bot API. It's compatible with Python versions **3.7+**.

In addition to the pure API implementation, this library features a number of high-level classes to make the development of bots easy and straightforward. These classes are contained in the `telegram.ext` submodule.

A pure API implementation *without* `telegram.ext` is available as the standalone package `python-telegram-bot-raw`. See here for details.

CHAPTER

ONE

NOTE

Installing both `python-telegram-bot` and `python-telegram-bot-raw` in conjunction will result in undesired side-effects, so only install *one* of both.

CHAPTER

TWO

TELEGRAM API SUPPORT

All types and methods of the Telegram Bot API **6.2** are supported.

CHAPTER

THREE

INSTALLING

You can install or upgrade `python-telegram-bot` via

```
$ pip install python-telegram-bot --upgrade
```

To install a pre-release, use the `--pre` flag in addition.

You can also install `python-telegram-bot` from source, though this is usually not necessary.

```
$ git clone https://github.com/python-telegram-bot/python-telegram-bot
$ cd python-telegram-bot
$ python setup.py install
```

3.1 Dependencies & Their Versions

`python-telegram-bot` tries to use as few 3rd party dependencies as possible. However, for some features using a 3rd party library is more sane than implementing the functionality again. The dependencies are:

- httpx ~= 0.23.0 for `telegram.request.HTTPXRequest`, the default networking backend
- tornado~=6.2 for `telegram.ext.Updater.start_webhook`
- cachetools~=5.2.0 for `telegram.ext.CallbackDataCache`
- APScheduler~=3.9.1 for `telegram.ext.JobQueue`

`python-telegram-bot` is most useful when used along with additional libraries. To minimize dependency conflicts, we try to be liberal in terms of version requirements on the dependencies. On the other hand, we have to ensure stability of `python-telegram-bot`, which is why we do apply version bounds. If you encounter dependency conflicts due to these bounds, feel free to reach out.

3.2 Optional Dependencies

PTB can be installed with optional dependencies:

- `pip install python-telegram-bot[passport]` installs the cryptography>=3.0 library. Use this, if you want to use Telegram Passport related functionality.
- `pip install python-telegram-bot[socks]` installs `httpx[socks]`. Use this, if you want to work behind a Socks5 server.
- `pip install python-telegram-bot[rate-limiter]` installs `aiolimiter~=1.0.0`. Use this, if you want to use `telegram.ext.AIORateLimiter`.

CHAPTER

FOUR

QUICK START

Our Wiki contains an Introduction to the API explaining how the pure Bot API can be accessed via `python-telegram-bot`. Moreover, the Tutorial: Your first Bot gives an introduction on how chatbots can be easily programmed with the help of the `telegram.ext` module.

CHAPTER

FIVE

RESOURCES

- The package documentation is the technical reference for `python-telegram-bot`. It contains descriptions of all available classes, modules, methods and arguments as well as the changelog.
- The wiki is home to number of more elaborate introductions of the different features of `python-telegram-bot` and other useful resources that go beyond the technical documentation.
- Our examples section contains several examples that showcase the different features of both the Bot API and `python-telegram-bot`. Even if it is not your approach for learning, please take a look at `echobot.py`. It is the de facto base for most of the bots out there. The code for these examples is released to the public domain, so you can start by grabbing the code and building on top of it.
- The official Telegram Bot API documentation is of course always worth a read.

CHAPTER

SIX

GETTING HELP

If the resources mentioned above don't answer your questions or simply overwhelm you, there are several ways of getting help.

1. We have a vibrant community of developers helping each other in our Telegram group. Join us! Asking a question here is often the quickest way to get a pointer in the right direction.
2. Ask questions by opening a discussion.
3. You can even ask for help on Stack Overflow using the python-telegram-bot tag.

CHAPTER

SEVEN

CONCURRENCY

Since v20.0, `python-telegram-bot` is built on top of Pythons `asyncio` module. Because `asyncio` is in general single-threaded, `python-telegram-bot` does currently not aim to be thread-safe. Noteworthy parts of `python-telegram-bots` API that are likely to cause issues (e.g. race conditions) when used in a multi-threaded setting include:

- `telegram.ext.Application/Updater.update_queue`
- `telegram.ext.ConversationHandler.check/handle_update`
- `telegram.ext.CallbackDataCache`
- `telegram.ext.BasePersistence`
- all classes in the `telegram.ext.filters` module that allow to add/remove allowed users/chats at runtime

CHAPTER

EIGHT

CONTRIBUTING

Contributions of all sizes are welcome. Please review our contribution guidelines to get started. You can also help by reporting bugs or feature requests.

CHAPTER

NINE

DONATING

Occasionally we are asked if we accept donations to support the development. While we appreciate the thought, maintaining PTB is our hobby, and we have almost no running costs for it. We therefore have nothing set up to accept donations. If you still want to donate, we kindly ask you to donate to another open source project/initiative of your choice instead.

CHAPTER

TEN

LICENSE

You may copy, distribute and modify the software provided that modifications are described and licensed for free under LGPL-3. Derivatives works (including modifications or anything statically linked to the library) can only be redistributed under LGPL-3, but applications that use the library don't have to be.

10.1 telegram package

10.1.1 Version Constants

A library that provides a Python interface to the Telegram Bot API

telegram.**__bot_api_version__ = '6.2'**

Shortcut for *telegram.constants.BOT_API_VERSION*.

Changed in version 20.0: This constant was previously named `bot_api_version`.

Type
str

telegram.**__bot_api_version_info__ = BotAPIVersion(major=6, minor=2)**

Shortcut for *telegram.constants.BOT_API_VERSION_INFO*.

New in version 20.0.

Type
typing.NamedTuple

telegram.**__version__ = '20.0a3'**

The version of the *python-telegram-bot* library as string. To get detailed information about the version number, please use *__version_info__* instead.

Type
str

telegram.**__version_info__ = Version(major=20, minor=0, micro=0, releaselevel='alpha', serial=3)**

A tuple containing the five components of the version number: *major*, *minor*, *micro*, *releaselevel*, and *serial*. All values except *releaselevel* are integers. The release level is `'alpha'`, `'beta'`, `'candidate'`, or `'final'`. The components can also be accessed by name, so `__version_info__[0]` is equivalent to `__version_info__.major` and so on.

New in version 20.0.

Type
typing.NamedTuple

10.1.2 Available Types

telegram.Animation

class telegram.**Animation**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *file_unique_id* is equal.

Parameters

- ***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.
- ***thumb*** (*telegram.PhotoSize*, optional) – Animation thumbnail as defined by sender.
- ***file_name*** (str, optional) – Original animation filename as defined by sender.
- ***mime_type*** (str, optional) – MIME type of the file as defined by sender.
- ***file_size*** (int, optional) – File size in bytes.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

file_id

File identifier.

Type
str

file_unique_id

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

width

Video width as defined by sender.

Type
int

height

Video height as defined by sender.

Type
int

duration

Duration of the video in seconds as defined by sender.

Type
int

thumb

Optional. Animation thumbnail as defined by sender.

Type
telegram.PhotoSize

file_name

Optional. Original animation filename as defined by sender.

Type
str

mime_type

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

Type
str

file_size

Optional. File size in bytes.

Type
int

bot

Optional. The Bot to use for instance methods.

Type
telegram.Bot

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

async get_file(**, read_timeout=None, write_timeout=None, connect_timeout=None, pool_timeout=None, api_kwargs=None*)

Convenience wrapper over *telegram.Bot.get_file*

For the documentation of the arguments, please see *telegram.Bot.get_file()*.

Returns
telegram.File

Raises
telegram.error.TelegramError –

telegram.Audio

class telegram.**Audio**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *file_unique_id* is equal.

Parameters

- ***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, optional) – Performer of the audio as defined by sender or by audio tags.

- ***title*** (str, optional) – Title of the audio as defined by sender or by audio tags.
- ***file_name*** (str, optional) – Original filename as defined by sender.
- ***mime_type*** (str, optional) – MIME type of the file as defined by sender.
- ***file_size*** (int, optional) – File size in bytes.
- ***thumb*** (*telegram.PhotoSize*, optional) – Thumbnail of the album cover to which the music file belongs.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

file_id

Identifier for this file.

Type
str

file_unique_id

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

duration

Duration of the audio in seconds.

Type
int

performer

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

Type
str

title

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

Type
str

file_name

Optional. Original filename as defined by sender.

Type
str

mime_type

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

Type
str

file_size

Optional. File size in bytes.

Type
int

thumb

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

Type
telegram.PhotoSize

bot

Optional. The Bot to use for instance methods.

Type
telegram.Bot

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

async get_file(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Convenience wrapper over *telegram.Bot.get_file*

For the documentation of the arguments, please see *telegram.Bot.get_file()*.

Returns
telegram.File

Raises
telegram.error.TelegramError –

telegram.Bot

class telegram.**Bot**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*, AbstractAsyncContextManager

This object represents a Telegram Bot.

Instances of this class can be used as asyncio context managers, where

```
async with bot:
    # code
```

is roughly equivalent to

```
try:
    await bot.initialize()
    # code
finally:
    await request_object.shutdown()
```

Note:

- Most bot methods have the argument `api_kwargs` which allows passing arbitrary keywords to the Telegram API. This can be used to access new features of the API before they are incorporated into PTB. However, this is not guaranteed to work, i.e. it will fail for passing files.
- Bots should not be serialized since if you for e.g. change the bots token, then your serialized instance will not reflect that change. Trying to pickle a bot instance will raise `pickle.PicklingError`.

See also:

telegram.ext.Application.bot, *telegram.ext.CallbackContext.bot*, *telegram.ext.Updater.bot*

New in version 13.2: Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *bot* is equal.

Changed in version 20.0:

- Removed the deprecated methods `kick_chat_member`, `kickChatMember`, `get_chat_members_count` and `getChatMembersCount`.
- Removed the deprecated property `commands`.
- Removed the deprecated `defaults` parameter. If you want to use *telegram.ext.Defaults*, please use the subclass *telegram.ext.ExtBot* instead.
- Attempting to pickle a bot instance will now raise `pickle.PicklingError`.
- The following are now keyword-only arguments in Bot methods: `location`, `filename`, `venue`, `contact`, `{read, write, connect, pool}_timeout`, `api_kwargs`. Use a named argument for those, and notice that some positional arguments changed position as a result.

Parameters

- ***token*** (`str`) – Bot's unique authentication token.
- ***base_url*** (`str`, optional) – Telegram Bot API service URL.
- ***base_file_url*** (`str`, optional) – Telegram Bot API file URL.
- ***request*** (*telegram.request.BaseRequest*, optional) – Pre initialized *telegram.request.BaseRequest* instances. Will be used for all bot methods *except* for *get_updates()*. If not passed, an instance of *telegram.request.HTTPXRequest* will be used.
- ***get_updates_request*** (*telegram.request.BaseRequest*, optional) – Pre initialized *telegram.request.BaseRequest* instances. Will be used exclusively for *get_updates()*. If not passed, an instance of *telegram.request.HTTPXRequest* will be used.
- ***private_key*** (`bytes`, optional) – Private key for decryption of telegram passport data.
- ***private_key_password*** (`bytes`, optional) – Password for above private key.

send_animation()	Used for sending animations
send_audio()	Used for sending audio files
send_chat_action()	Used for sending chat actions
send_contact()	Used for sending contacts
send_dice()	Used for sending dice messages
send_document()	Used for sending documents
send_game()	Used for sending a game
send_invoice()	Used for sending an invoice
send_location()	Used for sending location
send_media_group()	Used for sending media grouped together
send_message()	Used for sending text messages
send_photo()	Used for sending photos
send_poll()	Used for sending polls
send_sticker()	Used for sending stickers
send_venue()	Used for sending venue locations.
send_video()	Used for sending videos
send_video_note()	Used for sending video notes
send_voice()	Used for sending voice messages
copy_message()	Used for copying the contents of an arbitrary message
forward_message()	Used for forwarding messages

answer_callback_query()	Used for answering the callback query
answer_inline_query()	Used for answering the inline query
answer_pre_checkout_query()	Used for answering a pre checkout query
answer_shipping_query()	Used for answering a shipping query
answer_web_app_query()	Used for answering a web app query
edit_message_caption()	Used for editing captions
edit_message_media()	Used for editing the media on messages
edit_message_live_location()	Used for editing the location in live location messages
edit_message_reply_markup()	Used for editing the reply markup on messages
edit_message_text()	Used for editing text messages
stop_poll()	Used for stopping the running poll
delete_message()	Used for deleting messages.

ban_chat_member()	Used for banning a member from the chat
unban_chat_member()	Used for unbanning a member from the chat
ban_chat_sender_chat()	Used for banning a channel in a channel or supergroup
unban_chat_sender_chat()	Used for unbanning a channel in a channel or supergroup
restrict_chat_member()	Used for restricting a chat member
promote_chat_member()	Used for promoting a chat member
set_chat_administrator_custom_title()	Used for assigning a custom admin title to an admin
set_chat_permissions()	Used for setting the permissions of a chat
export_chat_invite_link()	Used for creating a new primary invite link for a chat
create_chat_invite_link()	Used for creating an additional invite link for a chat
edit_chat_invite_link()	Used for editing a non-primary invite link
revoke_chat_invite_link()	Used for revoking an invite link created by the bot
approve_chat_join_request()	Used for approving a chat join request
decline_chat_join_request()	Used for declining a chat join request
set_chat_photo()	Used for setting a photo to a chat
delete_chat_photo()	Used for deleting a chat photo
set_chat_title()	Used for setting a chat title
set_chat_description()	Used for setting the description of a chat
pin_chat_message()	Used for pinning a message
unpin_chat_message()	Used for unpinning a message
unpin_all_chat_messages()	Used for unpinning all pinned chat messages
get_user_profile_photos()	Used for obtaining user's profile pictures
get_chat()	Used for getting information about a chat
get_chat_administrators()	Used for getting the list of admins in a chat
get_chat_member_count()	Used for getting the number of members in a chat
get_chat_member()	Used for getting a member of a chat
set_my_commands()	Used for setting the list of commands
delete_my_commands()	Used for deleting the list of commands
get_my_commands()	Used for obtaining the list of commands
get_my_default_administrator_rights()	Used for obtaining the default administrator rights for the bot
set_my_default_administrator_rights()	Used for setting the default administrator rights for the bot
get_chat_menu_button()	Used for obtaining the menu button of a private chat or the default menu button
set_chat_menu_button()	Used for setting the menu button of a private chat or the default menu button
leave_chat()	Used for leaving a chat

add_sticker_to_set()	Used for adding a sticker to a set
delete_sticker_from_set()	Used for deleting a sticker from a set
create_new_sticker_set()	Used for creating a new sticker set
set_chat_sticker_set()	Used for setting a sticker set
delete_chat_sticker_set()	Used for deleting the set sticker set
set_sticker_position_in_set()	Used for moving a sticker's position in the set
set_sticker_set_thumb()	Used for setting the thumbnail of a sticker set
get_sticker_set()	Used for getting a sticker set
upload_sticker_file()	Used for uploading a sticker file
get_custom_emoji_stickers()	Used for getting custom emoji files based on their IDs

get_game_high_scores()	Used for getting the game high scores
set_game_score()	Used for setting the game score

get_updates()	Used for getting updates using long polling
get_webhook_info()	Used for getting current webhook status
set_webhook()	Used for setting a webhook to receive updates
delete_webhook()	Used for removing webhook integration

create_invoice_link()	Used to generate an HTTP link for an invoice
close()	Used for closing server instance when switching to another local server
log_out()	Used for logging out from cloud Bot API server
get_file()	Used for getting basic info about a file
get_me()	Used for getting basic information about the bot

bot	The user instance of the bot as returned by get_me()
can_join_groups	Whether the bot can join groups
can_read_all_group_messages	Whether the bot can read all incoming group messages
id	The user id of the bot
name	The username of the bot, with leading @
first_name	The first name of the bot
last_name	The last name of the bot
username	The username of the bot, without leading @
link	The t.me link of the bot
supports_inline_queries	Whether the bot supports inline queries

async addStickerToSet(*user_id*, *name*, *emojis*, *png_sticker=None*, *mask_position=None*, *tgs_sticker=None*, *webm_sticker=None*, ***, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for add_sticker_to_set()

async add_sticker_to_set(*user_id*, *name*, *emojis*, *png_sticker=None*, *mask_position=None*, *tgs_sticker=None*, *webm_sticker=None*, ***, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to add a new sticker to a set created by the bot. You **must** use exactly one of the fields png_sticker, tgs_sticker or webm_sticker. Animated stickers can be added to animated sticker sets and only to them. Animated sticker sets can have up to 50 stickers. Static sticker sets can have up to 120 stickers.

Warning: As of API 4.7 *png_sticker* is an optional argument and therefore the order of the arguments had to be changed. Use keyword arguments to make sure that the arguments are passed correctly.

Note: The *png_sticker* and *tgs_sticker* argument can be either a file_id, an URL or a file from disk `open(filename, 'rb')`

Parameters

- ***user_id*** (int) – User identifier of created sticker set owner.
- ***name*** (str) – Sticker set name.
- ***png_sticker*** (str | file object | bytes | pathlib.Path, optional) – **PNG** image with the sticker, must be up to 512 kilobytes in size, dimensions must not exceed 512px, and either width or height must be exactly 512px. Pass a file_id as a String to send a file that already exists on the Telegram servers, pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data.

  Changed in version 13.2: Accept bytes as input.
- ***tgs_sticker*** (str | file object | bytes | pathlib.Path, optional) – **TGS** animation with the sticker, uploaded using multipart/form-data. See https://core.telegram.org/stickers#animated-sticker-requirements for technical requirements.

  Changed in version 13.2: Accept bytes as input.
- ***webm_sticker*** (str | file object | bytes | pathlib.Path, optional) – **WEBM** video with the sticker, uploaded using multipart/form-data. See https://core.telegram.org/stickers#video-sticker-requirements for technical requirements.

  New in version 13.11.
- ***emojis*** (str) – One or more emoji corresponding to the sticker.
- ***mask_position*** (*telegram.MaskPosition*, optional) – Position where the mask should be placed on faces.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

async answerCallbackQuery(*callback_query_id*, *text=None*, *show_alert=None*, *url=None*, *cache_time=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *answer_callback_query()*

async answerInlineQuery(*inline_query_id*, *results*, *cache_time=None*, *is_personal=None*, *next_offset=None*, *switch_pm_text=None*, *switch_pm_parameter=None*, ***, *current_offset=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *answer_inline_query()*

async answerPreCheckoutQuery(*pre_checkout_query_id*, *ok*, *error_message=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *answer_pre_checkout_query()*

async answerShippingQuery(*shipping_query_id*, *ok*, *shipping_options=None*, *error_message=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *answer_shipping_query()*

async answerWebAppQuery(*web_app_query_id*, *result*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *answer_web_app_query()*

async answer_callback_query(*callback_query_id*, *text=None*, *show_alert=None*, *url=None*, *cache_time=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send answers to callback queries sent from inline keyboards. The answer will be displayed to the user as a notification at the top of the chat screen or as an alert. Alternatively, the user can be redirected to the specified Game URL. For this option to work, you must first create a game for your bot via @BotFather and accept the terms. Otherwise, you may use links like t.me/your_bot?start=XXXX that open your bot with a parameter.

See also:

telegram.CallbackQuery.answer

Parameters

- ***callback_query_id*** (`str`) – Unique identifier for the query to be answered.
- ***text*** (`str`, optional) – Text of the notification. If not specified, nothing will be shown to the user, 0-*200* characters.
- ***show_alert*** (`bool`, optional) – If `True`, an alert will be shown by the client instead of a notification at the top of the chat screen. Defaults to `False`.
- ***url*** (`str`, optional) – URL that will be opened by the user's client. If you have created a 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 callback game button. Otherwise, you may use links like t.me/your_bot?start=XXXX that open your bot with a parameter.
- ***cache_time*** (`int`, optional) – The maximum amount of time in seconds that the result of the callback query may be cached client-side. Defaults to 0.

Keyword Arguments

- ***read_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.

- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
bool On success, True is returned.

Raises
telegram.error.TelegramError –

async answer_inline_query(*inline_query_id*, *results*, *cache_time=None*, *is_personal=None*, *next_offset=None*, *switch_pm_text=None*, *switch_pm_parameter=None*, ***, *current_offset=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

> **Warning:** In most use cases *current_offset* should not be passed manually. Instead of calling this method directly, use the shortcut *telegram.InlineQuery.answer()* with *telegram.InlineQuery.answer.auto_pagination* set to True, which will take care of passing the correct value.

See also:

telegram.InlineQuery.answer

Parameters

- ***inline_query_id*** (str) – Unique identifier for the answered query.
- ***results*** (List[*telegram.InlineQueryResult*] | Callable) – A list of results for the inline query. In case *current_offset* is passed, *results* may also be a callable that accepts the current page index starting from 0. It must return either a list of *telegram.InlineQueryResult* instances or None if there are no more results.
- ***cache_time*** (int, optional) – 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, optional) – Pass True, if results may be cached on the server side only for the user that sent the query. By default, results may be returned to any user who sends the same query.
- ***next_offset*** (str, optional) – Pass the offset that a client should send in the next query with the same text to receive more results. Pass an empty string if there are no more results or if you don't support pagination. Offset length can't exceed 64 bytes.
- ***switch_pm_text*** (str, optional) – If passed, clients will display a button with specified text that switches the user to a private chat with the bot and sends the bot a start message with the parameter *switch_pm_parameter*.
- ***switch_pm_parameter*** (str, optional) – 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.

Keyword Arguments

- ***current_offset*** (str, optional) – The *telegram.InlineQuery.offset* of the inline query to answer. If passed, PTB will automatically take care of the pagination for you, i.e. pass the correct *next_offset* and truncate the results list/get the results from the callable you passed.

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Example

An inline bot that sends YouTube videos can ask the user to connect the bot to their YouTube account to adapt search results accordingly. To do this, it displays a 'Connect your YouTube account' button above the results, or even before showing any. The user presses the button, switches to a private chat with the bot and, in doing so, passes a start parameter that instructs the bot to return an oauth link. Once done, the bot can offer a switch_inline button so that the user can easily return to the chat where they wanted to use the bot's inline capabilities.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async **answer_pre_checkout_query**(*pre_checkout_query_id*, *ok*, *error_message=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

Note: The Bot API must receive an answer within 10 seconds after the pre-checkout query was sent.

See also:

telegram.PreCheckoutQuery.answer

Parameters

- ***pre_checkout_query_id*** (str) – Unique identifier for the query to be answered.
- ***ok*** (bool) – Specify True if everything is alright (goods are available, etc.) and the bot is ready to proceed with the order. Use False if there are any problems.
- ***error_message*** (str, optional) – 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.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async answer_shipping_query(*shipping_query_id*, *ok*, *shipping_options=None*, *error_message=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

If you sent an invoice requesting a shipping address and the parameter *send_invoice.is_flexible* was specified, the Bot API will send an *telegram.Update* with a *telegram.Update.shipping_query* field to the bot. Use this method to reply to shipping queries.

See also:

telegram.ShippingQuery.answer

Parameters

- ***shipping_query_id*** (str) – Unique identifier for the query to be answered.
- ***ok*** (bool) – Specify True if delivery to the specified address is possible and False if there are any problems (for example, if delivery to the specified address is not possible).
- ***shipping_options*** (List[*telegram.ShippingOption*]) – Required if ok is True. An array of available shipping options.
- ***error_message*** (str, optional) – 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.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, `True` is returned.

Return type
`bool`

Raises
telegram.error.TelegramError –

async **answer_web_app_query**(*web_app_query_id*, *result*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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.

New in version 20.0.

Parameters

- ***web_app_query_id*** (`str`) – Unique identifier for the query to be answered.
- ***result*** (*telegram.InlineQueryResult*) – An object describing the message to be sent.

Keyword Arguments

- ***read_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (`dict`, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, a sent *telegram.SentWebAppMessage* is returned.

Return type
telegram.SentWebAppMessage

Raises
telegram.error.TelegramError –

async **approveChatJoinRequest**(*chat_id*, *user_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *approve_chat_join_request()*

async **approve_chat_join_request**(*chat_id*, *user_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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 *telegram.ChatPermissions.can_invite_users* administrator right.

See also:

telegram.Chat.approve_join_request, *telegram.ChatJoinRequest.approve*, *telegram.User.approve_join_request*

New in version 13.8.

Parameters

- ***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.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async banChatMember(*chat_id*, *user_id*, *until_date=None*, *revoke_messages=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *ban_chat_member()*

async banChatSenderChat(*chat_id*, *sender_chat_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *ban_chat_sender_chat()*

async ban_chat_member(*chat_id*, *user_id*, *until_date=None*, *revoke_messages=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also:

telegram.Chat.ban_member

New in version 13.7.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target group or username of the target supergroup or channel (in the format @channelusername).
- ***user_id*** (int) – Unique identifier of the target user.
- ***until_date*** (int | datetime.datetime, optional) – 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. For timezone naive datetime.datetime objects,

the default timezone of the bot will be used, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

- ***revoke_messages*** (bool, optional) – 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.

  New in version 13.4.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async ban_chat_sender_chat(*chat_id*, *sender_chat_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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.

See also:

telegram.Chat.ban_chat, *telegram.Chat.ban_sender_chat*

New in version 13.9.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target group or username of the target supergroup or channel (in the format @channelusername).
- ***sender_chat_id*** (int) – Unique identifier of the target sender chat.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.

- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

property bot

User instance for the bot as returned by *get_me()*.

> **Warning:** This value is the cached return value of *get_me()*. If the bots profile is changed during runtime, this value won't reflect the changes until *get_me()* is called again.

See also:

initialize()

Type
telegram.User

property can_join_groups

Bot's *telegram.User.can_join_groups* attribute. Shortcut for the corresponding attribute of *bot*.

Type
bool

property can_read_all_group_messages

Bot's *telegram.User.can_read_all_group_messages* attribute. Shortcut for the corresponding attribute of *bot*.

Type
bool

async close(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success

Return type
True

Raises
telegram.error.TelegramError –

async **copyMessage**(*chat_id*, *from_chat_id*, *message_id*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *disable_notification=None*, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *reply_markup=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *copy_message()*

async **copy_message**(*chat_id*, *from_chat_id*, *message_id*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *disable_notification=None*, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *reply_markup=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to copy messages of any kind. Service messages and invoice messages can't be copied. The method is analogous to the method *forward_message()*, but the copied message doesn't have a link to the original message.

See also:

telegram.Message.copy, *telegram.Chat.send_copy*, *telegram.Chat.copy_message*, *telegram.User.send_copy*, *telegram.User.copy_message*

Parameters

- ***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.
- ***caption*** (str, optional) – New caption for media, 0-*1024* characters after entities parsing. If not specified, the original caption is kept.
- ***parse_mode*** (str, optional) – Mode for parsing entities in the new caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the new caption, which can be specified instead of parse_mode.
- ***disable_notification*** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.
- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message.
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- ***reply_markup*** (*InlineKeyboardMarkup* | *ReplyKeyboardMarkup* | *ReplyKeyboardRemove* | *ForceReply*, optional) – Additional interface options. An object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

Keyword Arguments

- **read_timeout** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- **write_timeout** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- **connect_timeout** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- **pool_timeout** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success

Return type
telegram.MessageId

Raises
telegram.error.TelegramError –

async createChatInviteLink(*chat_id*, *expire_date=None*, *member_limit=None*, *name=None*, *creates_join_request=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *create_chat_invite_link()*

async createInvoiceLink(*title*, *description*, *payload*, *provider_token*, *currency*, *prices*, *max_tip_amount=None*, *suggested_tip_amounts=None*, *provider_data=None*, *photo_url=None*, *photo_size=None*, *photo_width=None*, *photo_height=None*, *need_name=None*, *need_phone_number=None*, *need_email=None*, *need_shipping_address=None*, *send_phone_number_to_provider=None*, *send_email_to_provider=None*, *is_flexible=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *create_invoice_link()*

async createNewStickerSet(*user_id*, *name*, *title*, *emojis*, *png_sticker=None*, *mask_position=None*, *tgs_sticker=None*, *webm_sticker=None*, *sticker_type=None*, ***, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *create_new_sticker_set()*

async create_chat_invite_link(*chat_id*, *expire_date=None*, *member_limit=None*, *name=None*, *creates_join_request=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also:

telegram.Chat.create_invite_link

New in version 13.4.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- ***expire_date*** (int | datetime.datetime, optional) – Date when the link will expire. Integer input will be interpreted as Unix timestamp. For timezone naive datetime.datetime objects, the default timezone of the bot will be used, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***member_limit*** (int, optional) – Maximum number of users that can be members of the chat simultaneously after joining the chat via this invite link; 1-*99999*.
- ***name*** (str, optional) – Invite link name; 0-*32* characters.

  New in version 13.8.
- ***creates_join_request*** (bool, optional) – True, if users joining the chat via the link need to be approved by chat administrators. If True, *member_limit* can't be specified.

  New in version 13.8.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

telegram.ChatInviteLink

Raises

telegram.error.TelegramError –

async create_invoice_link(*title*, *description*, *payload*, *provider_token*, *currency*, *prices*, *max_tip_amount=None*, *suggested_tip_amounts=None*, *provider_data=None*, *photo_url=None*, *photo_size=None*, *photo_width=None*, *photo_height=None*, *need_name=None*, *need_phone_number=None*, *need_email=None*, *need_shipping_address=None*, *send_phone_number_to_provider=None*, *send_email_to_provider=None*, *is_flexible=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to create a link for an invoice.

New in version 20.0.

Parameters

- ***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) – Payments provider token, obtained via @BotFather.

- ***currency*** (str) – Three-letter ISO 4217 currency code, see more on currencies.
- ***prices*** (List[*telegram.LabeledPrice*) – Price breakdown, a list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.).
- ***max_tip_amount*** (int, 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], optional) – An 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 | object, optional) – 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. When an object is passed, it will be encoded as JSON.
- ***photo_url*** (str, optional) – URL of the product photo for the invoice. Can be a photo of the goods or a marketing image for a service.
- ***photo_size*** (int, optional) – Photo size in bytes.
- ***photo_width*** (int, optional) – Photo width.
- ***photo_height*** (int, optional) – Photo height.
- ***need_name*** (bool, optional) – Pass True, if you require the user's full name to complete the order.
- ***need_phone_number*** (bool, optional) – Pass True, if you require the user's phone number to complete the order.
- ***need_email*** (bool, optional) – Pass True, if you require the user's email address to complete the order.
- ***need_shipping_address*** (bool, optional) – Pass True, if you require the user's shipping address to complete the order.
- ***send_phone_number_to_provider*** (bool, optional) – Pass True, if user's phone number should be sent to provider.
- ***send_email_to_provider*** (bool, optional) – Pass True, if user's email address should be sent to provider.
- ***is_flexible*** (bool, optional) – Pass True, if the final price depends on the shipping method.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, the created invoice link is returned.

Return type
str

async **create_new_sticker_set**(*user_id*, *name*, *title*, *emojis*, *png_sticker=None*, *mask_position=None*, *tgs_sticker=None*, *webm_sticker=None*, *sticker_type=None*, ***, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

Warning: As of API 4.7 *png_sticker* is an optional argument and therefore the order of the arguments had to be changed. Use keyword arguments to make sure that the arguments are passed correctly.

Note: The *png_sticker* and *tgs_sticker* argument can be either a file_id, an URL or a file from disk `open(filename, 'rb')`

Changed in version 20.0: The parameter `contains_masks` has been removed. Use *sticker_type* instead.

Parameters

- ***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.
- ***png_sticker*** (str | file object | bytes | pathlib.Path, optional) – **PNG** image with the sticker, must be up to 512 kilobytes in size, dimensions must not exceed 512px, and either width or height must be exactly 512px. Pass a file_id as a String to send a file that already exists on the Telegram servers, pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data.

  Changed in version 13.2: Accept bytes as input.
- ***tgs_sticker*** (str | file object | bytes | pathlib.Path, optional) – **TGS** animation with the sticker, uploaded using multipart/form-data. See https://core.telegram.org/stickers#animated-sticker-requirements for technical requirements.

  Changed in version 13.2: Accept bytes as input.
- ***webm_sticker*** (str | file object | bytes | pathlib.Path, optional) – **WEBM** video with the sticker, uploaded using multipart/form-data. See https://core.telegram.org/stickers#video-sticker-requirements for technical requirements.

  New in version 13.11.
- ***emojis*** (str) – One or more emoji corresponding to the sticker.
- ***mask_position*** (*telegram.MaskPosition*, optional) – Position where the mask should be placed on faces.
- ***sticker_type*** (str, optional) – Type of stickers in the set, pass *telegram.Sticker.REGULAR* or *telegram.Sticker.MASK*. Custom emoji sticker sets can't be created via the Bot API at the moment. By default, a regular sticker set is created.

New in version 20.0.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

async declineChatJoinRequest(*chat_id*, *user_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *decline_chat_join_request()*

async decline_chat_join_request(*chat_id*, *user_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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 *telegram.ChatPermissions.can_invite_users* administrator right.

See also:

telegram.Chat.decline_join_request, *telegram.ChatJoinRequest.decline*, *telegram.User.decline_join_request*

New in version 13.8.

Parameters

- ***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.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.

- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async deleteChatPhoto(*chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *delete_chat_photo()*

async deleteChatStickerSet(*chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *delete_chat_sticker_set()*

async deleteMessage(*chat_id*, *message_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *delete_message()*

async deleteMyCommands(*scope=None*, *language_code=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *delete_my_commands()*

async deleteStickerFromSet(*sticker*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *delete_sticker_from_set()*

async deleteWebhook(*drop_pending_updates=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *delete_webhook()*

async delete_chat_photo(*chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also:

telegram.Chat.delete_photo

Parameters
chat_id (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.

- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async delete_chat_sticker_set(*chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to delete a group sticker set from a supergroup. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights. Use the field *telegram.Chat.can_set_sticker_set* optionally returned in *get_chat()* requests to check if the bot can use this method.

Parameters
chat_id (int | str) – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername).

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

async delete_message(*chat_id*, *message_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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.
- 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.

See also:

telegram.Message.delete(), *telegram.CallbackQuery.delete_message()*

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- ***message_id*** (int) – Identifier of the message to delete.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async delete_my_commands(*scope=None*, *language_code=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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.

New in version 13.7.

Parameters

- ***scope*** (*telegram.BotCommandScope*, optional) – An object, describing scope of users for which the commands are relevant. Defaults to *telegram.BotCommandScopeDefault*.
- ***language_code*** (str, optional) – 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.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, `True` is returned.

Return type
`bool`

Raises
telegram.error.TelegramError –

async delete_sticker_from_set(*sticker*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

Parameters
sticker (`str`) – File identifier of the sticker.

Keyword Arguments

- ***read_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (`dict`, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, `True` is returned.

Return type
`bool`

Raises
telegram.error.TelegramError –

async delete_webhook(*drop_pending_updates=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to remove webhook integration if you decide to switch back to *get_updates()*.

Parameters
drop_pending_updates (`bool`, optional) – Pass `True` to drop all pending updates.

Keyword Arguments

- ***read_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (`dict`, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, `True` is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async **editChatInviteLink**(*chat_id*, *invite_link*, *expire_date=None*, *member_limit=None*, *name=None*, *creates_join_request=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *edit_chat_invite_link()*

async **editMessageCaption**(*chat_id=None*, *message_id=None*, *inline_message_id=None*, *caption=None*, *reply_markup=None*, *parse_mode=None*, *caption_entities=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *edit_message_caption()*

async **editMessageLiveLocation**(*chat_id=None*, *message_id=None*, *inline_message_id=None*, *latitude=None*, *longitude=None*, *reply_markup=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, ***, *location=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *edit_message_live_location()*

async **editMessageMedia**(*media*, *chat_id=None*, *message_id=None*, *inline_message_id=None*, *reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *edit_message_media()*

async **editMessageReplyMarkup**(*chat_id=None*, *message_id=None*, *inline_message_id=None*, *reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *edit_message_reply_markup()*

async **editMessageText**(*text*, *chat_id=None*, *message_id=None*, *inline_message_id=None*, *parse_mode=None*, *disable_web_page_preview=None*, *reply_markup=None*, *entities=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *edit_message_text()*

async **edit_chat_invite_link**(*chat_id*, *invite_link*, *expire_date=None*, *member_limit=None*, *name=None*, *creates_join_request=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

Note: Though not stated explicitly in the official docs, Telegram changes not only the optional parameters that are explicitly passed, but also replaces all other optional parameters to the default values. However, since not documented, this behaviour may change unbeknown to PTB.

See also:

telegram.Chat.edit_invite_link

New in version 13.4.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- ***invite_link*** (str | *telegram.ChatInviteLink*) – The invite link to edit.

  Changed in version 20.0: Now also accepts *telegram.ChatInviteLink* instances.
- ***expire_date*** (int | datetime.datetime, optional) – Date when the link will expire. For timezone naive datetime.datetime objects, the default timezone of the bot will be used, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***member_limit*** (int, optional) – Maximum number of users that can be members of the chat simultaneously after joining the chat via this invite link; 1-*99999*.
- ***name*** (str, optional) – Invite link name; 0-*32* characters.

  New in version 13.8.
- ***creates_join_request*** (bool, optional) – True, if users joining the chat via the link need to be approved by chat administrators. If True, *member_limit* can't be specified.

  New in version 13.8.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

telegram.ChatInviteLink

Raises

telegram.error.TelegramError –

async edit_message_caption(*chat_id=None*, *message_id=None*, *inline_message_id=None*, *caption=None*, *reply_markup=None*, *parse_mode=None*, *caption_entities=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to edit captions of messages.

Note: It is currently only possible to edit messages without *telegram.Message.reply_markup* or with inline keyboards

---

See also:

telegram.Message.edit_caption, *telegram.CallbackQuery.edit_message_caption*

Parameters

- ***chat_id*** (int | str, optional) – 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, optional) – Required if inline_message_id is not specified. Identifier of the message to edit.
- ***inline_message_id*** (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.
- ***caption*** (str, optional) – New caption of the message, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in message text, which can be specified instead of *parse_mode*.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – An object for an inline keyboard.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, if edited message is not an inline message, the edited message is returned, otherwise True is returned.

Return type

telegram.Message

Raises

telegram.error.TelegramError –

async edit_message_live_location(*chat_id=None*, *message_id=None*, *inline_message_id=None*, *latitude=None*, *longitude=None*, *reply_markup=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, ***, *location=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

Note: You can either supply a *latitude* and *longitude* or a *location*.

See also:

telegram.Message.edit_live_location, *telegram.CallbackQuery.edit_message_live_location*

Parameters

- ***chat_id*** (int | str, optional) – 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, optional) – Required if inline_message_id is not specified. Identifier of the message to edit.
- ***inline_message_id*** (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.
- ***latitude*** (float, optional) – Latitude of location.
- ***longitude*** (float, optional) – Longitude of location.
- ***horizontal_accuracy*** (float, optional) – The radius of uncertainty for the location, measured in meters; 0-*1500*.
- ***heading*** (int, optional) – Direction in which the user is moving, in degrees. Must be between 1 and *360* if specified.
- ***proximity_alert_radius*** (int, optional) – Maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and *360* if specified.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – An object for a new inline keyboard.

Keyword Arguments

- ***location*** (*telegram.Location*, optional) – The location to send.
- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, if edited message is not an inline message, the edited message is returned, otherwise True is returned.

Return type

telegram.Message

async edit_message_media(*media*, *chat_id=None*, *message_id=None*, *inline_message_id=None*, *reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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.

Note: It is currently only possible to edit messages without *telegram.Message.reply_markup* or with inline keyboards

---

See also:

telegram.Message.edit_media, *telegram.CallbackQuery.edit_message_media*

Parameters

- ***media*** (*telegram.InputMedia*) – An object for a new media content of the message.
- ***chat_id*** (int | str, optional) – 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, optional) – Required if inline_message_id is not specified. Identifier of the message to edit.
- ***inline_message_id*** (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – An object for an inline keyboard.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, if edited message is not an inline message, the edited Message is returned, otherwise True is returned.

Return type

telegram.Message

Raises

telegram.error.TelegramError –

async edit_message_reply_markup(*chat_id=None*, *message_id=None*, *inline_message_id=None*, *reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

Note: It is currently only possible to edit messages without *telegram.Message.reply_markup* or with inline keyboards

---

See also:

telegram.Message.edit_reply_markup, *telegram.CallbackQuery.edit_message_reply_markup*

Parameters

- ***chat_id*** (int | str, optional) – 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, optional) – Required if inline_message_id is not specified. Identifier of the message to edit.
- ***inline_message_id*** (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – An object for an inline keyboard.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, if edited message is not an inline message, the edited message is returned, otherwise True is returned.

Return type

telegram.Message

Raises

telegram.error.TelegramError –

async edit_message_text(*text*, *chat_id=None*, *message_id=None*, *inline_message_id=None*, *parse_mode=None*, *disable_web_page_preview=None*, *reply_markup=None*, *entities=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to edit text and game messages.

Note: It is currently only possible to edit messages without *telegram.Message.reply_markup* or with inline keyboards.

See also:

telegram.Message.edit_text, *telegram.CallbackQuery.edit_message_text*

Parameters

- ***chat_id*** (int | str, optional) – 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, optional) – Required if inline_message_id is not specified. Identifier of the message to edit.
- ***inline_message_id*** (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.

- ***text*** (str) – New text of the message, 1-*4096* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in message text, which can be specified instead of *parse_mode*.
- ***disable_web_page_preview*** (bool, optional) – Disables link previews for links in this message.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – An object for an inline keyboard.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, if edited message is not an inline message, the edited message is returned, otherwise True is returned.

Return type

telegram.Message

Raises

telegram.error.TelegramError –

async exportChatInviteLink(*chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *export_chat_invite_link()*

async export_chat_invite_link(*chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also:

telegram.Chat.export_invite_link

Parameters

chat_id (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.

- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

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 *export_chat_invite_link()* or by calling the *get_chat()* method. If your bot needs to generate a new primary invite link replacing its previous one, use *export_chat_invite_link* again.

Returns
New invite link on success.

Return type
str

Raises
telegram.error.TelegramError –

property first_name

Bot's first name. Shortcut for the corresponding attribute of *bot*.

Type
str

async forwardMessage(*chat_id*, *from_chat_id*, *message_id*, *disable_notification=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *forward_message()*

async forward_message(*chat_id*, *from_chat_id*, *message_id*, *disable_notification=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to forward messages of any kind. Service messages can't be forwarded.

Note: Since the release of Bot API 5.5 it can be impossible to forward messages from some chats. Use the attributes *telegram.Message.has_protected_content* and *telegram.Chat.has_protected_content* to check this.

As a workaround, it is still possible to use *copy_message()*. However, this behaviour is undocumented and might be changed by Telegram.

See also:

telegram.Message.forward, *telegram.Chat.forward_to*, *telegram.Chat.forward_from*

Parameters

- ***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.

- ***disable_notification*** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, the sent Message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async getChat(*chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_chat()*

async getChatAdministrators(*chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_chat_administrators()*

async getChatMember(*chat_id*, *user_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_chat_member()*

async getChatMemberCount(*chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_chat_member_count()*

async getChatMenuButton(*chat_id=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_chat_menu_button()*

async getCustomEmojiStickers(*custom_emoji_ids*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_custom_emoji_stickers()*

async getFile(*file_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_file()*

async getGameHighScores(*user_id*, *chat_id=None*, *message_id=None*, *inline_message_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_game_high_scores()*

async getMe(***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_me()*

async getMyCommands(*scope=None*, *language_code=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_my_commands()*

async getMyDefaultAdministratorRights(*for_channels=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_my_default_administrator_rights()*

async getStickerSet(*name*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_sticker_set()*

async getUpdates(*offset=None*, *limit=None*, *timeout=None*, *allowed_updates=None*, ***, *read_timeout=2*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_updates()*

async getUserProfilePhotos(*user_id*, *offset=None*, *limit=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_user_profile_photos()*

async getWebhookInfo(***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_webhook_info()*

async get_chat(*chat_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to get up to date information about the chat (current name of the user for one-on-one conversations, current username of a user, group or channel, etc.).

Parameters

chat_id (int | str) – Unique identifier for the target chat or username of the target supergroup or channel (in the format `@channelusername`).

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
telegram.Chat

Raises
telegram.error.TelegramError –

async get_chat_administrators(*chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also:

telegram.Chat.get_administrators

Parameters
chat_id (int | str) – Unique identifier for the target chat or username of the target supergroup or channel (in the format @channelusername).

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

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

Return type
List[*telegram.ChatMember*]

Raises
telegram.error.TelegramError –

async get_chat_member(*chat_id*, *user_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to get information about a member of a chat.

See also:

telegram.Chat.get_member

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target supergroup or channel (in the format @channelusername).
- ***user_id*** (int) – Unique identifier of the target user.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.

- **write_timeout** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- **connect_timeout** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- **pool_timeout** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
telegram.ChatMember

Raises
telegram.error.TelegramError –

async get_chat_member_count(*chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also:

telegram.Chat.get_member_count

New in version 13.7.

Parameters
chat_id (int | str) – Unique identifier for the target chat or username of the target supergroup or channel (in the format `@channelusername`).

Keyword Arguments

- **read_timeout** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- **write_timeout** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- **connect_timeout** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- **pool_timeout** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
Number of members in the chat.

Return type
int

Raises
telegram.error.TelegramError –

async get_chat_menu_button(*chat_id=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to get the current value of the bot's menu button in a private chat, or the default menu button.

See also:

set_chat_menu_button(), *telegram.Chat.get_menu_button()*, *telegram.Chat.set_menu_button()*, *telegram.User.get_menu_button()*, *telegram.User.set_menu_button()*

New in version 20.0.

Parameters

chat_id (int, optional) – Unique identifier for the target private chat. If not specified, default bot's menu button will be returned.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, the current menu button is returned.

Return type

telegram.MenuButton

async get_custom_emoji_stickers(*custom_emoji_ids*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to get information about emoji stickers by their identifiers.

Parameters

custom_emoji_ids (List[str]) – List of custom emoji identifiers. At most *200* custom emoji identifiers can be specified.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

List[*telegram.Sticker*]

Raises

telegram.error.TelegramError –

async get_file(*file_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to get basic info about a file and prepare it for downloading. For the moment, bots can download files of up to *20 MB* in size. The file can then be downloaded with *telegram.File.download()*. 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 get_file 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.

Parameters

file_id – Either the file identifier or an object that has a file_id attribute to get file information about.

Returns

telegram.File

Raises

telegram.error.TelegramError –

async get_game_high_scores(*user_id*, *chat_id=None*, *message_id=None*, *inline_message_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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.

Note: 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 his neighbors are not among them. Please note that this behavior is subject to change.

See also:

telegram.CallbackQuery.get_game_high_scores

Parameters

- ***user_id*** (int) – Target user id.
- ***chat_id*** (int | str, optional) – Required if inline_message_id is not specified. Unique identifier for the target chat.
- ***message_id*** (int, optional) – Required if inline_message_id is not specified. Identifier of the sent message.
- ***inline_message_id*** (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

List[*telegram.GameHighScore*]

Raises

telegram.error.TelegramError –

async get_me(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

A simple method for testing your bot's auth token. Requires no parameters.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

A *telegram.User* instance representing that bot if the credentials are valid, None otherwise.

Return type

telegram.User

Raises

telegram.error.TelegramError –

async get_my_commands(*scope=None*, *language_code=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to get the current list of the bot's commands for the given scope and user language.

Parameters

- ***scope*** (*telegram.BotCommandScope*, optional) – An object, describing scope of users. Defaults to *telegram.BotCommandScopeDefault*.

  New in version 13.7.

- ***language_code*** (str, optional) – A two-letter ISO 639-1 language code or an empty string.

  New in version 13.7.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, the commands set for the bot. An empty list is returned if commands are not set.

Return type
List[*telegram.BotCommand*]

Raises
telegram.error.TelegramError –

async get_my_default_administrator_rights(*for_channels=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to get the current default administrator rights of the bot.

See also:

set_my_default_administrator_rights()

New in version 20.0.

Parameters
for_channels (bool, optional) – 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.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success.

Return type
telegram.ChatAdministratorRights

Raises
telegram.error.TelegramError –

async get_sticker_set(*name*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to get a sticker set.

Parameters
name (str) – Name of the sticker set.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.

- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
telegram.StickerSet

Raises
telegram.error.TelegramError –

async get_updates(*offset=None*, *limit=None*, *timeout=None*, *allowed_updates=None*, ***, *read_timeout=2*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to receive incoming updates using long polling.

Parameters

- ***offset*** (int, optional) – 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 this method is called with an offset higher than its *telegram.Update.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 forgotten.
- ***limit*** (int, optional) – Limits the number of updates to be retrieved. Values between 1-100 are accepted. Defaults to `100`.
- ***timeout*** (int, optional) – 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]), optional) – A list the types of updates you want your bot to receive. For example, specify [“message”, “edited_channel_post”, “callback_query”] to only receive updates of these types. See *telegram.Update* for a complete list of available update types. Specify an empty list to receive all updates except *telegram.Update.chat_member* (default). If not specified, the previous setting will be used. Please note that this parameter doesn't affect updates created before the call to the get_updates, so unwanted updates may be received for a short period of time.

Keyword Arguments

- ***read_timeout*** (float, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to `2`. *timeout* will be added to this value.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Note:

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.
3. To take full advantage of this library take a look at *telegram.ext.Updater*

Returns
List[*telegram.Update*]

Raises
telegram.error.TelegramError –

async get_user_profile_photos(*user_id*, *offset=None*, *limit=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to get a list of profile pictures for a user.

See also:

telegram.User.get_profile_photos()

Parameters

- ***user_id*** (int) – Unique identifier of the target user.
- ***offset*** (int, optional) – Sequential number of the first photo to be returned. By default, all photos are returned.
- ***limit*** (int, optional) – Limits the number of photos to be retrieved. Values between 1-100 are accepted. Defaults to `100`.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
telegram.UserProfilePhotos

Raises
telegram.error.TelegramError –

async get_webhook_info(***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to get current webhook status. Requires no parameters.

If the bot is using *get_updates()*, will return an object with the *telegram.WebhookInfo.url* field empty.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.

- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
telegram.WebhookInfo

property id

Unique identifier for this bot. Shortcut for the corresponding attribute of *bot*.

Type
int

async initialize()

Initialize resources used by this class. Currently calls *get_me()* to cache *bot* and calls *telegram.request.BaseRequest.initialize()* for the request objects used by this bot.

See also:

shutdown()

New in version 20.0.

property last_name

Optional. Bot's last name. Shortcut for the corresponding attribute of *bot*.

Type
str

async leaveChat(*chat_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *leave_chat()*

async leave_chat(*chat_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also:

telegram.Chat.leave

Parameters
chat_id (int | str) – Unique identifier for the target chat or username of the target supergroup or channel (in the format `@channelusername`).

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

property link

Convenience property. Returns the t.me link of the bot.

Type
str

async logOut(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *log_out()*

async log_out(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

  New in version 20.0.

Returns
On success

Return type
True

Raises
telegram.error.TelegramError –

property name

Bot's @username. Shortcut for the corresponding attribute of *bot*.

Type
str

async pinChatMessage(*chat_id*, *message_id*, *disable_notification=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *pin_chat_message()*

async pin_chat_message(*chat_id*, *message_id*, *disable_notification=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also:

telegram.Chat.pin_message, *telegram.User.pin_message*

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- ***message_id*** (int) – Identifier of a message to pin.
- ***disable_notification*** (bool, optional) – 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.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async promoteChatMember(*chat_id*, *user_id*, *can_change_info=None*, *can_post_messages=None*, *can_edit_messages=None*, *can_delete_messages=None*, *can_invite_users=None*, *can_restrict_members=None*, *can_pin_messages=None*, *can_promote_members=None*, *is_anonymous=None*, *can_manage_chat=None*, *can_manage_video_chats=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *promote_chat_member()*

async promote_chat_member(*chat_id*, *user_id*, *can_change_info=None*, *can_post_messages=None*, *can_edit_messages=None*, *can_delete_messages=None*, *can_invite_users=None*, *can_restrict_members=None*, *can_pin_messages=None*, *can_promote_members=None*, *is_anonymous=None*, *can_manage_chat=None*, *can_manage_video_chats=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also:

telegram.Chat.promote_member

Changed in version 20.0: The argument `can_manage_voice_chats` was renamed to *can_manage_video_chats* in accordance to Bot API 6.0.

Parameters

- ***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`, optional) – Pass `True`, if the administrator's presence in the chat is hidden.
- ***can_manage_chat*** (`bool`, optional) – Pass `True`, if the administrator can access the chat event log, chat statistics, message statistics in channels, see channel members, see anonymous administrators in supergroups and ignore slow mode. Implied by any other administrator privilege.

  New in version 13.4.
- ***can_manage_video_chats*** (`bool`, optional) – Pass `True`, if the administrator can manage video chats.

  New in version 20.0.
- ***can_change_info*** (`bool`, optional) – Pass `True`, if the administrator can change chat title, photo and other settings.
- ***can_post_messages*** (`bool`, optional) – Pass `True`, if the administrator can create channel posts, channels only.
- ***can_edit_messages*** (`bool`, optional) – Pass `True`, if the administrator can edit messages of other users and can pin messages, channels only.
- ***can_delete_messages*** (`bool`, optional) – Pass `True`, if the administrator can delete messages of other users.
- ***can_invite_users*** (`bool`, optional) – Pass `True`, if the administrator can invite new users to the chat.
- ***can_restrict_members*** (`bool`, optional) – Pass `True`, if the administrator can restrict, ban or unban chat members.
- ***can_pin_messages*** (`bool`, optional) – Pass `True`, if the administrator can pin messages, supergroups only.
- ***can_promote_members*** (`bool`, optional) – Pass `True`, if the administrator can add new administrators with a subset of his own privileges or demote administrators that he has promoted, directly or indirectly (promoted by administrators that were appointed by him).

Keyword Arguments

- ***read_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.

- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

property request

The *BaseRequest* object used by this bot.

> **Warning:** Requests to the Bot API are made by the various methods of this class. This attribute should *not* be used manually.

async restrictChatMember(*chat_id*, *user_id*, *permissions*, *until_date=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *restrict_chat_member()*

async restrict_chat_member(*chat_id*, *user_id*, *permissions*, *until_date=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also:

telegram.ChatPermissions.all_permissions(), *telegram.Chat.restrict_member*

Parameters

- ***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.
- ***until_date*** (int | datetime.datetime, optional) – 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. For timezone naive datetime.datetime objects, the default timezone of the bot will be used, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***permissions*** (*telegram.ChatPermissions*) – An object for new user permissions.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.

- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async revokeChatInviteLink(*chat_id*, *invite_link*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *revoke_chat_invite_link()*

async revoke_chat_invite_link(*chat_id*, *invite_link*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also:

telegram.Chat.revoke_invite_link

New in version 13.4.

Parameters

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

  Changed in version 20.0: Now also accepts *telegram.ChatInviteLink* instances.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
telegram.ChatInviteLink

Raises
telegram.error.TelegramError –

async sendAnimation(*chat_id*, *animation*, *duration=None*, *width=None*, *height=None*, *thumb=None*, *caption=None*, *parse_mode=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_animation()*

async sendAudio(*chat_id*, *audio*, *duration=None*, *performer=None*, *title=None*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *thumb=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_audio()*

async sendChatAction(*chat_id*, *action*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_chat_action()*

async sendContact(*chat_id*, *phone_number=None*, *first_name=None*, *last_name=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *vcard=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *contact=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_contact()*

async sendDice(*chat_id*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *emoji=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_dice()*

async sendDocument(*chat_id*, *document*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *thumb=None*, *disable_content_type_detection=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_document()*

async sendGame(*chat_id*, *game_short_name*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_game()*

async sendInvoice(*chat_id*, *title*, *description*, *payload*, *provider_token*, *currency*, *prices*, *start_parameter=None*, *photo_url=None*, *photo_size=None*, *photo_width=None*, *photo_height=None*, *need_name=None*, *need_phone_number=None*, *need_email=None*, *need_shipping_address=None*, *is_flexible=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *provider_data=None*, *send_phone_number_to_provider=None*, *send_email_to_provider=None*, *allow_sending_without_reply=None*, *max_tip_amount=None*, *suggested_tip_amounts=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for send_invoice()

async sendLocation(*chat_id*, *latitude=None*, *longitude=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *live_period=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *location=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for send_location()

async sendMediaGroup(*chat_id*, *media*, *disable_notification=None*, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for send_media_group()

async sendMessage(*chat_id*, *text*, *parse_mode=None*, *entities=None*, *disable_web_page_preview=None*, *disable_notification=None*, *protect_content=None*, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for send_message()

async sendPhoto(*chat_id*, *photo*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for send_photo()

async sendPoll(*chat_id*, *question*, *options*, *is_anonymous=None*, *type=None*, *allows_multiple_answers=None*, *correct_option_id=None*, *is_closed=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *explanation=None*, *explanation_parse_mode=None*, *open_period=None*, *close_date=None*, *allow_sending_without_reply=None*, *explanation_entities=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for send_poll()

async sendSticker(*chat_id*, *sticker*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for send_sticker()

async sendVenue(*chat_id*, *latitude=None*, *longitude=None*, *title=None*, *address=None*, *foursquare_id=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *foursquare_type=None*, *google_place_id=None*, *google_place_type=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *venue=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for send_venue()

async sendVideo(*chat_id*, *video*, *duration=None*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *width=None*, *height=None*, *parse_mode=None*, *supports_streaming=None*, *thumb=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_video()*

async sendVideoNote(*chat_id*, *video_note*, *duration=None*, *length=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *thumb=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_video_note()*

async sendVoice(*chat_id*, *voice*, *duration=None*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_voice()*

async send_animation(*chat_id*, *animation*, *duration=None*, *width=None*, *height=None*, *thumb=None*, *caption=None*, *parse_mode=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send animation files (GIF or H.264/MPEG-4 AVC video without sound). Bots can currently send animation files of up to *50 MB* in size, this limit may be changed in the future.

Note: *thumb* will be ignored for small files, for which Telegram can easily generate thumb nails. However, this behaviour is undocumented and might be changed by Telegram.

See also:

telegram.Message.reply_animation, *telegram.Chat.send_animation*, *telegram.User.send_animation*

Parameters

- ***chat_id*** (`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***animation*** (`str` | file object | `bytes` | `pathlib.Path` | *telegram.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. Lastly you can pass an existing *telegram.Animation* object to send.

  Changed in version 13.2: Accept `bytes` as input.
- ***duration*** (`int`, optional) – Duration of sent animation in seconds.
- ***width*** (`int`, optional) – Animation width.
- ***height*** (`int`, optional) – Animation height.
- ***thumb*** (file object | `bytes` | `pathlib.Path`, 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.

  Changed in version 13.2: Accept `bytes` as input.

- ***caption*** (str, optional) – Animation caption (may also be used when resending animations by file_id), 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in message text, which can be specified instead of *parse_mode*.
- ***disable_notification*** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.
- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message.
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- ***reply_markup*** (*InlineKeyboardMarkup* | *ReplyKeyboardMarkup* | *ReplyKeyboardRemove* | *ForceReply*, optional) – Additional interface options. An object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

Keyword Arguments

- ***filename*** (str, optional) – Custom file name for the animation, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the tempfile module.

  New in version 13.1.
- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to 20.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, the sent Message is returned.

Return type

telegram.Message

Raises

telegram.error.TelegramError –

async send_audio(*chat_id*, *audio*, *duration=None*, *performer=None*, *title=None*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *thumb=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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.

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 *send_voice()* method instead.

Note: The audio argument can be either a file_id, an URL or a file from disk `open(filename, 'rb')`

See also:

telegram.Message.reply_audio, *telegram.Chat.send_audio*, *telegram.User.send_audio*

Parameters

- ***chat_id*** (`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***audio*** (`str` | file object | `bytes` | `pathlib.Path` | *telegram.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. Lastly you can pass an existing *telegram.Audio* object to send.

  Changed in version 13.2: Accept `bytes` as input.
- ***caption*** (`str`, optional) – Audio caption, 0-*1024* characters after entities parsing.
- ***parse_mode*** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in message text, which can be specified instead of *parse_mode*.
- ***duration*** (`int`, optional) – Duration of sent audio in seconds.
- ***performer*** (`str`, optional) – Performer.
- ***title*** (`str`, optional) – Track name.
- ***disable_notification*** (`bool`, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***protect_content*** (`bool`, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.
- ***reply_to_message_id*** (`int`, optional) – If the message is a reply, ID of the original message.
- ***allow_sending_without_reply*** (`bool`, optional) – Pass `True`, if the message should be sent even if the specified replied-to message is not found.
- ***reply_markup*** (*InlineKeyboardMarkup* | *ReplyKeyboardMarkup* | *ReplyKeyboardRemove* | *ForceReply*, optional) – Additional interface options. An object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- ***thumb*** (file object | `bytes` | `pathlib.Path`, 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.

Changed in version 13.2: Accept `bytes` as input.

Keyword Arguments

- ***filename*** (`str`, optional) – Custom file name for the audio, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the `tempfile` module.

  New in version 13.1.

- ***read_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to `20`.
- ***connect_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (`dict`, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, the sent Message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async send_chat_action(*chat_id*, *action*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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). Telegram only recommends using this method when a response from the bot will take a noticeable amount of time to arrive.

See also:

telegram.Message.reply_chat_action, *telegram.Chat.send_action*, *telegram.User.send_chat_action*

Parameters

- ***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. For convenience look at the constants in *telegram.constants.ChatAction*.

Keyword Arguments

- ***read_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.

- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async send_contact(*chat_id*, *phone_number=None*, *first_name=None*, *last_name=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *vcard=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *contact=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send phone contacts.

Note: You can either supply *contact* or *phone_number* and *first_name* with optionally *last_name* and optionally *vcard*.

See also:

telegram.Message.reply_contact, *telegram.Chat.send_contact*, *telegram.User.send_contact*

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***phone_number*** (str, optional) – Contact's phone number.
- ***first_name*** (str, optional) – Contact's first name.
- ***last_name*** (str, optional) – Contact's last name.
- ***vcard*** (str, optional) – Additional data about the contact in the form of a vCard, 0-2048 bytes.
- ***disable_notification*** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.
- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message.
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- ***reply_markup*** (*InlineKeyboardMarkup* | *ReplyKeyboardMarkup* | *ReplyKeyboardRemove* | *ForceReply*, optional) – Additional interface options. An object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

Keyword Arguments

- ***contact*** (*telegram.Contact*, optional) – The contact to send.

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, the sent Message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async send_dice(*chat_id*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *emoji=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send an animated emoji that will display a random value.

See also:

telegram.Message.reply_dice, *telegram.Chat.send_dice*, *telegram.User.send_dice*

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***disable_notification*** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message.
- ***reply_markup*** (*InlineKeyboardMarkup* | *ReplyKeyboardMarkup* | *ReplyKeyboardRemove* | *ForceReply*, optional) – Additional interface options. An object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- ***emoji*** (str, optional) – Emoji on which the dice throw animation is based. Currently, must be one of *telegram.constants.DiceEmoji*. Dice can have values 1-6 for ‘🎲’, ‘🎯’ and ‘🎳’, values 1-5 for ‘🏀’ and ‘⚽’, and values 1-64 for ‘🎰’. Defaults to ‘🎲’.

  Changed in version 13.4: Added the ‘🎳’ emoji..
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.

- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, the sent Message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async send_document(*chat_id*, *document*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *thumb=None*, *disable_content_type_detection=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send general files.

Bots can currently send files of any type of up to *50 MB* in size, this limit may be changed in the future.

Note:

- The document argument can be either a file_id, an URL or a file from disk `open(filename, 'rb')`.
- Sending by URL will currently only work `GIF`, `PDF` & `ZIP` files.

See also:

telegram.Message.reply_document, *telegram.Chat.send_document*, *telegram.User.send_document*

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***document*** (str | file object | bytes | pathlib.Path | *telegram.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. Lastly you can pass an existing *telegram.Document* object to send.

  Changed in version 13.2: Accept bytes as input.
- ***caption*** (str, optional) – Document caption (may also be used when resending documents by file_id), 0-*1024* characters after entities parsing.
- ***disable_content_type_detection*** (bool, optional) – Disables automatic server-side content type detection for files uploaded using multipart/form-data.

- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in message text, which can be specified instead of *parse_mode*.
- ***disable_notification*** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.
- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message.
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- ***reply_markup*** (*InlineKeyboardMarkup* | *ReplyKeyboardMarkup* | *ReplyKeyboardRemove* | *ForceReply*, optional) – Additional interface options. An object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- ***thumb*** (file object | bytes | pathlib.Path, 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.

  Changed in version 13.2: Accept bytes as input.

Keyword Arguments

- ***filename*** (str, optional) – Custom file name for the document, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the tempfile module.
- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to 20.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, the sent Message is returned.

Return type

telegram.Message

Raises

telegram.error.TelegramError –

async send_game(*chat_id*, *game_short_name*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send a game.

See also:

telegram.Message.reply_game, *telegram.Chat.send_game*, *telegram.User.send_game*

Parameters

- ***chat_id*** (int | str) – 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.
- ***disable_notification*** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.
- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message.
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – An object for a new inline keyboard. If empty, one 'Play game_title' button will be shown. If not empty, the first button must launch the game.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, the sent Message is returned.

Return type

telegram.Message

Raises

telegram.error.TelegramError –

async send_invoice(*chat_id*, *title*, *description*, *payload*, *provider_token*, *currency*, *prices*, *start_parameter=None*, *photo_url=None*, *photo_size=None*, *photo_width=None*, *photo_height=None*, *need_name=None*, *need_phone_number=None*, *need_email=None*, *need_shipping_address=None*, *is_flexible=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *provider_data=None*, *send_phone_number_to_provider=None*, *send_email_to_provider=None*, *allow_sending_without_reply=None*, *max_tip_amount=None*, *suggested_tip_amounts=None*, *protect_content=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send invoices.

Warning: As of API 5.2 *start_parameter* is an optional argument and therefore the order of the arguments had to be changed. Use keyword arguments to make sure that the arguments are passed correctly.

See also:

telegram.Message.reply_invoice, *telegram.Chat.send_invoice*, *telegram.User.send_invoice*

Changed in version 13.5: As of Bot API 5.2, the parameter *start_parameter* is optional.

Parameters

- ***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) – Payments provider token, obtained via @BotFather.
- ***currency*** (str) – Three-letter ISO 4217 currency code, see more on currencies.
- ***prices*** (List[*telegram.LabeledPrice*) – Price breakdown, a list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.).
- ***max_tip_amount*** (int, 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`.

  New in version 13.5.
- ***suggested_tip_amounts*** (List[int], optional) – An 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*.

  New in version 13.5.
- ***start_parameter*** (str, optional) – 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.

  Changed in version 13.5: As of Bot API 5.2, this parameter is optional.
- ***provider_data*** (str | object, optional) – 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. When an object is passed, it will be encoded as JSON.
- ***photo_url*** (str, optional) – 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*** (str, optional) – Photo size.

- ***photo_width*** (int, optional) – Photo width.
- ***photo_height*** (int, optional) – Photo height.
- ***need_name*** (bool, optional) – Pass True, if you require the user's full name to complete the order.
- ***need_phone_number*** (bool, optional) – Pass True, if you require the user's phone number to complete the order.
- ***need_email*** (bool, optional) – Pass True, if you require the user's email to complete the order.
- ***need_shipping_address*** (bool, optional) – Pass True, if you require the user's shipping address to complete the order.
- ***send_phone_number_to_provider*** (bool, optional) – Pass True, if user's phone number should be sent to provider.
- ***send_email_to_provider*** (bool, optional) – Pass True, if user's email address should be sent to provider.
- ***is_flexible*** (bool, optional) – Pass True, if the final price depends on the shipping method.
- ***disable_notification*** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.
- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message.
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – An 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.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, the sent Message is returned.

Return type

telegram.Message

Raises

telegram.error.TelegramError –

async send_location(*chat_id*, *latitude=None*, *longitude=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *live_period=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, *allow_sending_without_reply=None*, *protect_content=None*, *, *location=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send point on the map.

Note: You can either supply a *latitude* and *longitude* or a *location*.

See also:

telegram.Message.reply_location, *telegram.Chat.send_location*, *telegram.User.send_location*

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***latitude*** (float, optional) – Latitude of location.
- ***longitude*** (float, optional) – Longitude of location.
- ***horizontal_accuracy*** (int, optional) – The radius of uncertainty for the location, measured in meters; 0-*1500*.
- ***live_period*** (int, optional) – Period in seconds for which the location will be updated, should be between 60 and 86400.
- ***heading*** (int, 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, optional) – For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and *360* if specified.
- ***disable_notification*** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.
- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message.
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- ***reply_markup*** (*InlineKeyboardMarkup* | *ReplyKeyboardMarkup* | *ReplyKeyboardRemove* | *ForceReply*, optional) – Additional interface options. An object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

Keyword Arguments

- ***location*** (*telegram.Location*, optional) – The location to send.
- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.

- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, the sent Message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async send_media_group(*chat_id*, *media*, *disable_notification=None*, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also:

telegram.Message.reply_media_group, *telegram.Chat.send_media_group*, *telegram.User.send_media_group*

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***media*** (List[*telegram.InputMediaAudio*, *telegram.InputMediaDocument*, *telegram.InputMediaPhoto*, *telegram.InputMediaVideo*]) – An array describing messages to be sent, must include 2–10 items.
- ***disable_notification*** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.
- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message.
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to `20`.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
An array of the sent Messages.

Return type
List[*telegram.Message*]

Raises
telegram.error.TelegramError –

async **send_message**(*chat_id*, *text*, *parse_mode=None*, *entities=None*, *disable_web_page_preview=None*, *disable_notification=None*, *protect_content=None*, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send text messages.

See also:

telegram.Message.reply_text, *telegram.Chat.send_message*, *telegram.User.send_message*

Parameters

- ***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. Max *4096* characters after entities parsing.
- ***parse_mode*** (str) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in message text, which can be specified instead of *parse_mode*.
- ***disable_web_page_preview*** (bool, optional) – Disables link previews for links in this message.
- ***disable_notification*** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***protect_content*** (bool, optional) – Protects the contents of sent messages from forwarding and saving.

  New in version 13.10.
- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message.
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- ***reply_markup*** (*InlineKeyboardMarkup* | *ReplyKeyboardMarkup* | *ReplyKeyboardRemove* | *ForceReply*, optional) – Additional interface options. An object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.

- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, the sent message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async **send_photo**(*chat_id*, *photo*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send photos.

Note: The photo argument can be either a file_id, an URL or a file from disk `open(filename, 'rb')`

See also:

telegram.Message.reply_photo, *telegram.Chat.send_photo*, *telegram.User.send_photo*

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***photo*** (str | file object | bytes | pathlib.Path | *telegram.PhotoSize*) – 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. Lastly you can pass an existing *telegram.PhotoSize* object to send.

  Changed in version 13.2: Accept bytes as input.
- ***caption*** (str, optional) – Photo caption (may also be used when resending photos by file_id), 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in message text, which can be specified instead of *parse_mode*.
- ***disable_notification*** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message.
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- ***reply_markup*** (*InlineKeyboardMarkup* | *ReplyKeyboardMarkup* | *ReplyKeyboardRemove* | *ForceReply*, optional) – Additional interface options. An object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

Keyword Arguments

- ***filename*** (str, optional) – Custom file name for the photo, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the tempfile module.

  New in version 13.1.
- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to 20.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, the sent Message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async send_poll(*chat_id*, *question*, *options*, *is_anonymous=None*, *type=None*, *allows_multiple_answers=None*, *correct_option_id=None*, *is_closed=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *explanation=None*, *explanation_parse_mode=None*, *open_period=None*, *close_date=None*, *allow_sending_without_reply=None*, *explanation_entities=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send a native poll.

See also:

telegram.Message.reply_poll, *telegram.Chat.send_poll*, *telegram.User.send_poll*

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- ***question*** (str) – Poll question, 1-*300* characters.
- ***options*** (List[str]) – List of answer options, 2-*10* strings 1-*100* characters each.
- ***is_anonymous*** (bool, optional) – True, if the poll needs to be anonymous, defaults to True.

- ***type*** (str, optional) – Poll type, *'quiz'* or *'regular'*, defaults to *'regular'*.
- ***allows_multiple_answers*** (bool, optional) – True, if the poll allows multiple answers, ignored for polls in quiz mode, defaults to False.
- ***correct_option_id*** (int, optional) – 0-based identifier of the correct answer option, required for polls in quiz mode.
- ***explanation*** (str, 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 with at most 2 line feeds after entities parsing.
- ***explanation_parse_mode*** (str, optional) – Mode for parsing entities in the explanation. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***explanation_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in message text, which can be specified instead of *explanation_parse_mode*.
- ***open_period*** (int, optional) – Amount of time in seconds the poll will be active after creation, 5-600. Can't be used together with *close_date*.
- ***close_date*** (int | datetime.datetime, optional) – 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*. For timezone naive datetime.datetime objects, the default timezone of the bot will be used, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***is_closed*** (bool, optional) – Pass True, if the poll needs to be immediately closed. This can be useful for poll preview.
- ***disable_notification*** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.
- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message.
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- ***reply_markup*** (*InlineKeyboardMarkup* | *ReplyKeyboardMarkup* | *ReplyKeyboardRemove* | *ForceReply*, optional) – Additional interface options. An object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, the sent Message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async **send_sticker**(*chat_id*, *sticker*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send static `.WEBP`, animated `.TGS`, or video `.WEBM` stickers.

Note: The *sticker* argument can be either a file_id, an URL or a file from disk `open(filename, 'rb')`

See also:

telegram.Message.reply_sticker, *telegram.Chat.send_sticker*, *telegram.User.send_sticker*

Parameters

- ***chat_id*** (`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***sticker*** (`str` | file object | `bytes` | `pathlib.Path` | *telegram.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 file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing *telegram.Sticker* object to send.

  Changed in version 13.2: Accept `bytes` as input.
- ***disable_notification*** (`bool`, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***protect_content*** (`bool`, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.
- ***reply_to_message_id*** (`int`, optional) – If the message is a reply, ID of the original message.
- ***allow_sending_without_reply*** (`bool`, optional) – Pass `True`, if the message should be sent even if the specified replied-to message is not found.
- ***reply_markup*** (*InlineKeyboardMarkup* | *ReplyKeyboardMarkup* | *ReplyKeyboardRemove* | *ForceReply*, optional) – Additional interface options. An object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

Keyword Arguments

- ***read_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to `20`.
- ***connect_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.

- **pool_timeout** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, the sent Message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async **send_venue**(*chat_id*, *latitude=None*, *longitude=None*, *title=None*, *address=None*, *foursquare_id=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *foursquare_type=None*, *google_place_id=None*, *google_place_type=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *venue=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send information about a venue.

Note:

- You can either supply *venue*, or *latitude*, *longitude*, *title* and *address* and optionally *foursquare_id* and *foursquare_type* or optionally *google_place_id* and *google_place_type*.
- Foursquare details and Google Place details are mutually exclusive. However, this behaviour is undocumented and might be changed by Telegram.

See also:

telegram.Message.reply_venue, *telegram.Chat.send_venue*, *telegram.User.send_venue*

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **latitude** (float, optional) – Latitude of venue.
- **longitude** (float, optional) – Longitude of venue.
- **title** (str, optional) – Name of the venue.
- **address** (str, optional) – Address of the venue.
- **foursquare_id** (str, optional) – Foursquare identifier of the venue.
- **foursquare_type** (str, optional) – Foursquare type of the venue, if known. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”.)
- **google_place_id** (str, optional) – Google Places identifier of the venue.
- **google_place_type** (str, optional) – Google Places type of the venue. (See supported types.)
- **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.

- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message.

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.

- ***reply_markup*** (*InlineKeyboardMarkup* | *ReplyKeyboardMarkup* | *ReplyKeyboardRemove* | *ForceReply*, optional) – Additional interface options. An object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

Keyword Arguments

- ***venue*** (*telegram.Venue*, optional) – The venue to send.
- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, the sent Message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async send_video(*chat_id*, *video*, *duration=None*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *width=None*, *height=None*, *parse_mode=None*, *supports_streaming=None*, *thumb=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send video files, Telegram clients support mp4 videos (other formats may be sent as Document).

Bots can currently send video files of up to *50 MB* in size, this limit may be changed in the future.

Note:

- The *video* argument can be either a file_id, an URL or a file from disk `open(filename, 'rb')`
- *thumb* will be ignored for small video files, for which Telegram can easily generate thumbnails. However, this behaviour is undocumented and might be changed by Telegram.

See also:

telegram.Message.reply_video, *telegram.Chat.send_video*, *telegram.User.send_video*

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- ***video*** (str | file object | bytes | pathlib.Path | *telegram.Video*) – Video file to send. Pass a file_id as String to send an video file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an video file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing *telegram.Video* object to send.

  Changed in version 13.2: Accept bytes as input.
- ***duration*** (int, optional) – Duration of sent video in seconds.
- ***width*** (int, optional) – Video width.
- ***height*** (int, optional) – Video height.
- ***caption*** (str, optional) – Video caption (may also be used when resending videos by file_id), 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in message text, which can be specified instead of *parse_mode*.
- ***supports_streaming*** (bool, optional) – Pass True, if the uploaded video is suitable for streaming.
- ***disable_notification*** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.
- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message.
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- ***reply_markup*** (*InlineKeyboardMarkup* | *ReplyKeyboardMarkup* | *ReplyKeyboardRemove* | *ForceReply*, optional) – Additional interface options. An object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- ***thumb*** (file object | bytes | pathlib.Path, 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.

  Changed in version 13.2: Accept bytes as input.

Keyword Arguments

- ***filename*** (str, optional) – Custom file name for the video, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the tempfile module.

  New in version 13.1.
- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.

- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to 20.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, the sent Message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async send_video_note(*chat_id*, *video_note*, *duration=None*, *length=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *thumb=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

As of v.4.0, Telegram clients support rounded square mp4 videos of up to 1 minute long. Use this method to send video messages.

Note:

- The *video_note* argument can be either a file_id or a file from disk `open(filename, 'rb')`
- *thumb* will be ignored for small video files, for which Telegram can easily generate thumbnails. However, this behaviour is undocumented and might be changed by Telegram.

See also:

telegram.Message.reply_video_note, *telegram.Chat.send_video_note*, *telegram.User.send_video_note*

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***video_note*** (str | file object | bytes | pathlib.Path | *telegram.VideoNote*) – 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. Or you can pass an existing *telegram.VideoNote* object to send. Sending video notes by a URL is currently unsupported.

  Changed in version 13.2: Accept bytes as input.
- ***duration*** (int, optional) – Duration of sent video in seconds.
- ***length*** (int, optional) – Video width and height, i.e. diameter of the video message.
- ***disable_notification*** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

New in version 13.10.

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message.
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- ***reply_markup*** (*InlineKeyboardMarkup* | *ReplyKeyboardMarkup* | *ReplyKeyboardRemove* | *ForceReply*, optional) – Additional interface options. An object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- ***thumb*** (file object | bytes | pathlib.Path, 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.

  Changed in version 13.2: Accept bytes as input.

Keyword Arguments

- ***filename*** (str, optional) – Custom file name for the video note, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the tempfile module.

  New in version 13.1.
- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to 20.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, the sent Message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async send_voice(*chat_id*, *voice*, *duration=None*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send audio files, if you want Telegram clients to display the file as a playable voice message. For this to work, your audio must be in an .ogg file encoded with OPUS (other formats may be sent as Audio or Document). Bots can currently send voice messages of up to *50 MB* in size, this limit may be changed in the future.

Note:

- The *voice* argument can be either a file_id, an URL or a file from disk open(filename, 'rb').

- To use this method, the file must have the type *audio/ogg* and be no more than 1MB in size. 1-20MB voice notes will be sent as files.

See also:

telegram.Message.reply_voice, *telegram.Chat.send_voice*, *telegram.User.send_voice*

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- ***voice*** (str | file object | bytes | pathlib.Path | *telegram.Voice*) – Voice file to send. Pass a file_id as String to send an voice file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an voice file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing *telegram.Voice* object to send.

  Changed in version 13.2: Accept bytes as input.
- ***caption*** (str, optional) – Voice message caption, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in message text, which can be specified instead of *parse_mode*.
- ***duration*** (int, optional) – Duration of the voice message in seconds.
- ***disable_notification*** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.
- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message.
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- ***reply_markup*** (*InlineKeyboardMarkup* | *ReplyKeyboardMarkup* | *ReplyKeyboardRemove* | *ForceReply*, optional) – Additional interface options. An object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

Keyword Arguments

- ***filename*** (str, optional) – Custom file name for the voice, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the tempfile module.

  New in version 13.1.
- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to 20.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.

- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, the sent Message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async setChatAdministratorCustomTitle(*chat_id*, *user_id*, *custom_title*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_chat_administrator_custom_title()*

async setChatDescription(*chat_id*, *description=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_chat_description()*

async setChatMenuButton(*chat_id=None*, *menu_button=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_chat_menu_button()*

async setChatPermissions(*chat_id*, *permissions*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_chat_permissions()*

async setChatPhoto(*chat_id*, *photo*, ***, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_chat_photo()*

async setChatStickerSet(*chat_id*, *sticker_set_name*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_chat_sticker_set()*

async setChatTitle(*chat_id*, *title*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_chat_title()*

async setGameScore(*user_id*, *score*, *chat_id=None*, *message_id=None*, *inline_message_id=None*, *force=None*, *disable_edit_message=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_game_score()*

async setMyCommands(*commands*, *scope=None*, *language_code=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_my_commands()*

async setMyDefaultAdministratorRights(*rights=None*, *for_channels=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_my_default_administrator_rights()*

async setPassportDataErrors(*user_id*, *errors*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_passport_data_errors()*

async setStickerPositionInSet(*sticker*, *position*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_sticker_position_in_set()*

async setStickerSetThumb(*name*, *user_id*, *thumb=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_sticker_set_thumb()*

async setWebhook(*url*, *certificate=None*, *max_connections=None*, *allowed_updates=None*, *ip_address=None*, *drop_pending_updates=None*, *secret_token=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_webhook()*

async set_chat_administrator_custom_title(*chat_id*, *user_id*, *custom_title*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to set a custom title for administrators promoted by the bot in a supergroup. The bot must be an administrator for this to work.

See also:

telegram.Chat.set_administrator_custom_title

Parameters

- ***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 administrator.
- ***custom_title*** (str) – New custom title for the administrator; 0-16 characters, emoji are not allowed.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

async set_chat_description(*chat_id*, *description=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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 admin rights.

See also:

telegram.Chat.set_description

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***description*** (str, optional) – New chat description, 0-255 characters.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async set_chat_menu_button(*chat_id=None*, *menu_button=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to change the bot's menu button in a private chat, or the default menu button.

See also:

get_chat_menu_button(), *telegram.Chat.set_menu_button()*, *telegram.Chat.get_menu_button()*, meth:*telegram.User.set_menu_button*, *telegram.User.get_menu_button()*

New in version 20.0.

Parameters

- ***chat_id*** (int, optional) – Unique identifier for the target private chat. If not specified, default bot's menu button will be changed
- ***menu_button*** (*telegram.MenuButton*, optional) – An object for the new bot's menu button. Defaults to *telegram.MenuButtonDefault*.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.

- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

async set_chat_permissions(*chat_id*, *permissions*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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 *telegram.ChatMemberAdministrator.can_restrict_members* admin rights.

See also:

telegram.Chat.set_permissions

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target supergroup (in the format *@supergroupusername*).
- ***permissions*** (*telegram.ChatPermissions*) – New default chat permissions.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async set_chat_photo(*chat_id*, *photo*, ***, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to set a new profile photo for the chat.

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

See also:

telegram.Chat.set_photo

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- ***photo*** (file object | bytes | pathlib.Path) – New chat photo.

  Changed in version 13.2: Accept bytes as input.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async set_chat_sticker_set(*chat_id*, *sticker_set_name*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to set a new group sticker set for a supergroup. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights. Use the field *telegram.Chat.can_set_sticker_set* optionally returned in *get_chat()* requests to check if the bot can use this method.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername).
- ***sticker_set_name*** (str) – Name of the sticker set to be set as the group sticker set.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.

- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

async set_chat_title(*chat_id*, *title*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also:

telegram.Chat.set_title

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***title*** (str) – New chat title, 1-255 characters.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async set_game_score(*user_id*, *score*, *chat_id=None*, *message_id=None*, *inline_message_id=None*, *force=None*, *disable_edit_message=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to set the score of the specified user in a game message.

Parameters

- ***user_id*** (int) – User identifier.
- ***score*** (int) – New score, must be non-negative.
- ***force*** (bool, optional) – Pass True, if the high score is allowed to decrease. This can be useful when fixing mistakes or banning cheaters.
- ***disable_edit_message*** (bool, optional) – Pass True, if the game message should not be automatically edited to include the current scoreboard.

- ***chat_id*** (int | str, optional) – Required if inline_message_id is not specified. Unique identifier for the target chat.
- ***message_id*** (int, optional) – Required if inline_message_id is not specified. Identifier of the sent message.
- ***inline_message_id*** (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

The edited message. If the message is not an inline message , True.

Return type

telegram.Message

Raises

telegram.error.TelegramError – If the new score is not greater than the user's current score in the chat and force is False.

async set_my_commands(*commands*, *scope=None*, *language_code=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to change the list of the bot's commands. See the Telegram docs for more details about bot commands.

Parameters

- ***commands*** (List[*BotCommand* | (str, str)]) – A list of bot commands to be set as the list of the bot's commands. At most 100 commands can be specified.
- ***scope*** (*telegram.BotCommandScope*, optional) – An object, describing scope of users for which the commands are relevant. Defaults to *telegram.BotCommandScopeDefault*.

  New in version 13.7.

- ***language_code*** (str, optional) – 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.

  New in version 13.7.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.

- **pool_timeout** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async set_my_default_administrator_rights(*rights=None*, *for_channels=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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 are free to modify the list before adding the bot.

See also:

get_my_default_administrator_rights()

New in version 20.0.

Parameters

- **rights** (*telegram.ChatAdministratorRights*, optional) – A *telegram.ChatAdministratorRights* object describing new default administrator rights. If not specified, the default administrator rights will be cleared.
- **for_channels** (bool, optional) – 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.

Keyword Arguments

- **read_timeout** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- **write_timeout** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- **connect_timeout** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- **pool_timeout** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
Returns True on success.

Return type
bool

Raises
telegram.error.TelegramError –

async set_passport_data_errors(*user_id*, *errors*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

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.

Parameters

- ***user_id*** (int) – User identifier
- ***errors*** (List[*PassportElementError*]) – An array describing the errors.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async set_sticker_position_in_set(*sticker*, *position*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

Parameters

- ***sticker*** (str) – File identifier of the sticker.
- ***position*** (int) – New sticker position in the set, zero-based.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, `True` is returned.

Return type
`bool`

Raises
telegram.error.TelegramError –

async **set_sticker_set_thumb**(*name*, *user_id*, *thumb=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

Note: The *thumb* can be either a file_id, an URL or a file from disk `open(filename, 'rb')`

Parameters

- ***name*** (`str`) – Sticker set name
- ***user_id*** (`int`) – User identifier of created sticker set owner.
- ***thumb*** (`str` | file object | `bytes` | `pathlib.Path`, optional) – A **PNG** image with the thumbnail, must be up to 128 kilobytes in size and have width and height exactly 100px, or a **TGS** animation with the thumbnail up to 32 kilobytes in size; see https://core.telegram.org/stickers#animated-sticker-requirements for animated sticker technical requirements, or a **WEBM** video with the thumbnail up to 32 kilobytes in size; see https://core.telegram.org/stickers#video-sticker-requirements for video sticker technical requirements. Pass a file_id as a String to send a file that already exists on the Telegram servers, pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data. Animated sticker set thumbnails can't be uploaded via HTTP URL.

  Changed in version 13.2: Accept `bytes` as input.

Keyword Arguments

- ***read_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (`dict`, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, `True` is returned.

Return type
`bool`

Raises
telegram.error.TelegramError –

async set_webhook(*url*, *certificate=None*, *max_connections=None*, *allowed_updates=None*, *ip_address=None*, *drop_pending_updates=None*, *secret_token=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to specify a url and receive incoming updates via an outgoing webhook. Whenever there is an update for the bot, Telegram will send an HTTPS POST request to the specified url, containing An Update. In case of an unsuccessful request, Telegram will give up after a reasonable amount of attempts.

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.

Note: The certificate argument should be a file from disk `open(filename, 'rb')`.

Parameters

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

  New in version 20.0.

Keyword Arguments

- ***read_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.

- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Note:

1. You will not be able to receive updates using *get_updates()* for 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: *telegram.constants.SUPPORTED_WEBHOOK_PORTS*.

If you're having any trouble setting up webhooks, please check out this guide to Webhooks.

Returns
bool On success, True is returned.

Raises
telegram.error.TelegramError –

async shutdown()

Stop & clear resources used by this class. Currently just calls *telegram.request.BaseRequest.shutdown()* for the request objects used by this bot.

See also:

initialize()

New in version 20.0.

async stopMessageLiveLocation(*chat_id=None*, *message_id=None*, *inline_message_id=None*, *reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *stop_message_live_location()*

async stopPoll(*chat_id*, *message_id*, *reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *stop_poll()*

async stop_message_live_location(*chat_id=None*, *message_id=None*, *inline_message_id=None*, *reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also:

telegram.Message.stop_live_location *telegram.CallbackQuery.stop_message_live_location*

Parameters

- ***chat_id*** (int | str) – 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, optional) – Required if inline_message_id is not specified. Identifier of the sent message with live location to stop.
- ***inline_message_id*** (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – An object for a new inline keyboard.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, if edited message is not an inline message, the edited message is returned, otherwise True is returned.

Return type

telegram.Message

async stop_poll(*chat_id*, *message_id*, *reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to stop a poll which was sent by the bot.

See also:

telegram.Message.stop_poll

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***message_id*** (int) – Identifier of the original message with the poll.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – An object for a new message inline keyboard.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, the stopped Poll is returned.

Return type
telegram.Poll

Raises
telegram.error.TelegramError –

property supports_inline_queries

Bot's *telegram.User.supports_inline_queries* attribute. Shortcut for the corresponding attribute of *bot*.

Type
bool

to_dict()

See *telegram.TelegramObject.to_dict()*.

async unbanChatMember(*chat_id*, *user_id*, *only_if_banned=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *unban_chat_member()*

async unbanChatSenderChat(*chat_id*, *sender_chat_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *unban_chat_sender_chat()*

async unban_chat_member(*chat_id*, *user_id*, *only_if_banned=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to unban a previously kicked user in a supergroup or channel.

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

See also:

telegram.Chat.unban_member

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target supergroup or channel (in the format `@channelusername`).
- ***user_id*** (int) – Unique identifier of the target user.
- ***only_if_banned*** (bool, optional) – Do nothing if the user is not banned.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.

- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async unban_chat_sender_chat(*chat_id*, *sender_chat_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also:

telegram.Chat.unban_chat

New in version 13.9.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target supergroup or channel (in the format @channelusername).
- ***sender_chat_id*** (int) – Unique identifier of the target sender chat.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async unpinAllChatMessages(*chat_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *unpin_all_chat_messages()*

async unpinChatMessage(*chat_id*, *message_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *unpin_chat_message()*

async unpin_all_chat_messages(*chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also:

telegram.Chat.unpin_all_messages, *telegram.User.unpin_all_messages*

Parameters

chat_id (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

async unpin_chat_message(*chat_id*, *message_id=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also:

telegram.Chat.unpin_message, *telegram.User.unpin_message*

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- ***message_id*** (int, optional) – Identifier of a message to unpin. If not specified, the most recent pinned message (by sending date) will be unpinned.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.

- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async uploadStickerFile(*user_id*, *png_sticker*, ***, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *upload_sticker_file()*

async upload_sticker_file(*user_id*, *png_sticker*, ***, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to upload a .PNG file with a sticker for later use in *create_new_sticker_set()* and *add_sticker_to_set()* methods (can be used multiple times).

Note: The *png_sticker* argument can be either a file_id, an URL or a file from disk `open(filename, 'rb')`

Parameters

- ***user_id*** (int) – User identifier of sticker file owner.
- ***png_sticker*** (str | file object | bytes | pathlib.Path) – **PNG** image with the sticker, must be up to 512 kilobytes in size, dimensions must not exceed 512px, and either width or height must be exactly 512px.

  Changed in version 13.2: Accept bytes as input.

Keyword Arguments

- ***read_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***api_kwargs*** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, the uploaded File is returned.

Return type
telegram.File

Raises
telegram.error.TelegramError –

property username

Bot's username. Shortcut for the corresponding attribute of *bot*.

Type
str

telegram.BotCommand

class telegram.**BotCommand**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a bot command.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *command* and *description* are equal.

Parameters

- ***command*** (str) – Text of the command; 1-32 characters. Can contain only lowercase English letters, digits and underscores.
- ***description*** (str) – Description of the command; 1-256 characters.

command

Text of the command.

Type
str

description

Description of the command.

Type
str

telegram.BotCommandScope

class telegram.**BotCommandScope**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

Base class for objects that represent the scope to which bot commands are applied. Currently, the following 7 scopes are supported:

- *telegram.BotCommandScopeDefault*
- *telegram.BotCommandScopeAllPrivateChats*
- *telegram.BotCommandScopeAllGroupChats*
- *telegram.BotCommandScopeAllChatAdministrators*
- *telegram.BotCommandScopeChat*
- *telegram.BotCommandScopeChatAdministrators*
- *telegram.BotCommandScopeChatMember*

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *type* is equal. For subclasses with additional attributes, the notion of equality is overridden.

Note: Please see the official docs on how Telegram determines which commands to display.

New in version 13.7.

Parameters
type (str) – Scope type.

type
Scope type.

Type
str

ALL_CHAT_ADMINISTRATORS = 'all_chat_administrators'
telegram.constants.BotCommandScopeType.ALL_CHAT_ADMINISTRATORS

ALL_GROUP_CHATS = 'all_group_chats'
telegram.constants.BotCommandScopeType.ALL_GROUP_CHATS

ALL_PRIVATE_CHATS = 'all_private_chats'
telegram.constants.BotCommandScopeType.ALL_PRIVATE_CHATS

CHAT = 'chat'
telegram.constants.BotCommandScopeType.CHAT

CHAT_ADMINISTRATORS = 'chat_administrators'
telegram.constants.BotCommandScopeType.CHAT_ADMINISTRATORS

CHAT_MEMBER = 'chat_member'
telegram.constants.BotCommandScopeType.CHAT_MEMBER

DEFAULT = 'default'
telegram.constants.BotCommandScopeType.DEFAULT

classmethod de_json(*data*, *bot*)
Converts JSON data to the appropriate *BotCommandScope* object, i.e. takes care of selecting the correct subclass.

Parameters
- ***data*** (Dict[str, ...]) – The JSON data.
- ***bot*** (*telegram.Bot*) – The bot associated with this object.

Returns
The Telegram object.

telegram.BotCommandScopeAllChatAdministrators

class telegram.**BotCommandScopeAllChatAdministrators**(**args*, ***kwargs*)
Bases: *telegram.BotCommandScope*

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

New in version 13.7.

type
Scope type *'all_chat_administrators'*.

Type
str

telegram.BotCommandScopeAllGroupChats

class telegram.**BotCommandScopeAllGroupChats**(**args*, ***kwargs*)

Bases: *telegram.BotCommandScope*

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

New in version 13.7.

type

Scope type *'all_group_chats'*.

Type
str

telegram.BotCommandScopeAllPrivateChats

class telegram.**BotCommandScopeAllPrivateChats**(**args*, ***kwargs*)

Bases: *telegram.BotCommandScope*

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

New in version 13.7.

type

Scope type *'all_private_chats'*.

Type
str

telegram.BotCommandScopeChat

class telegram.**BotCommandScopeChat**(**args*, ***kwargs*)

Bases: *telegram.BotCommandScope*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *type* and *chat_id* are equal.

New in version 13.7.

Parameters
chat_id (str | int) – Unique identifier for the target chat or username of the target supergroup (in the format `@supergroupusername`)

type

Scope type *'chat'*.

Type
str

chat_id

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

Type
str | int

telegram.BotCommandScopeChatAdministrators

class telegram.**BotCommandScopeChatAdministrators**(**args*, ***kwargs*)

Bases: *telegram.BotCommandScope*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *type* and *chat_id* are equal.

New in version 13.7.

Parameters

chat_id (str | int) – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername)

type

Scope type *'chat_administrators'*.

Type

str

chat_id

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

Type

str | int

telegram.BotCommandScopeChatMember

class telegram.**BotCommandScopeChatMember**(**args*, ***kwargs*)

Bases: *telegram.BotCommandScope*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *type*, *chat_id* and *user_id* are equal.

New in version 13.7.

Parameters

- ***chat_id*** (str | int) – 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.

type

Scope type *'chat_member'*.

Type

str

chat_id

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

Type

str | int

user_id

Unique identifier of the target user.

Type

int

telegram.BotCommandScopeDefault

class telegram.**BotCommandScopeDefault**(**args*, ***kwargs*)

Bases: *telegram.BotCommandScope*

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

New in version 13.7.

type

Scope type *'default'*.

Type
str

telegram.CallbackQuery

class telegram.**CallbackQuery**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents an incoming callback query from a callback button in an inline keyboard.

If the button that originated the query was attached to a message sent by the bot, the field *message* will be present. If the button was attached to a message sent via the bot (in inline mode), the field *inline_message_id* will be present.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *id* is equal.

Note:

- In Python `from` is a reserved word use *from_user* instead.
- Exactly one of the fields *data* or *game_short_name* will be present.
- After the user presses an inline button, Telegram clients will display a progress bar until you call *answer*. It is, therefore, necessary to react by calling *telegram.Bot.answer_callback_query* even if no notification to the user is needed (e.g., without specifying any of the optional parameters).
- If you're using *telegram.ext.ExtBot.arbitrary_callback_data*, *data* may be an instance of *telegram.ext.InvalidCallbackData*. This will be the case, if the data associated with the button triggering the *telegram.CallbackQuery* was already deleted or if *data* was manipulated by a malicious client.

  New in version 13.6.

Parameters

- ***id*** (str) – Unique identifier for this query.
- ***from_user*** (*telegram.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 games.
- ***message*** (*telegram.Message*, optional) – Message with the callback button that originated the query. Note that message content and message date will not be available if the message is too old.
- ***data*** (str, optional) – Data associated with the callback button. Be aware that the message, which originated the query, can contain no callback buttons with this data.

- ***inline_message_id*** (str, optional) – Identifier of the message sent via the bot in inline mode, that originated the query.
- ***game_short_name*** (str, optional) – Short name of a Game to be returned, serves as the unique identifier for the game
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.

id

Unique identifier for this query.

Type
str

from_user

Sender.

Type
telegram.User

chat_instance

Global identifier, uniquely corresponding to the chat to which the message with the callback button was sent.

Type
str

message

Optional. Message with the callback button that originated the query.

Type
telegram.Message

data

Optional. Data associated with the callback button.

Tip: The value here is the same as the value passed in *telegram.InlineKeyboardButton.callback_data*.

Type
str|object

inline_message_id

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

Type
str

game_short_name

Optional. Short name of a Game to be returned.

Type
str

bot

The Bot to use for instance methods.

Type
telegram.Bot, optional

MAX_ANSWER_TEXT_LENGTH = 200

telegram.constants.CallbackQueryLimit.ANSWER_CALLBACK_QUERY_TEXT_LENGTH

New in version 13.2.

async answer(*text=None*, *show_alert=None*, *url=None*, *cache_time=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.answer_callback_query(update.callback_query.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.answer_callback_query()*.

Returns
On success, `True` is returned.

Return type
`bool`

async copy_message(*chat_id*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *disable_notification=None*, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *reply_markup=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await update.callback_query.message.copy(
    from_chat_id=update.message.chat_id,
    message_id=update.message.message_id,
    *args,
    **kwargs
)
```

For the documentation of the arguments, please see *telegram.Message.copy()*.

Returns
On success, returns the MessageId of the sent message.

Return type
telegram.MessageId

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

async delete_message(***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await update.callback_query.message.delete(*args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Message.delete()*.

Returns
On success, `True` is returned.

Return type
`bool`

async edit_message_caption(*caption=None*, *reply_markup=None*, *parse_mode=None*, *caption_entities=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for either:

```
await update.callback_query.message.edit_caption(*args, **kwargs)
```

or:

```
await bot.edit_message_caption(
    inline_message_id=update.callback_query.inline_message_id, *args,␣
↪**kwargs,
)
```

For the documentation of the arguments, please see *telegram.Bot.edit_message_caption()* and *telegram.Message.edit_caption()*.

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

Return type
telegram.Message

async edit_message_live_location(*latitude=None*, *longitude=None*, *reply_markup=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, ***, *location=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for either:

```
await update.callback_query.message.edit_live_location(*args, **kwargs)
```

or:

```
await bot.edit_message_live_location(
    inline_message_id=update.callback_query.inline_message_id, *args,␣
↪**kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.edit_message_live_location()* and *telegram.Message.edit_live_location()*.

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

Return type
telegram.Message

async edit_message_media(*media*, *reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for either:

```
await update.callback_query.message.edit_media(*args, **kwargs)
```

or:

```
await bot.edit_message_media(
    inline_message_id=update.callback_query.inline_message_id, *args,␣
↪**kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.edit_message_media()* and *telegram.Message.edit_media()*.

Returns
On success, if edited message is not an inline message, the edited Message is returned, otherwise `True` is returned.

Return type
telegram.Message

async edit_message_reply_markup(*reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for either:

```
await update.callback_query.message.edit_reply_markup(*args, **kwargs)
```

or:

```
await bot.edit_message_reply_markup(
    inline_message_id=update.callback_query.inline_message_id, *args,␣
↪**kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.edit_message_reply_markup()* and *telegram.Message.edit_reply_markup()*.

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

Return type
telegram.Message

async edit_message_text(*text*, *parse_mode=None*, *disable_web_page_preview=None*, *reply_markup=None*, *entities=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for either:

```
await update.callback_query.message.edit_text(*args, **kwargs)
```

or:

```
await bot.edit_message_text(
    inline_message_id=update.callback_query.inline_message_id, *args,␣
↪**kwargs,
)
```

For the documentation of the arguments, please see *telegram.Bot.edit_message_text()* and *telegram.Message.edit_text()*.

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

Return type
telegram.Message

async get_game_high_scores(*user_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for either:

```
await update.callback_query.message.get_game_high_score(*args, **kwargs)
```

or:

```
await bot.get_game_high_scores(
    inline_message_id=update.callback_query.inline_message_id, *args,␣
↪**kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.get_game_high_scores()* and *telegram.Message.get_game_high_scores()*.

Returns
List[*telegram.GameHighScore*]

async pin_message(*disable_notification=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await update.callback_query.message.pin(*args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Message.pin()*.

Returns
On success, `True` is returned.

Return type
`bool`

async set_game_score(*user_id*, *score*, *force=None*, *disable_edit_message=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for either:

```
await update.callback_query.message.set_game_score(*args, **kwargs)
```

or:

```
await bot.set_game_score(
    inline_message_id=update.callback_query.inline_message_id, *args,␣
↪**kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.set_game_score()* and *telegram.Message.set_game_score()*.

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

Return type
telegram.Message

async stop_message_live_location(*reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for either:

```
await update.callback_query.message.stop_live_location(*args, **kwargs)
```

or:

```
await bot.stop_message_live_location(
    inline_message_id=update.callback_query.inline_message_id, *args,␣
↪**kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.stop_message_live_location()* and *telegram.Message.stop_live_location()*.

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

Return type
telegram.Message

async unpin_message(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await update.callback_query.message.unpin(*args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Message.unpin()*.

Returns
On success, `True` is returned.

Return type
`bool`

telegram.Chat

class telegram.**Chat**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a chat.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *id* is equal.

Changed in version 20.0:

- Removed the deprecated methods `kick_member` and `get_members_count`.
- The following are now keyword-only arguments in Bot methods: `location`, `filename`, `contact`, `{read, write, connect, pool}_timeout`, `api_kwargs`. Use a named argument for those, and notice that some positional arguments changed position as a result.

Parameters

- ***id*** (`int`) – Unique identifier for this chat. This number 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.
- ***type*** (`str`) – Type of chat, can be either *PRIVATE*, *GROUP*, *SUPERGROUP* or *CHANNEL*.
- ***title*** (`str`, optional) – Title, for supergroups, channels and group chats.
- ***username*** (`str`, optional) – Username, for private chats, supergroups and channels if available.
- ***first_name*** (`str`, optional) – First name of the other party in a private chat.
- ***last_name*** (`str`, optional) – Last name of the other party in a private chat.

- ***photo*** (*telegram.ChatPhoto*, optional) – Chat photo. Returned only in *telegram.Bot.get_chat()*.
- ***bio*** (str, optional) – Bio of the other party in a private chat. Returned only in *telegram.Bot.get_chat()*.
- ***has_private_forwards*** (bool, 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 *telegram.Bot.get_chat()*.

  New in version 13.9.
- ***description*** (str, optional) – Description, for groups, supergroups and channel chats. Returned only in *telegram.Bot.get_chat()*.
- ***invite_link*** (str, optional) – Primary invite link, for groups, supergroups and channel. Returned only in *telegram.Bot.get_chat()*.
- ***pinned_message*** (*telegram.Message*, optional) – The most recent pinned message (by sending date). Returned only in *telegram.Bot.get_chat()*.
- ***permissions*** (*telegram.ChatPermissions*) – Optional. Default chat member permissions, for groups and supergroups. Returned only in *telegram.Bot.get_chat()*.
- ***slow_mode_delay*** (int, optional) – For supergroups, the minimum allowed delay between consecutive messages sent by each unprivileged user. Returned only in *telegram.Bot.get_chat()*.
- ***message_auto_delete_time*** (int, optional) – The time after which all messages sent to the chat will be automatically deleted; in seconds. Returned only in *telegram.Bot.get_chat()*.

  New in version 13.4.
- ***has_protected_content*** (bool, optional) – True, if messages from the chat can't be forwarded to other chats. Returned only in *telegram.Bot.get_chat()*.

  New in version 13.9.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- ***sticker_set_name*** (str, optional) – For supergroups, name of group sticker set. Returned only in *telegram.Bot.get_chat()*.
- ***can_set_sticker_set*** (bool, optional) – True, if the bot can change group the sticker set. Returned only in *telegram.Bot.get_chat()*.
- ***linked_chat_id*** (int, optional) – Unique identifier for the linked chat, i.e. the discussion group identifier for a channel and vice versa; for supergroups and channel chats. Returned only in *telegram.Bot.get_chat()*.
- ***location*** (*telegram.ChatLocation*, optional) – For supergroups, the location to which the supergroup is connected. Returned only in *telegram.Bot.get_chat()*.
- ***join_to_send_messages*** (bool, optional) – True, if users need to join the supergroup before they can send messages. Returned only in *telegram.Bot.get_chat()*.

  New in version 20.0.
- ***join_by_request*** (bool, optional) – True, if all users directly joining the supergroup need to be approved by supergroup administrators. Returned only in *telegram.Bot.get_chat()*.

  New in version 20.0.
- ***has_restricted_voice_and_video_messages*** (bool, optional) – True, if the privacy settings of the other party restrict sending voice and video note messages in the private chat. Returned only in *telegram.Bot.get_chat()*.

  New in version 20.0.

- ****kwargs*** (dict) – Arbitrary keyword arguments.

id

Unique identifier for this chat.

Type
int

type

Type of chat.

Type
str

title

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

Type
str

username

Optional. Username.

Type
str

first_name

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

Type
str

last_name

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

Type
str

photo

Optional. Chat photo.

Type
telegram.ChatPhoto

bio

Optional. Bio of the other party in a private chat. Returned only in *telegram.Bot.get_chat()*.

Type
str

has_private_forwards

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.

New in version 13.9.

Type
bool

description

Optional. Description, for groups, supergroups and channel chats.

Type
str

invite_link

Optional. Primary invite link, for groups, supergroups and channel. Returned only in *telegram.Bot.get_chat()*.

Type
str

pinned_message

Optional. The most recent pinned message (by sending date). Returned only in *telegram.Bot.get_chat()*.

Type
telegram.Message

permissions

Optional. Default chat member permissions, for groups and supergroups. Returned only in *telegram.Bot.get_chat()*.

Type
telegram.ChatPermissions

slow_mode_delay

Optional. For supergroups, the minimum allowed delay between consecutive messages sent by each unprivileged user. Returned only in *telegram.Bot.get_chat()*.

Type
int

message_auto_delete_time

Optional. The time after which all messages sent to the chat will be automatically deleted; in seconds. Returned only in *telegram.Bot.get_chat()*.

New in version 13.4.

Type
int

has_protected_content

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

New in version 13.9.

Type
bool

sticker_set_name

Optional. For supergroups, name of Group sticker set.

Type
str

can_set_sticker_set

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

Type
bool

linked_chat_id

Optional. Unique identifier for the linked chat, i.e. the discussion group identifier for a channel and vice versa; for supergroups and channel chats. Returned only in *telegram.Bot.get_chat()*.

Type
int

location

Optional. For supergroups, the location to which the supergroup is connected. Returned only in *telegram.Bot.get_chat()*.

Type
telegram.ChatLocation

join_to_send_messages

Optional. True, if users need to join the supergroup before they can send messages. Returned only in *telegram.Bot.get_chat()*.

New in version 20.0.

Type
bool

join_by_request

Optional. True, if all users directly joining the supergroup need to be approved by supergroup administrators. Returned only in *telegram.Bot.get_chat()*.

New in version 20.0.

Type
bool

has_restricted_voice_and_video_messages

Optional. True, if the privacy settings of the other party restrict sending voice and video note messages in the private chat. Returned only in *telegram.Bot.get_chat()*.

New in version 20.0.

Type
bool

CHANNEL = 'channel'

telegram.constants.ChatType.CHANNEL

GROUP = 'group'

telegram.constants.ChatType.GROUP

PRIVATE = 'private'

telegram.constants.ChatType.PRIVATE

SENDER = 'sender'

telegram.constants.ChatType.SENDER

New in version 13.5.

SUPERGROUP = 'supergroup'

telegram.constants.ChatType.SUPERGROUP

async approve_join_request(*user_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.approve_chat_join_request(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.approve_chat_join_request()*.

New in version 13.8.

Returns
On success, True is returned.

Return type
bool

async **ban_chat**(*chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.ban_chat_sender_chat(
    sender_chat_id=update.effective_chat.id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.ban_chat_sender_chat()*.

New in version 13.9.

Returns
On success, True is returned.

Return type
bool

async **ban_member**(*user_id*, *revoke_messages=None*, *until_date=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.ban_chat_member(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.ban_chat_member()*.

Returns
On success, True is returned.

Return type
bool

async **ban_sender_chat**(*sender_chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.ban_chat_sender_chat(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.ban_chat_sender_chat()*.

New in version 13.9.

Returns
On success, True is returned.

Return type
bool

async **copy_message**(*chat_id*, *message_id*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *disable_notification=None*, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *reply_markup=None*, *protect_content=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.copy_message(from_chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.copy_message()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async create_invite_link(*expire_date=None*, *member_limit=None*, *name=None*, *creates_join_request=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.create_chat_invite_link(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.create_chat_invite_link()*.

New in version 13.4.

Changed in version 13.8: Edited signature according to the changes of *telegram.Bot.create_chat_invite_link()*.

Returns
telegram.ChatInviteLink

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

async decline_join_request(*user_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.decline_chat_join_request(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.decline_chat_join_request()*.

New in version 13.8.

Returns
On success, `True` is returned.

Return type
`bool`

async delete_photo(***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.delete_chat_photo(
    chat_id=update.effective_chat.id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.delete_chat_photo()*.

New in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

async edit_invite_link(*invite_link*, *expire_date=None*, *member_limit=None*, *name=None*, *creates_join_request=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.edit_chat_invite_link(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.edit_chat_invite_link()*.

New in version 13.4.

Changed in version 13.8: Edited signature according to the changes of *telegram.Bot.edit_chat_invite_link()*.

Returns
telegram.ChatInviteLink

async export_invite_link(***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.export_chat_invite_link(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.export_chat_invite_link()*.

New in version 13.4.

Returns
New invite link on success.

Return type
str

async forward_from(*from_chat_id*, *message_id*, *disable_notification=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.forward_message(chat_id=update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.forward_message()*.

See also:

forward_to()

New in version 20.0.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async forward_to(*chat_id*, *message_id*, *disable_notification=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.forward_message(from_chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.forward_message()*.

See also:

forward_from()

New in version 20.0.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

property full_name

Convenience property. If *first_name* is not None gives, *first_name* followed by (if available) *last_name*.

Note: *full_name* will always be None, if the chat is a (super)group or channel.

New in version 13.2.

Type
str

async get_administrators(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.get_chat_administrators(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.get_chat_administrators()*.

Returns
A list of administrators in a chat. An Array of *telegram.ChatMember* objects that contains information about all chat administrators except other bots. If the chat is a group or a supergroup and no administrators were appointed, only the creator will be returned.

Return type
List[*telegram.ChatMember*]

async get_member(*user_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.get_chat_member(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.get_chat_member()*.

Returns
telegram.ChatMember

async get_member_count(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.get_chat_member_count(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.get_chat_member_count()*.

Returns
int

async get_menu_button(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.get_chat_menu_button(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.set_chat_menu_button()*.

Caution: Can only work, if the chat is a private chat.

See also:

set_menu_button()

New in version 20.0.

Returns
On success, the current menu button is returned.

Return type
telegram.MenuButton

async leave(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.leave_chat(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.leave_chat()*.

Returns
On success, `True` is returned.

Return type
`bool`

property link

Convenience property. If the chat has a *username*, returns a t.me link of the chat.

Type
`str`

async pin_message(*message_id*, *disable_notification=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.pin_chat_message(chat_id=update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.pin_chat_message()*.

Returns
On success, `True` is returned.

Return type
`bool`

async promote_member(*user_id*, *can_change_info=None*, *can_post_messages=None*, *can_edit_messages=None*, *can_delete_messages=None*, *can_invite_users=None*, *can_restrict_members=None*, *can_pin_messages=None*, *can_promote_members=None*, *is_anonymous=None*, *can_manage_chat=None*, *can_manage_video_chats=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.promote_chat_member(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.promote_chat_member()*.

New in version 13.2.

Changed in version 20.0: The argument `can_manage_voice_chats` was renamed to *can_manage_video_chats* in accordance to Bot API 6.0.

Returns
On success, `True` is returned.

Return type
`bool`

async restrict_member(*user_id*, *permissions*, *until_date=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.restrict_chat_member(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.restrict_chat_member()*.

New in version 13.2.

Returns
On success, `True` is returned.

Return type
`bool`

async revoke_invite_link(*invite_link*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.revoke_chat_invite_link(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.revoke_chat_invite_link()*.

New in version 13.4.

Returns
telegram.ChatInviteLink

async send_action(*action*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_chat_action*

async send_animation(*animation*, *duration=None*, *width=None*, *height=None*, *thumb=None*, *caption=None*, *parse_mode=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_animation(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_animation()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_audio(*audio*, *duration=None*, *performer=None*, *title=None*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *thumb=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_audio(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_audio()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_chat_action(*action*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_chat_action(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_chat_action()*.

Returns
On success, `True` is returned.

Return type
`bool`

async send_contact(*phone_number=None*, *first_name=None*, *last_name=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *vcard=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *contact=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_contact(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_contact()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_copy(*from_chat_id*, *message_id*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *disable_notification=None*, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *reply_markup=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.copy_message(chat_id=update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.copy_message()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_dice(*disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *emoji=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_dice(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_dice()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_document(*document*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *thumb=None*, *disable_content_type_detection=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_document(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_document()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_game(*game_short_name*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_game(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_game()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_invoice(*title*, *description*, *payload*, *provider_token*, *currency*, *prices*, *start_parameter=None*, *photo_url=None*, *photo_size=None*, *photo_width=None*, *photo_height=None*, *need_name=None*, *need_phone_number=None*, *need_email=None*, *need_shipping_address=None*, *is_flexible=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *provider_data=None*, *send_phone_number_to_provider=None*, *send_email_to_provider=None*, *allow_sending_without_reply=None*, *max_tip_amount=None*, *suggested_tip_amounts=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_invoice(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_invoice()*.

> **Warning:** As of API 5.2 *start_parameter* is an optional argument and therefore the order of the arguments had to be changed. Use keyword arguments to make sure that the arguments are passed correctly.

Changed in version 13.5: As of Bot API 5.2, the parameter *start_parameter* is optional.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_location(*latitude=None*, *longitude=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *live_period=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *location=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_location(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_location()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_media_group(*media*, *disable_notification=None*, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_media_group(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_media_group()*.

Returns
On success, instance representing the message posted.

Return type
List[*telegram.Message*]

async send_message(*text*, *parse_mode=None*, *disable_web_page_preview=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *entities=None*, *protect_content=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_message(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_message()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_photo(*photo*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, *, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_photo(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_photo()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_poll(*question*, *options*, *is_anonymous=None*, *type=None*, *allows_multiple_answers=None*, *correct_option_id=None*, *is_closed=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *explanation=None*, *explanation_parse_mode=None*, *open_period=None*, *close_date=None*, *allow_sending_without_reply=None*, *explanation_entities=None*, *protect_content=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_poll(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_poll()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async **send_sticker**(*sticker*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_sticker(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_sticker()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async **send_venue**(*latitude=None*, *longitude=None*, *title=None*, *address=None*, *foursquare_id=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *foursquare_type=None*, *google_place_id=None*, *google_place_type=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *venue=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_venue(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_venue()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async **send_video**(*video*, *duration=None*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *width=None*, *height=None*, *parse_mode=None*, *supports_streaming=None*, *thumb=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_video(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_video()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async **send_video_note**(*video_note*, *duration=None*, *length=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *thumb=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_video_note(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_video_note()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_voice(*voice*, *duration=None*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_voice(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_voice()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async set_administrator_custom_title(*user_id*, *custom_title*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.set_chat_administrator_custom_title(
    update.effective_chat.id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.set_chat_administrator_custom_title()*.

Returns
On success, `True` is returned.

Return type
`bool`

async set_description(*description=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.set_chat_description(
    chat_id=update.effective_chat.id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.set_chat_description()*.

New in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

async set_menu_button(*menu_button=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.set_chat_menu_button(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.set_chat_menu_button()*.

Caution: Can only work, if the chat is a private chat.

See also:

get_menu_button()

New in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

async set_permissions(*permissions*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.set_chat_permissions(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.set_chat_permissions()*.

Returns
On success, `True` is returned.

Return type
`bool`

async set_photo(*photo*, ***, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.set_chat_photo(
    chat_id=update.effective_chat.id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.set_chat_photo()*.

New in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

async set_title(*title*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.set_chat_title(
    chat_id=update.effective_chat.id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.set_chat_title()*.

New in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

async unban_chat(*chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.unban_chat_sender_chat(
    sender_chat_id=update.effective_chat.id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.unban_chat_sender_chat()*.

New in version 13.9.

Returns
On success, `True` is returned.

Return type
`bool`

async unban_member(*user_id*, *only_if_banned=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.unban_chat_member(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.unban_chat_member()*.

Returns
On success, `True` is returned.

Return type
`bool`

async unban_sender_chat(*sender_chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.unban_chat_sender_chat(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.unban_chat_sender_chat()*.

New in version 13.9.

Returns
On success, `True` is returned.

Return type
`bool`

async unpin_all_messages(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.unpin_all_chat_messages(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.unpin_all_chat_messages()*.

Returns
On success, True is returned.

Return type
bool

async unpin_message(*message_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.unpin_chat_message(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.unpin_chat_message()*.

Returns
On success, True is returned.

Return type
bool

telegram.ChatAdministratorRights

New in version 20.0.

class telegram.**ChatAdministratorRights**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

Represents the rights of an administrator in a chat.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *is_anonymous*, *can_manage_chat*, *can_delete_messages*, *can_manage_video_chats*, *can_restrict_members*, *can_promote_members*, *can_change_info*, *can_invite_users*, *can_post_messages*, *can_edit_messages*, *can_pin_messages* are equal.

New in version 20.0.

Parameters

- ***is_anonymous*** (bool) – True, if the user's presence in the chat is hidden.
- ***can_manage_chat*** (bool) – True, if the administrator can access the chat event log, chat statistics, message statistics in channels, see channel members, see anonymous administrators in supergroups and ignore slow mode. Implied by any other administrator privilege.
- ***can_delete_messages*** (bool) – True, if the administrator can delete messages of other users.
- ***can_manage_video_chats*** (bool) – True, if the administrator can manage video chats.
- ***can_restrict_members*** (bool) – True, if the administrator can restrict, ban or unban chat members.
- ***can_promote_members*** (bool) – True, if the administrator can add new administrators with a subset of their own privileges or demote administrators that he has promoted, directly or indirectly (promoted by administrators that were appointed by the user.)
- ***can_change_info*** (bool) – True, if the user is allowed to change the chat title ,photo and other settings.
- ***can_invite_users*** (bool) – True, if the user is allowed to invite new users to the chat.

- ***can_post_messages*** (bool, optional) – True, if the administrator can post messages in the channel; channels only.
- ***can_edit_messages*** (bool, optional) – True, if the administrator can edit messages of other users.
- ***can_pin_messages*** (bool, optional) – True, if the user is allowed to pin messages; groups and supergroups only.

is_anonymous

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

Type
bool

can_manage_chat

True, if the administrator can access the chat event log, chat statistics, message statistics in channels, see channel members, see anonymous administrators in supergroups and ignore slow mode. Implied by any other administrator privilege.

Type
bool

can_delete_messages

True, if the administrator can delete messages of other users.

Type
bool

can_manage_video_chats

True, if the administrator can manage video chats.

Type
bool

can_restrict_members

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

Type
bool

can_promote_members

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

Type
bool

can_change_info

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

Type
bool

can_invite_users

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

Type
bool

can_post_messages

Optional. True, if the administrator can post messages in the channel; channels only.

Type
bool

can_edit_messages

Optional. True, if the administrator can edit messages of other users.

Type
bool

can_pin_messages

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

Type
bool

classmethod all_rights()

This method returns the *ChatAdministratorRights* object with all attributes set to True. This is e.g. useful when changing the bot's default administrator rights with *telegram.Bot.set_my_default_administrator_rights()*.

New in version 20.0.

classmethod no_rights()

This method returns the *ChatAdministratorRights* object with all attributes set to False.

New in version 20.0.

telegram.ChatInviteLink

class telegram.**ChatInviteLink**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents an invite link for a chat.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *invite_link*, *creator*, *creates_join_request*, *is_primary* and *is_revoked* are equal.

New in version 13.4.

Changed in version 20.0:

- The argument & attribute *creates_join_request* is now required to comply with the Bot API.
- Comparing objects of this class now also takes *creates_join_request* into account.

Parameters

- ***invite_link*** (str) – The invite link.
- ***creator*** (*telegram.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.

  New in version 13.8.
- ***is_primary*** (bool) – True, if the link is primary.
- ***is_revoked*** (bool) – True, if the link is revoked.
- ***expire_date*** (datetime.datetime, optional) – Date when the link will expire or has been expired.
- ***member_limit*** (int, optional) – Maximum number of users that can be members of the chat simultaneously after joining the chat via this invite link; 1-*99999*.
- ***name*** (str, optional) – Invite link name. 0-*32* characters.

  New in version 13.8.

- *pending_join_request_count* (int, optional) – Number of pending join requests created using this link.

  New in version 13.8.

invite_link

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

Type
str

creator

Creator of the link.

Type
telegram.User

creates_join_request

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

New in version 13.8.

Type
bool

is_primary

True, if the link is primary.

Type
bool

is_revoked

True, if the link is revoked.

Type
bool

expire_date

Optional. Date when the link will expire or has been expired.

Type
datetime.datetime

member_limit

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

Type
int

name

Optional. Invite link name.

New in version 13.8.

Type
str

pending_join_request_count

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

New in version 13.8.

Type
int

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.ChatJoinRequest

class telegram.**ChatJoinRequest**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a join request sent to a chat.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *chat*, *from_user* and *date* are equal.

Note: Since Bot API 5.5, bots are allowed to contact users who sent a join request to a chat where the bot is an administrator with the *can_invite_users* administrator right – even if the user never interacted with the bot before.

New in version 13.8.

Parameters

- ***chat*** (*telegram.Chat*) – Chat to which the request was sent.
- ***from_user*** (*telegram.User*) – User that sent the join request.
- ***date*** (datetime.datetime) – Date the request was sent.
- ***bio*** (str, optional) – Bio of the user.
- ***invite_link*** (*telegram.ChatInviteLink*, optional) – Chat invite link that was used by the user to send the join request.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.

chat

Chat to which the request was sent.

Type
telegram.Chat

from_user

User that sent the join request.

Type
telegram.User

date

Date the request was sent.

Type
datetime.datetime

bio

Optional. Bio of the user.

Type
str

invite_link

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

Type
telegram.ChatInviteLink

async approve(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.approve_chat_join_request(
    chat_id=update.effective_chat.id, user_id=update.effective_user.id,␣
↪*args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.approve_chat_join_request()*.

Returns
On success, `True` is returned.

Return type
`bool`

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

async decline(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.decline_chat_join_request(
    chat_id=update.effective_chat.id, user_id=update.effective_user.id,␣
↪*args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.decline_chat_join_request()*.

Returns
On success, `True` is returned.

Return type
`bool`

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.ChatLocation

class telegram.**ChatLocation**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a location to which a chat is connected.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *location* is equal.

Parameters

- ***location*** (*telegram.Location*) – The location to which the supergroup is connected. Can't be a live location.

- ***address*** (str) – Location address; 1-64 characters, as defined by the chat owner
- ****kwargs*** (dict) – Arbitrary keyword arguments.

location

The location to which the supergroup is connected.

Type
telegram.Location

address

Location address, as defined by the chat owner

Type
str

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

telegram.ChatMember

class telegram.**ChatMember**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

Base class for Telegram ChatMember Objects. Currently, the following 6 types of chat members are supported:

- *telegram.ChatMemberOwner*
- *telegram.ChatMemberAdministrator*
- *telegram.ChatMemberMember*
- *telegram.ChatMemberRestricted*
- *telegram.ChatMemberLeft*
- *telegram.ChatMemberBanned*

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *user* and *status* are equal.

See also:

Chat Member Example

Changed in version 20.0:

- As of Bot API 5.3, *ChatMember* is nothing but the base class for the subclasses listed above and is no longer returned directly by *get_chat()*. Therefore, most of the arguments and attributes were removed and you should no longer use *ChatMember* directly.
- The constant ChatMember.CREATOR was replaced by *OWNER*
- The constant ChatMember.KICKED was replaced by *BANNED*

Parameters

- ***user*** (*telegram.User*) – Information about the user.
- ***status*** (str) – The member's status in the chat. Can be *ADMINISTRATOR*, *OWNER*, *BANNED*, *LEFT*, *MEMBER* or *RESTRICTED*.

user

Information about the user.

Type
telegram.User

status

The member's status in the chat.

Type

str

ADMINISTRATOR = 'administrator'

telegram.constants.ChatMemberStatus.ADMINISTRATOR

BANNED = 'kicked'

telegram.constants.ChatMemberStatus.BANNED

LEFT = 'left'

telegram.constants.ChatMemberStatus.LEFT

MEMBER = 'member'

telegram.constants.ChatMemberStatus.MEMBER

OWNER = 'creator'

telegram.constants.ChatMemberStatus.OWNER

RESTRICTED = 'restricted'

telegram.constants.ChatMemberStatus.RESTRICTED

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.ChatMemberAdministrator

class telegram.**ChatMemberAdministrator**(**args*, ***kwargs*)

Bases: *telegram.ChatMember*

Represents a chat member that has some additional privileges.

New in version 13.7.

Changed in version 20.0: Argument and attribute `can_manage_voice_chats` were renamed to *can_manage_video_chats* and *can_manage_video_chats* in accordance to Bot API 6.0.

Parameters

- ***user*** (*telegram.User*) – Information about the user.
- ***can_be_edited*** (bool) – True, if the bot is allowed to edit administrator privileges of that user.
- ***is_anonymous*** (bool) – True, if the user's presence in the chat is hidden.
- ***can_manage_chat*** (bool) – True, if the administrator can access the chat event log, chat statistics, message statistics in channels, see channel members, see anonymous administrators in supergroups and ignore slow mode. Implied by any other administrator privilege.
- ***can_delete_messages*** (bool) – True, if the administrator can delete messages of other users.
- ***can_manage_video_chats*** (bool) – True, if the administrator can manage video chats.

  New in version 20.0.

- ***can_restrict_members*** (bool) – True, if the administrator can restrict, ban or unban chat members.

- ***can_promote_members*** (bool) – True, if the administrator can add new administrators with a subset of his own privileges or demote administrators that he has promoted, directly or indirectly (promoted by administrators that were appointed by the user).
- ***can_change_info*** (bool) – True, if the user can change the chat title, photo and other settings.
- ***can_invite_users*** (bool) – True, if the user can invite new users to the chat.
- ***can_post_messages*** (bool, optional) – True, if the administrator can post in the channel, channels only.
- ***can_edit_messages*** (bool, optional) – True, if the administrator can edit messages of other users and can pin messages; channels only.
- ***can_pin_messages*** (bool, optional) – True, if the user is allowed to pin messages; groups and supergroups only.
- ***custom_title*** (str, optional) – Custom title for this user.

status

The member's status in the chat, always *'administrator'*.

Type
str

user

Information about the user.

Type
telegram.User

can_be_edited

True, if the bot is allowed to edit administrator privileges of that user.

Type
bool

is_anonymous

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

Type
bool

can_manage_chat

True, if the administrator can access the chat event log, chat statistics, message statistics in channels, see channel members, see anonymous administrators in supergroups and ignore slow mode. Implied by any other administrator privilege.

Type
bool

can_delete_messages

True, if the administrator can delete messages of other users.

Type
bool

can_manage_video_chats

True, if the administrator can manage video chats.

New in version 20.0.

Type
bool

can_restrict_members

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

Type
bool

can_promote_members

True, if the administrator can add new administrators with a subset of his own privileges or demote administrators that he has promoted, directly or indirectly (promoted by administrators that were appointed by the user).

Type
bool

can_change_info

True, if the user can change the chat title, photo and other settings.

Type
bool

can_invite_users

True, if the user can invite new users to the chat.

Type
bool

can_post_messages

Optional. True, if the administrator can post in the channel, channels only.

Type
bool

can_edit_messages

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

Type
bool

can_pin_messages

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

Type
bool

custom_title

Optional. Custom title for this user.

Type
str

telegram.ChatMemberBanned

class telegram.**ChatMemberBanned**(**args*, ***kwargs*)

Bases: *telegram.ChatMember*

Represents a chat member that was banned in the chat and can't return to the chat or view chat messages.

New in version 13.7.

Parameters

- ***user*** (*telegram.User*) – Information about the user.
- ***until_date*** (datetime.datetime) – Date when restrictions will be lifted for this user.

status
The member's status in the chat, always *'kicked'*.

Type
str

user
Information about the user.

Type
telegram.User

until_date
Date when restrictions will be lifted for this user.

Type
datetime.datetime

telegram.ChatMemberLeft

class telegram.**ChatMemberLeft**(**args*, ***kwargs*)
Bases: *telegram.ChatMember*

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

New in version 13.7.

Parameters
user (*telegram.User*) – Information about the user.

status
The member's status in the chat, always *'left'*.

Type
str

user
Information about the user.

Type
telegram.User

telegram.ChatMemberMember

class telegram.**ChatMemberMember**(**args*, ***kwargs*)
Bases: *telegram.ChatMember*

Represents a chat member that has no additional privileges or restrictions.

New in version 13.7.

Parameters
user (*telegram.User*) – Information about the user.

status
The member's status in the chat, always *'member'*.

Type
str

user
Information about the user.

Type
telegram.User

telegram.ChatMemberOwner

class telegram.**ChatMemberOwner**(**args*, ***kwargs*)

Bases: *telegram.ChatMember*

Represents a chat member that owns the chat and has all administrator privileges.

New in version 13.7.

Parameters

- ***user*** (*telegram.User*) – Information about the user.
- ***is_anonymous*** (bool) – True, if the user's presence in the chat is hidden.
- ***custom_title*** (str, optional) – Custom title for this user.

status

The member's status in the chat, always *'creator'*.

Type
str

user

Information about the user.

Type
telegram.User

is_anonymous

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

Type
bool

custom_title

Optional. Custom title for this user.

Type
str

telegram.ChatMemberRestricted

class telegram.**ChatMemberRestricted**(**args*, ***kwargs*)

Bases: *telegram.ChatMember*

Represents a chat member that is under certain restrictions in the chat. Supergroups only.

New in version 13.7.

Parameters

- ***user*** (*telegram.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_change_info*** (bool) – True, if the user can change the chat title, photo and other settings.
- ***can_invite_users*** (bool) – True, if the user can invite new users to the chat.
- ***can_pin_messages*** (bool) – True, if the user is allowed to pin messages; groups and supergroups only.
- ***can_send_messages*** (bool) – True, if the user is allowed to send text messages, contacts, locations and venues.

- ***can_send_media_messages*** (bool) – True, if the user is allowed to send audios, documents, photos, videos, video notes and 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.
- ***until_date*** (datetime.datetime) – Date when restrictions will be lifted for this user.

status

The member's status in the chat, always *'restricted'*.

Type
str

user

Information about the user.

Type
telegram.User

is_member

True, if the user is a member of the chat at the moment of the request.

Type
bool

can_change_info

True, if the user can change the chat title, photo and other settings.

Type
bool

can_invite_users

True, if the user can invite new users to the chat.

Type
bool

can_pin_messages

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

Type
bool

can_send_messages

True, if the user is allowed to send text messages, contacts, locations and venues.

Type
bool

can_send_media_messages

True, if the user is allowed to send audios, documents, photos, videos, video notes and voice notes.

Type
bool

can_send_polls

True, if the user is allowed to send polls.

Type
bool

can_send_other_messages

True, if the user is allowed to send animations, games, stickers and use inline bots.

Type
bool

can_add_web_page_previews

True, if the user is allowed to add web page previews to their messages.

Type
bool

until_date

Date when restrictions will be lifted for this user.

Type
datetime.datetime

telegram.ChatMemberUpdated

class telegram.**ChatMemberUpdated**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *chat*, *from_user*, *date*, *old_chat_member* and *new_chat_member* are equal.

New in version 13.4.

Note: In Python from is a reserved word use *from_user* instead.

Parameters

- ***chat*** (*telegram.Chat*) – Chat the user belongs to.
- ***from_user*** (*telegram.User*) – Performer of the action, which resulted in the change.
- ***date*** (datetime.datetime) – Date the change was done in Unix time. Converted to datetime.datetime.
- ***old_chat_member*** (*telegram.ChatMember*) – Previous information about the chat member.
- ***new_chat_member*** (*telegram.ChatMember*) – New information about the chat member.
- ***invite_link*** (*telegram.ChatInviteLink*, optional) – Chat invite link, which was used by the user to join the chat. For joining by invite link events only.

chat

Chat the user belongs to.

Type
telegram.Chat

from_user

Performer of the action, which resulted in the change.

Type
telegram.User

date

Date the change was done in Unix time. Converted to datetime.datetime.

Type
datetime.datetime

old_chat_member

Previous information about the chat member.

Type
telegram.ChatMember

new_chat_member

New information about the chat member.

Type
telegram.ChatMember

invite_link

Optional. Chat invite link, which was used by the user to join the chat.

Type
telegram.ChatInviteLink

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

difference()

Computes the difference between *old_chat_member* and *new_chat_member*.

Example

```
>>> chat_member_updated.difference()
{'custom_title': ('old title', 'new title')}
```

Note: To determine, if the *telegram.ChatMember.user* attribute has changed, *every* attribute of the user will be checked.

New in version 13.5.

Returns
A dictionary mapping attribute names to tuples of the form `(old_value, new_value)`

Return type
Dict[str, Tuple[object, object]]

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.ChatPermissions

class telegram.**ChatPermissions**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *can_send_messages*, *can_send_media_messages*, *can_send_polls*, *can_send_other_messages*, *can_add_web_page_previews*, *can_change_info*, *can_invite_users* and *can_pin_messages* are equal.

Note: Though not stated explicitly in the official docs, Telegram changes not only the permissions that are set, but also sets all the others to `False`. However, since not documented, this behaviour may change unbeknown to PTB.

Parameters

- ***can_send_messages*** (`bool`, optional) – `True`, if the user is allowed to send text messages, contacts, locations and venues.
- ***can_send_media_messages*** (`bool`, optional) – `True`, if the user is allowed to send audios, documents, photos, videos, video notes and voice notes, implies *can_send_messages*.
- ***can_send_polls*** (`bool`, optional) – `True`, if the user is allowed to send polls, implies *can_send_messages*.
- ***can_send_other_messages*** (`bool`, optional) – `True`, if the user is allowed to send animations, games, stickers and use inline bots, implies *can_send_media_messages*.
- ***can_add_web_page_previews*** (`bool`, optional) – `True`, if the user is allowed to add web page previews to their messages, implies *can_send_media_messages*.
- ***can_change_info*** (`bool`, optional) – `True`, if the user is allowed to change the chat title, photo and other settings. Ignored in public supergroups.
- ***can_invite_users*** (`bool`, optional) – `True`, if the user is allowed to invite new users to the chat.
- ***can_pin_messages*** (`bool`, optional) – `True`, if the user is allowed to pin messages. Ignored in public supergroups.

can_send_messages

Optional. `True`, if the user is allowed to send text messages, contacts, locations and venues.

Type
`bool`

can_send_media_messages

Optional. `True`, if the user is allowed to send audios, documents, photos, videos, video notes and voice notes, implies *can_send_messages*.

Type
`bool`

can_send_polls

Optional. `True`, if the user is allowed to send polls, implies *can_send_messages*.

Type
`bool`

can_send_other_messages

Optional. True, if the user is allowed to send animations, games, stickers and use inline bots, implies *can_send_media_messages*.

Type
bool

can_add_web_page_previews

Optional. True, if the user is allowed to add web page previews to their messages, implies *can_send_media_messages*.

Type
bool

can_change_info

Optional. True, if the user is allowed to change the chat title, photo and other settings. Ignored in public supergroups.

Type
bool

can_invite_users

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

Type
bool

can_pin_messages

Optional. True, if the user is allowed to pin messages. Ignored in public supergroups.

Type
bool

classmethod all_permissions()

This method returns an *ChatPermissions* instance with all attributes set to True. This is e.g. useful when unrestricting a chat member with *telegram.Bot.restrict_chat_member()*.

New in version 20.0.

classmethod no_permissions()

This method returns an *ChatPermissions* instance with all attributes set to False.

New in version 20.0.

telegram.ChatPhoto

class telegram.**ChatPhoto**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a chat photo.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *small_file_unique_id* and *big_file_unique_id* are equal.

Parameters

- ***small_file_id*** (str) – Unique file identifier of small (160x160) chat photo. This file_id can be used only for photo download and only for as long as the photo is not changed.
- ***small_file_unique_id*** (str) – Unique file identifier of small (160x160) chat photo, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.

- ***big_file_id*** (str) – Unique 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.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods
- *****kwargs*** (dict) – Arbitrary keyword arguments.

small_file_id

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.

Type
str

small_file_unique_id

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.

Type
str

big_file_id

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.

Type
str

big_file_unique_id

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.

Type
str

async get_big_file(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Convenience wrapper over *telegram.Bot.get_file* for getting the big (640x640) chat photo

For the documentation of the arguments, please see *telegram.Bot.get_file()*.

Returns
telegram.File

Raises
telegram.error.TelegramError –

async get_small_file(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Convenience wrapper over *telegram.Bot.get_file* for getting the small (160x160) chat photo

For the documentation of the arguments, please see *telegram.Bot.get_file()*.

Returns
telegram.File

Raises
telegram.error.TelegramError –

telegram.Contact

class telegram.**Contact**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a phone contact.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *phone_number* is equal.

Parameters

- ***phone_number*** (str) – Contact's phone number.
- ***first_name*** (str) – Contact's first name.
- ***last_name*** (str, optional) – Contact's last name.
- ***user_id*** (int, optional) – Contact's user identifier in Telegram.
- ***vcard*** (str, optional) – Additional data about the contact in the form of a vCard.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

phone_number

Contact's phone number.

Type
str

first_name

Contact's first name.

Type
str

last_name

Optional. Contact's last name.

Type
str

user_id

Optional. Contact's user identifier in Telegram.

Type
int

vcard

Optional. Additional data about the contact in the form of a vCard.

Type
str

telegram.Dice

class telegram.**Dice**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents an animated emoji with a random value for currently supported base emoji. (The singular form of "dice" is "die". However, PTB mimics the Telegram API, which uses the term "dice".)

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *value* and *emoji* are equal.

Note: If *emoji* is “”, a value of 6 currently represents a bullseye, while a value of 1 indicates that the dartboard was missed. However, this behaviour is undocumented and might be changed by Telegram.

If *emoji* is “”, a value of 4 or 5 currently score a basket, while a value of 1 to 3 indicates that the basket was missed. However, this behaviour is undocumented and might be changed by Telegram.

If *emoji* is “”, a value of 4 to 5 currently scores a goal, while a value of 1 to 3 indicates that the goal was missed. However, this behaviour is undocumented and might be changed by Telegram.

If *emoji* is “”, a value of 6 knocks all the pins, while a value of 1 means all the pins were missed. However, this behaviour is undocumented and might be changed by Telegram.

If *emoji* is “”, each value corresponds to a unique combination of symbols, which can be found at our wiki. However, this behaviour is undocumented and might be changed by Telegram.

Parameters

- ***value*** (int) – Value of the dice. 1-6 for dice, darts and bowling balls, 1-5 for basketball and football/soccer ball, 1-64 for slot machine.
- ***emoji*** (str) – Emoji on which the dice throw animation is based.

value

Value of the dice.

Type
int

emoji

Emoji on which the dice throw animation is based.

Type
str

ALL_EMOJI = [<DiceEmoji.DICE>, <DiceEmoji.DARTS>, <DiceEmoji.BASKETBALL>, <DiceEmoji.FOOTBALL>, <DiceEmoji.SLOT_MACHINE>, <DiceEmoji.BOWLING>]

A list of all available dice emoji.

Type
List[str]

BASKETBALL = ''

telegram.constants.DiceEmoji.BASKETBALL

BOWLING = ''

telegram.constants.DiceEmoji.BOWLING

New in version 13.4.

DARTS = ''

telegram.constants.DiceEmoji.DARTS

DICE = ''

telegram.constants.DiceEmoji.DICE

FOOTBALL = ''

telegram.constants.DiceEmoji.FOOTBALL

SLOT_MACHINE = ''

telegram.constants.DiceEmoji.SLOT_MACHINE

telegram.Document

class telegram.**Document**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *file_unique_id* is equal.

Parameters

- ***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.
- ***thumb*** (*telegram.PhotoSize*, optional) – Document thumbnail as defined by sender.
- ***file_name*** (str, optional) – Original filename as defined by sender.
- ***mime_type*** (str, optional) – MIME type of the file as defined by sender.
- ***file_size*** (int, optional) – File size in bytes.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

file_id

File identifier.

Type
str

file_unique_id

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

thumb

Optional. Document thumbnail.

Type
telegram.PhotoSize

file_name

Original filename.

Type
str

mime_type

Optional. MIME type of the file.

Type
str

file_size

Optional. File size in bytes.

Type
int

bot

Optional. The Bot to use for instance methods.

Type
telegram.Bot

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

async get_file(**, read_timeout=None, write_timeout=None, connect_timeout=None, pool_timeout=None, api_kwargs=None*)

Convenience wrapper over *telegram.Bot.get_file*

For the documentation of the arguments, please see *telegram.Bot.get_file()*.

Returns
telegram.File

Raises
telegram.error.TelegramError –

telegram.File

class telegram.**File**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a file ready to be downloaded. The file can be downloaded with *download*. 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 *telegram.Bot.get_file()*.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *file_unique_id* is equal.

Note:

- Maximum file size to download is *20 MB*.
- If you obtain an instance of this class from *telegram.PassportFile.get_file*, then it will automatically be decrypted as it downloads when you call *download()*.

Parameters

- ***file_id*** (str) – Identifier for this file, which can be used to download or reuse the file.
- ***file_unique_id*** (str) – Unique identifier for this file, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.
- ***file_size*** (int, optional) – Optional. File size in bytes, if known.
- ***file_path*** (str, optional) – File path. Use *download* to get the file.
- ***bot*** (*telegram.Bot*, optional) – Bot to use with shortcut method.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

file_id

Identifier for this file.

Type
str

file_unique_id

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

file_size

Optional. File size in bytes.

Type
str

file_path

Optional. File path. Use *download()* to get the file.

Type
str

async download(*custom_path=None*, *out=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*)

Download this file. By default, the file is saved in the current working directory with its original filename as reported by Telegram. If the file has no filename, it the file ID will be used as filename. If a *custom_path* is supplied, it will be saved to that path instead. If *out* is defined, the file contents will be saved to that object using the `out.write` method.

Note:

- *custom_path* and *out* are mutually exclusive.
- If neither *custom_path* nor *out* is provided and *file_path* is the path of a local file (which is the case when a Bot API Server is running in local mode), this method will just return the path.

Changed in version 20.0:

- *custom_path* parameter now also accepts `pathlib.Path` as argument.
- Returns `pathlib.Path` object in cases where previously a `str` was returned.

Parameters

- ***custom_path*** (`pathlib.Path` | `str`, optional) – Custom path.
- ***out*** (`io.BufferedWriter`, optional) – A file-like object. Must be opened for writing in binary mode, if applicable.
- ***read_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.pool_timeout*. Defaults to *DEFAULT_NONE*.

Returns
The same object as *out* if specified. Otherwise, returns the filename downloaded to or the file path of the local file.

Return type
`pathlib.Path` | `io.BufferedWriter`

Raises
ValueError – If both *custom_path* and *out* are passed.

async download_as_bytearray(*buf=None*)

Download this file and return it as a bytearray.

Parameters
buf (bytearray, optional) – Extend the given bytearray with the downloaded data.

Returns
The same object as *buf* if it was specified. Otherwise a newly allocated bytearray.

Return type
bytearray

set_credentials(*credentials*)

Sets the passport credentials for the file.

Parameters
credentials (*telegram.FileCredentials*) – The credentials.

telegram.ForceReply

class telegram.**ForceReply**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

Upon receiving a message with this object, Telegram clients will display a reply interface to the user (act as if the user has selected the bot's message and tapped 'Reply'). This can be extremely useful if you want to create user-friendly step-by-step interfaces without having to sacrifice privacy mode.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *selective* is equal.

Changed in version 20.0: The (undocumented) argument `force_reply` was removed and instead *force_reply* is now always set to True as expected by the Bot API.

Parameters

- ***selective*** (bool, 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 *telegram.Message* object.
  2) If the bot's message is a reply (has `reply_to_message_id`), sender of the original message.
- ***input_field_placeholder*** (str, optional) – The placeholder to be shown in the input field when the reply is active; 1-64 characters.

  New in version 13.7.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

force_reply

Shows reply interface to the user, as if they manually selected the bots message and tapped 'Reply'.

Type
True

selective

Optional. Force reply from specific users only.

Type
bool

input_field_placeholder

Optional. The placeholder shown in the input field when the reply is active.

New in version 13.7.

Type
str

telegram.InlineKeyboardButton

class telegram.**InlineKeyboardButton**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents one button of an inline keyboard.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *text*, *url*, *login_url*, *callback_data*, *switch_inline_query*, *switch_inline_query_current_chat*, *callback_game*, *web_app* and *pay* are equal.

Note:

- You must use exactly one of the optional fields. Mind that *callback_game* is not working as expected. Putting a game short name in it might, but is not guaranteed to work.
- If your bot allows for arbitrary callback data, in keyboards returned in a response from telegram, *callback_data* maybe be an instance of *telegram.ext.InvalidCallbackData*. This will be the case, if the data associated with the button was already deleted.

  New in version 13.6.
- Since Bot API 5.5, it's now allowed to mention users by their ID in inline keyboards. This will only work in Telegram versions released after December 7, 2021. Older clients will display *unsupported message*.

Warning:

- If your bot allows your arbitrary callback data, buttons whose callback data is a non-hashable object will become unhashable. Trying to evaluate `hash(button)` will result in a `TypeError`.

  Changed in version 13.6.
- After Bot API 6.1, only `HTTPS` links will be allowed in *login_url*.

See also:

Inline Keyboard Example 1, Inline Keyboard Example 2, *telegram.InlineKeyboardMarkup*

Changed in version 20.0: *web_app* is considered as well when comparing objects of this type in terms of equality.

Parameters

- ***text*** (str) – Label text on the button.
- ***url*** (str, optional) – HTTP or tg:// url to be opened when the button is pressed. Links `tg://user?id=<user_id>` can be used to mention a user by their ID without using a username, if this is allowed by their privacy settings.

  Changed in version 13.9: You can now mention a user using `tg://user?id=<user_id>`.

- ***login_url*** (*telegram.LoginUrl*, optional) – An HTTPS URL used to automatically authorize the user. Can be used as a replacement for the Telegram Login Widget.

  Caution: Only HTTPS links are allowed after Bot API 6.1.

- ***callback_data*** (str | object, optional) – Data to be sent in a callback query to the bot when button is pressed, UTF-8 1-64 bytes. If the bot instance allows arbitrary callback data, anything can be passed.

  Tip: The value entered here will be available in *telegram.CallbackQuery.data*.

- ***web_app*** (*telegram.WebAppInfo*, 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 *answer_web_app_query()*. Available only in private chats between a user and the bot.

  New in version 20.0.

- ***switch_inline_query*** (str, 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. Can be empty, in which case just the bot's username will be inserted. This offers an easy way for users to start using your bot in inline mode when they are currently in a private chat with it. Especially useful when combined with switch_pm* actions - in this case the user will be automatically returned to the chat they switched from, skipping the chat selection screen.
- ***switch_inline_query_current_chat*** (str, optional) – If set, pressing the button will insert the bot's username and the specified inline query in the current chat's input field. Can be empty, in which case only the bot's username will be inserted. This offers a quick way for the user to open your bot in inline mode in the same chat - good for selecting something from multiple options.
- ***callback_game*** (*telegram.CallbackGame*, optional) – Description of the game that will be launched when the user presses the button. This type of button must always be the `first` button in the first row.
- ***pay*** (bool, optional) – Specify True, to send a Pay button. This type of button must always be the *first* button in the first row and can only be used in invoice messages.
- ****kwargs*** (dict) – Arbitrary keyword arguments.

text

Label text on the button.

Type
str

url

Optional. HTTP or tg:// url to be opened when the button is pressed. Links `tg://user?id=<user_id>` can be used to mention a user by their ID without using a username, if this is allowed by their privacy settings.

Changed in version 13.9: You can now mention a user using `tg://user?id=<user_id>`.

Type
str

login_url

Optional. An HTTPS URL used to automatically authorize the user. Can be used as a replacement for the Telegram Login Widget.

> **Caution:** Only HTTPS links are allowed after Bot API 6.1.

Type
telegram.LoginUrl

callback_data

Optional. Data to be sent in a callback query to the bot when button is pressed, UTF-8 1-64 bytes.

Type
str|object

web_app

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 *answer_web_app_query()*. Available only in private chats between a user and the bot.

New in version 20.0.

Type
telegram.WebAppInfo

switch_inline_query

Optional. 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. Can be empty, in which case just the bot's username will be inserted.

Type
str

switch_inline_query_current_chat

Optional. Will insert the bot's username and the specified inline query in the current chat's input field. Can be empty, in which case just the bot's username will be inserted.

Type
str

callback_game

Optional. Description of the game that will be launched when the user presses the button.

Type
telegram.CallbackGame

pay

Optional. Specify True, to send a Pay button.

Type
bool

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

update_callback_data(*callback_data*)

Sets *callback_data* to the passed object. Intended to be used by *telegram.ext.CallbackDataCache*.

New in version 13.6.

Parameters
callback_data (object) – The new callback data.

telegram.InlineKeyboardMarkup

class telegram.**InlineKeyboardMarkup**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents an inline keyboard that appears right next to the message it belongs to.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their size of *inline_keyboard* and all the buttons are equal.

See also:

Inline Keyboard Example 1, Inline Keyboard Example 2

Parameters

- ***inline_keyboard*** (List[List[*telegram.InlineKeyboardButton*]]) – List of button rows, each represented by a list of InlineKeyboardButton objects.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

inline_keyboard

List of button rows, each represented by a list of InlineKeyboardButton objects.

Type
List[List[*telegram.InlineKeyboardButton*]]

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

classmethod from_button(*button*, ***kwargs*)

Shortcut for:

```
InlineKeyboardMarkup([[button]], **kwargs)
```

Return an InlineKeyboardMarkup from a single InlineKeyboardButton

Parameters

- ***button*** (*telegram.InlineKeyboardButton*) – The button to use in the markup
- *****kwargs*** (dict) – Arbitrary keyword arguments.

classmethod from_column(*button_column*, ***kwargs*)

Shortcut for:

```
InlineKeyboardMarkup([[button] for button in button_column], **kwargs)
```

Return an InlineKeyboardMarkup from a single column of InlineKeyboardButtons

Parameters

- ***button_column*** (List[*telegram.InlineKeyboardButton*]) – The button to use in the markup
- *****kwargs*** (dict) – Arbitrary keyword arguments.

classmethod from_row(*button_row*, ***kwargs*)

Shortcut for:

```
InlineKeyboardMarkup([button_row], **kwargs)
```

Return an InlineKeyboardMarkup from a single row of InlineKeyboardButtons

Parameters

- ***button_row*** (List[*telegram.InlineKeyboardButton*]) – The button to use in the markup
- *****kwargs*** (dict) – Arbitrary keyword arguments.

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.InputFile

class telegram.**InputFile**(*obj*, *filename=None*, *attach=False*)

Bases: object

This object represents a Telegram InputFile.

Changed in version 20.0:

- The former attribute `attach` was renamed to *attach_name*.
- Method `is_image` was removed. If you pass `bytes` to *obj* and would like to have the mime type automatically guessed, please pass *filename* in addition.

Parameters

- ***obj*** (file object | `bytes` | `str`) – An open file descriptor or the files content as bytes or string.

  Note: If *obj* is a string, it will be encoded as bytes via `obj.encode('utf-8')`.

  Changed in version 20.0: Accept string input.
- ***filename*** (`str`, optional) – Filename for this InputFile.
- ***attach*** (`bool`, optional) – Pass `True` if the parameter this file belongs to in the request to Telegram should point to the multipart data via an `attach://` URI. Defaults to *False*.

input_file_content

The binary content of the file to send.

Type
bytes

attach_name

Optional. If present, the parameter this file belongs to in the request to Telegram should point to the multipart data via a an URI of the form `attach://<attach_name>` URI.

Type
str

filename

Filename for the file to be sent.

Type
str

mimetype

The mimetype inferred from the file to be sent.

Type
str

property attach_uri

URI to insert into the JSON data for uploading the file. Returns `None`, if *attach_name* is `None`.

property field_tuple

Field tuple representing the contents of the file for upload to the Telegram servers.

Return type
Tuple[str, bytes, str]

telegram.InputMedia

class telegram.**InputMedia**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

Base class for Telegram InputMedia Objects.

Changed in version 20.0:: Added arguments and attributes *type*, *media*, *caption*, *caption_entities*, *parse_mode*.

Parameters

- ***media_type*** (str) – Type of media that the instance represents.
- ***media*** (str | file object | bytes | pathlib.Path | *telegram.Animation* | *telegram.Audio* | *telegram.Document* | *telegram.PhotoSize* | *telegram.Video*) – 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. Lastly you can pass an existing telegram media object of the corresponding type to send.
- ***caption*** (str, optional) – Caption of the media to be sent, 0-*1024* characters after entities parsing.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.

type

Type of the input media.

Type
str

media

Media to send.

Type
str | *telegram.InputFile*

caption

Optional. Caption of the media to be sent.

Type
str

parse_mode

Optional. The parse mode to use for text formatting.

Type
str

caption_entities

Optional. List of special entities that appear in the caption.

Type
List[*telegram.MessageEntity*]

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.InputMediaAnimation

class telegram.**InputMediaAnimation**(**args*, ***kwargs*)

Bases: *telegram.InputMedia*

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

Note: When using a *telegram.Animation* for the *media* attribute, it will take the width, height and duration from that video, unless otherwise specified with the optional arguments.

Parameters

- ***media*** (str | file object | bytes | pathlib.Path | *telegram.Animation*) – 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. Lastly you can pass an existing *telegram.Animation* object to send.

  Changed in version 13.2: Accept bytes as input.

- ***filename*** (str, optional) – Custom file name for the animation, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the tempfile module.

  New in version 13.1.

- ***thumb*** (file object | bytes | pathlib.Path, 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.

  Changed in version 13.2: Accept bytes as input.

- ***caption*** (str, optional) – Caption of the animation to be sent, 0-*1024* characters after entities parsing.

- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.

- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.

- ***width*** (int, optional) – Animation width.

- ***height*** (int, optional) – Animation height.

- ***duration*** (int, optional) – Animation duration in seconds.

type

'animation'.

Type

str

media

Animation to send.

Type

str | *telegram.InputFile*

caption

Optional. Caption of the document to be sent.

Type
str

parse_mode

Optional. The parse mode to use for text formatting.

Type
str

caption_entities

Optional. List of special entities that appear in the caption.

Type
List[*telegram.MessageEntity*]

thumb

Optional. Thumbnail of the file to send.

Type
telegram.InputFile

width

Optional. Animation width.

Type
int

height

Optional. Animation height.

Type
int

duration

Optional. Animation duration in seconds.

Type
int

telegram.InputMediaAudio

class telegram.**InputMediaAudio**(**args*, ***kwargs*)

Bases: *telegram.InputMedia*

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

Note: When using a *telegram.Audio* for the *media* attribute, it will take the duration, performer and title from that video, unless otherwise specified with the optional arguments.

Parameters

- ***media*** (str | file object | bytes | pathlib.Path | *telegram.Audio*) – 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. Lastly you can pass an existing *telegram.Audio* object to send.

  Changed in version 13.2: Accept bytes as input.

- ***filename*** (str, optional) – Custom file name for the audio, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the tempfile module.

  New in version 13.1.

- ***caption*** (str, optional) – Caption of the audio to be sent, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- ***duration*** (int) – Duration of the audio in seconds as defined by sender.
- ***performer*** (str, optional) – Performer of the audio as defined by sender or by audio tags.
- ***title*** (str, optional) – Title of the audio as defined by sender or by audio tags.
- ***thumb*** (file object | bytes | pathlib.Path, 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.

  Changed in version 13.2: Accept bytes as input.

type

'audio'.

Type
str

media

Audio file to send.

Type
str | *telegram.InputFile*

caption

Optional. Caption of the document to be sent.

Type
str

parse_mode

Optional. The parse mode to use for text formatting.

Type
str

caption_entities

Optional. List of special entities that appear in the caption.

Type
List[*telegram.MessageEntity*]

duration

Duration of the audio in seconds.

Type
int

performer

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

Type
str

title

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

Type
str

thumb

Optional. Thumbnail of the file to send.

Type
telegram.InputFile

telegram.InputMediaDocument

class telegram.**InputMediaDocument**(**args*, ***kwargs*)

Bases: *telegram.InputMedia*

Represents a general file to be sent.

Parameters

- ***media*** (str | file object | bytes | pathlib.Path | *telegram.Document*) – 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. Lastly you can pass an existing *telegram.Document* object to send.

  Changed in version 13.2: Accept bytes as input.

- ***filename*** (str, optional) – Custom file name for the document, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the tempfile module.

  New in version 13.1.

- ***caption*** (str, optional) – Caption of the document to be sent, 0-*1024* characters after entities parsing.

- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.

- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.

- ***thumb*** (file object | bytes | pathlib.Path, 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.

  Changed in version 13.2: Accept bytes as input.

- ***disable_content_type_detection*** (bool, 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.

type

'document'.

Type
str

media

File to send.

Type
str | *telegram.InputFile*

caption

Optional. Caption of the document to be sent.

Type
str

parse_mode

Optional. The parse mode to use for text formatting.

Type
str

caption_entities

Optional. List of special entities that appear in the caption.

Type
List[*telegram.MessageEntity*]

thumb

Optional. Thumbnail of the file to send.

Type
telegram.InputFile

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.

Type
bool

telegram.InputMediaPhoto

class telegram.**InputMediaPhoto**(**args*, ***kwargs*)

Bases: *telegram.InputMedia*

Represents a photo to be sent.

Parameters

- ***media*** (str | file object | bytes | pathlib.Path | *telegram.PhotoSize*) – 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. Lastly you can pass an existing *telegram.PhotoSize* object to send.

  Changed in version 13.2: Accept bytes as input.

- ***filename*** (str, optional) – Custom file name for the photo, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the tempfile module.

  New in version 13.1.

- ***caption*** (str, optional) – Caption of the photo to be sent, 0-*1024* characters after entities parsing.

- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.

type

'photo'.

Type
str

media

Photo to send.

Type
str | *telegram.InputFile*

caption

Optional. Caption of the document to be sent.

Type
str

parse_mode

Optional. The parse mode to use for text formatting.

Type
str

caption_entities

Optional. List of special entities that appear in the caption.

Type
List[*telegram.MessageEntity*]

telegram.InputMediaVideo

class telegram.**InputMediaVideo**(**args*, ***kwargs*)

Bases: *telegram.InputMedia*

Represents a video to be sent.

Note:

- When using a *telegram.Video* for the *media* attribute, it will take the width, height and duration from that video, unless otherwise specified with the optional arguments.
- *thumb* will be ignored for small video files, for which Telegram can easily generate thumbnails. However, this behaviour is undocumented and might be changed by Telegram.

Parameters

- ***media*** (str | file object | bytes | pathlib.Path | *telegram.Video*) – 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. Lastly you can pass an existing *telegram.Video* object to send.

  Changed in version 13.2: Accept bytes as input.

- ***filename*** (str, optional) – Custom file name for the video, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the tempfile module.

  New in version 13.1.

- ***caption*** (str, optional) – Caption of the video to be sent, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- ***width*** (int, optional) – Video width.
- ***height*** (int, optional) – Video height.
- ***duration*** (int, optional) – Video duration in seconds.
- ***supports_streaming*** (bool, optional) – Pass True, if the uploaded video is suitable for streaming.
- ***thumb*** (file object | bytes | pathlib.Path, 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.

  Changed in version 13.2: Accept bytes as input.

type

'video'.

Type
str

media

Video file to send.

Type
str | *telegram.InputFile*

caption

Optional. Caption of the document to be sent.

Type
str

parse_mode

Optional. The parse mode to use for text formatting.

Type
str

caption_entities

Optional. List of special entities that appear in the caption.

Type
List[*telegram.MessageEntity*]

width

Optional. Video width.

Type
int

height

Optional. Video height.

Type
int

duration

Optional. Video duration in seconds.

Type
int

supports_streaming

Optional. Pass True, if the uploaded video is suitable for streaming.

Type
bool

thumb

Optional. Thumbnail of the file to send.

Type
telegram.InputFile

telegram.KeyboardButton

class telegram.**KeyboardButton**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents one button of the reply keyboard. For simple text buttons String can be used instead of this object to specify text of the button.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *text*, *request_contact*, *request_location*, *request_poll* and *web_app* are equal.

Note:

- Optional fields are mutually exclusive.
- *request_contact* and *request_location* options will only work in Telegram versions released after 9 April, 2016. Older clients will display unsupported message.
- *request_poll* option will only work in Telegram versions released after 23 January, 2020. Older clients will display unsupported message.
- *web_app* option will only work in Telegram versions released after 16 April, 2022. Older clients will display unsupported message.

Changed in version 20.0: *web_app* is considered as well when comparing objects of this type in terms of equality.

Parameters

- ***text*** (str) – Text of the button. If none of the optional fields are used, it will be sent to the bot as a message when the button is pressed.
- ***request_contact*** (bool, optional) – If True, the user's phone number will be sent as a contact when the button is pressed. Available in private chats only.
- ***request_location*** (bool, optional) – If True, the user's current location will be sent when the button is pressed. Available in private chats only.

- ***request_poll*** (*KeyboardButtonPollType*, 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*, optional) – If specified, the described Web App will be launched when the button is pressed. The Web App will be able to send a *Message.web_app_data* service message. Available in private chats only.

  New in version 20.0.

text

Text of the button.

Type
str

request_contact

Optional. The user's phone number will be sent.

Type
bool

request_location

Optional. The user's current location will be sent.

Type
bool

request_poll

Optional. If the user should create a poll.

Type
KeyboardButtonPollType

web_app

Optional. If the described Web App will be launched when the button is pressed.

New in version 20.0.

Type
WebAppInfo

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

telegram.KeyboardButtonPollType

class telegram.**KeyboardButtonPollType**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *type* is equal.

See also:

Pollbot Example

type

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.

Type
str

telegram.Location

class telegram.**Location**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a point on the map.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *longitude* and *latitude* are equal.

Parameters

- ***longitude*** (float) – Longitude as defined by sender.
- ***latitude*** (float) – Latitude as defined by sender.
- ***horizontal_accuracy*** (float, optional) – The radius of uncertainty for the location, measured in meters; 0-*1500*.
- ***live_period*** (int, optional) – Time relative to the message sending date, during which the location can be updated, in seconds. For active live locations only.
- ***heading*** (int, optional) – The direction in which user is moving, in degrees; 1-*360*. For active live locations only.
- ***proximity_alert_radius*** (int, optional) – Maximum distance for proximity alerts about approaching another chat member, in meters. For sent live locations only.
- ****kwargs*** (dict) – Arbitrary keyword arguments.

longitude

Longitude as defined by sender.

Type
float

latitude

Latitude as defined by sender.

Type
float

horizontal_accuracy

Optional. The radius of uncertainty for the location, measured in meters.

Type
float

live_period

Optional. Time relative to the message sending date, during which the location can be updated, in seconds. For active live locations only.

Type
int

heading

Optional. The direction in which user is moving, in degrees. For active live locations only.

Type
int

proximity_alert_radius

Optional. Maximum distance for proximity alerts about approaching another chat member, in meters. For sent live locations only.

Type
int

telegram.LoginUrl

class telegram.**LoginUrl**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *url* is equal.

Note: You must always check the hash of the received data to verify the authentication and the integrity of the data as described in Checking authorization

---

Parameters

- ***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, optional) – New text of the button in forwarded messages.
- ***bot_username*** (str, 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, optional) – Pass True to request the permission for your bot to send messages to the user.

url

An HTTPS URL to be opened with user authorization data.

Type
str

forward_text

Optional. New text of the button in forwarded messages.

Type
str

bot_username

Optional. Username of a bot, which will be used for user authorization.

Type
str

request_write_access

Optional. Pass `True` to request the permission for your bot to send messages to the user.

Type
`bool`

telegram.MenuButton

class telegram.**MenuButton**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

- *telegram.MenuButtonCommands*
- *telegram.MenuButtonWebApp*
- *telegram.MenuButtonDefault*

If a menu button other than *telegram.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.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *type* is equal. For subclasses with additional attributes, the notion of equality is overridden.

New in version 20.0.

Parameters
type (`str`) – Type of menu button that the instance represents.

type

Type of menu button that the instance represents.

Type
`str`

COMMANDS = 'commands'

telegram.constants.MenuButtonType.COMMANDS

DEFAULT = 'default'

telegram.constants.MenuButtonType.DEFAULT

WEB_APP = 'web_app'

telegram.constants.MenuButtonType.WEB_APP

classmethod de_json(*data*, *bot*)

Converts JSON data to the appropriate *MenuButton* object, i.e. takes care of selecting the correct subclass.

Parameters

- ***data*** (Dict[`str`, ...]) – The JSON data.
- ***bot*** (*telegram.Bot*) – The bot associated with this object.

Returns
The Telegram object.

telegram.MenuButtonCommands

class telegram.**MenuButtonCommands**(**args*, ***kwargs*)

Bases: *telegram.MenuButton*

Represents a menu button, which opens the bot's list of commands.

New in version 20.0.

type

'commands'.

Type
str

telegram.MenuButtonDefault

class telegram.**MenuButtonDefault**(**args*, ***kwargs*)

Bases: *telegram.MenuButton*

Describes that no specific value for the menu button was set.

New in version 20.0.

type

'default'.

Type
str

telegram.MenuButtonWebApp

class telegram.**MenuButtonWebApp**(**args*, ***kwargs*)

Bases: *telegram.MenuButton*

Represents a menu button, which launches a Web App.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *type*, *text* and *web_app* are equal.

New in version 20.0.

Parameters

- ***text*** (str) – Text of the button.
- ***web_app*** (*telegram.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 *answerWebAppQuery()*.

type

'web_app'.

Type
str

text

Text of the button.

Type
str

web_app

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 *answerWebAppQuery()*.

Type
telegram.WebAppInfo

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.Message

class telegram.**Message**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a message.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *message_id* and *chat* are equal.

Note: In Python `from` is a reserved word use *from_user* instead.

Changed in version 20.0:

- The arguments and attributes `voice_chat_scheduled`, `voice_chat_started` and `voice_chat_ended`, `voice_chat_participants_invited` were renamed to *video_chat_scheduled*/*video_chat_scheduled*, *video_chat_started*/*video_chat_started*, *video_chat_ended*/*video_chat_ended* and *video_chat_participants_invited*/*video_chat_participants_invited*, respectively, in accordance to Bot API 6.0.
- The following are now keyword-only arguments in Bot methods: `{read, write, connect, pool}_timeout`, `api_kwargs`, `contact`, `quote`, `filename`, `loaction`, `venue`. Use a named argument for those, and notice that some positional arguments changed position as a result.

Parameters

- ***message_id*** (`int`) – Unique message identifier inside this chat.
- ***from_user*** (*telegram.User*, optional) – Sender of the message; empty for messages sent to channels. For backward compatibility, this will contain a fake sender user in non-channel chats, if the message was sent on behalf of a chat.
- ***sender_chat*** (*telegram.Chat*, 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, *from_user* contains a fake sender user in non-channel chats, if the message was sent on behalf of a chat.
- ***date*** (`datetime.datetime`) – Date the message was sent in Unix time. Converted to `datetime.datetime`.
- ***chat*** (*telegram.Chat*) – Conversation the message belongs to.
- ***forward_from*** (*telegram.User*, optional) – For forwarded messages, sender of the original message.
- ***forward_from_chat*** (*telegram.Chat*, optional) – For messages forwarded from channels or from anonymous administrators, information about the original sender chat.

- ***forward_from_message_id*** (int, optional) – For forwarded channel posts, identifier of the original message in the channel.
- ***forward_sender_name*** (str, optional) – Sender's name for messages forwarded from users who disallow adding a link to their account in forwarded messages.
- ***forward_date*** (datetime.datetime, optional) – For forwarded messages, date the original message was sent in Unix time. Converted to datetime.datetime.
- ***is_automatic_forward*** (bool, optional) – True, if the message is a channel post that was automatically forwarded to the connected discussion group.

  New in version 13.9.
- ***reply_to_message*** (*telegram.Message*, optional) – For replies, the original message.
- ***edit_date*** (datetime.datetime, optional) – Date the message was last edited in Unix time. Converted to datetime.datetime.
- ***has_protected_content*** (bool, optional) – True, if the message can't be forwarded.

  New in version 13.9.
- ***media_group_id*** (str, optional) – The unique identifier of a media message group this message belongs to.
- ***text*** (str, optional) – For text messages, the actual UTF-8 text of the message, 0-*4096* characters.
- ***entities*** (List[*telegram.MessageEntity*], optional) – For text messages, special entities like usernames, URLs, bot commands, etc. that appear in the text. See *parse_entity* and *parse_entities* methods for how to use properly.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – For messages with a Caption. Special entities like usernames, URLs, bot commands, etc. that appear in the caption. See *Message.parse_caption_entity* and *parse_caption_entities* methods for how to use properly.
- ***audio*** (*telegram.Audio*, optional) – Message is an audio file, information about the file.
- ***document*** (*telegram.Document*, optional) – Message is a general file, information about the file.
- ***animation*** (*telegram.Animation*, optional) – Message is an animation, information about the animation. For backward compatibility, when this field is set, the document field will also be set.
- ***game*** (*telegram.Game*, optional) – Message is a game, information about the game.
- ***photo*** (List[*telegram.PhotoSize*], optional) – Message is a photo, available sizes of the photo.
- ***sticker*** (*telegram.Sticker*, optional) – Message is a sticker, information about the sticker.
- ***video*** (*telegram.Video*, optional) – Message is a video, information about the video.
- ***voice*** (*telegram.Voice*, optional) – Message is a voice message, information about the file.
- ***video_note*** (*telegram.VideoNote*, optional) – Message is a video note, information about the video message.
- ***new_chat_members*** (List[*telegram.User*], optional) – New members that were added to the group or supergroup and information about them (the bot itself may be one of these members).

- ***caption*** (str, optional) – Caption for the animation, audio, document, photo, video or voice, 0-*1024* characters.
- ***contact*** (*telegram.Contact*, optional) – Message is a shared contact, information about the contact.
- ***location*** (*telegram.Location*, optional) – Message is a shared location, information about the location.
- ***venue*** (*telegram.Venue*, optional) – Message is a venue, information about the venue. For backward compatibility, when this field is set, the location field will also be set.
- ***left_chat_member*** (*telegram.User*, optional) – A member was removed from the group, information about them (this member may be the bot itself).
- ***new_chat_title*** (str, optional) – A chat title was changed to this value.
- ***new_chat_photo*** (List[*telegram.PhotoSize*], optional) – A chat photo was changed to this value.
- ***delete_chat_photo*** (bool, optional) – Service message: The chat photo was deleted.
- ***group_chat_created*** (bool, optional) – Service message: The group has been created.
- ***supergroup_chat_created*** (bool, 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, 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*** (*telegram.MessageAutoDeleteTimerChanged*, optional) – Service message: auto-delete timer settings changed in the chat.

  New in version 13.4.
- ***migrate_to_chat_id*** (int, optional) – The group has been migrated to a supergroup with the specified identifier. This number 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.
- ***migrate_from_chat_id*** (int, optional) – The supergroup has been migrated from a group with the specified identifier. This number 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.
- ***pinned_message*** (*telegram.Message*, optional) – Specified message was pinned. Note that the Message object in this field will not contain further *reply_to_message* fields even if it is itself a reply.
- ***invoice*** (*telegram.Invoice*, optional) – Message is an invoice for a payment, information about the invoice.
- ***successful_payment*** (*telegram.SuccessfulPayment*, optional) – Message is a service message about a successful payment, information about the payment.
- ***connected_website*** (str, optional) – The domain name of the website on which the user has logged in.

- ***forward_signature*** (str, optional) – For messages forwarded from channels, signature of the post author if present.
- ***author_signature*** (str, optional) – Signature of the post author for messages in channels, or the custom title of an anonymous group administrator.
- ***passport_data*** (*telegram.PassportData*, optional) – Telegram Passport data.
- ***poll*** (*telegram.Poll*, optional) – Message is a native poll, information about the poll.
- ***dice*** (*telegram.Dice*, optional) – Message is a dice with random value from 1 to 6.
- ***via_bot*** (*telegram.User*, optional) – Message was sent through an inline bot.
- ***proximity_alert_triggered*** (*telegram.ProximityAlertTriggered*, optional) – Service message. A user in the chat triggered another user's proximity alert while sharing Live Location.
- ***video_chat_scheduled*** (*telegram.VideoChatScheduled*, optional) – Service message: video chat scheduled.

  New in version 20.0.
- ***video_chat_started*** (*telegram.VideoChatStarted*, optional) – Service message: video chat started.

  New in version 20.0.
- ***video_chat_ended*** (*telegram.VideoChatEnded*, optional) – Service message: video chat ended.

  New in version 20.0.
- ***video_chat_participants_invited*** (*telegram.VideoChatParticipantsInvited* optional) – Service message: new participants invited to a video chat.

  New in version 20.0.
- ***web_app_data*** (*telegram.WebAppData*, optional) – Service message: data sent by a Web App.

  New in version 20.0.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message. `login_url` buttons are represented as ordinary url buttons.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.

message_id

Unique message identifier inside this chat.

Type
int

from_user

Optional. Sender of the message; empty for messages sent to channels. For backward compatibility, this will contain a fake sender user in non-channel chats, if the message was sent on behalf of a chat.

Type
telegram.User

sender_chat

Optional. Sender of the message, sent on behalf of a chat. For backward compatibility, *from_user* contains a fake sender user in non-channel chats, if the message was sent on behalf of a chat.

Type
telegram.Chat

date

Date the message was sent.

Type
datetime.datetime

chat

Conversation the message belongs to.

Type
telegram.Chat

forward_from

Optional. Sender of the original message.

Type
telegram.User

forward_from_chat

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

Type
telegram.Chat

forward_from_message_id

Optional. Identifier of the original message in the channel.

Type
int

forward_date

Optional. Date the original message was sent.

Type
datetime.datetime

is_automatic_forward

Optional. True, if the message is a channel post that was automatically forwarded to the connected discussion group.

New in version 13.9.

Type
bool

reply_to_message

Optional. For replies, the original message. Note that the Message object in this field will not contain further `reply_to_message` fields even if it itself is a reply.

Type
telegram.Message

edit_date

Optional. Date the message was last edited.

Type
datetime.datetime

has_protected_content

Optional. True, if the message can't be forwarded.

New in version 13.9.

Type
bool

media_group_id

Optional. The unique identifier of a media message group this message belongs to.

Type
str

text

Optional. The actual UTF-8 text of the message.

Type
str

entities

Special entities like usernames, URLs, bot commands, etc. that appear in the text. See *Message.parse_entity* and *parse_entities* methods for how to use properly. This list is empty if the message does not contain entities.

Type
List[*telegram.MessageEntity*]

caption_entities

Special entities like usernames, URLs, bot commands, etc. that appear in the caption. See *Message.parse_caption_entity* and *parse_caption_entities* methods for how to use properly. This list is empty if the message does not contain caption entities.

Type
List[*telegram.MessageEntity*]

audio

Optional. Information about the file.

Type
telegram.Audio

document

Optional. Information about the file.

Type
telegram.Document

animation

For backward compatibility, when this field is set, the document field will also be set.

Type
telegram.Animation

game

Optional. Information about the game.

Type
telegram.Game

photo

Available sizes of the photo. This list is empty if the message does not contain a photo.

Type
List[*telegram.PhotoSize*]

sticker

Optional. Information about the sticker.

Type
telegram.Sticker

video

Optional. Information about the video.

Type
telegram.Video

voice

Optional. Information about the file.

Type
telegram.Voice

video_note

Optional. Information about the video message.

Type
telegram.VideoNote

new_chat_members

Information about new members to the chat. The bot itself may be one of these members. This list is empty if the message does not contain new chat members.

Type
List[*telegram.User*]

caption

Optional. Caption for the document, photo or video, 0-*1024* characters.

Type
str

contact

Optional. Information about the contact.

Type
telegram.Contact

location

Optional. Information about the location.

Type
telegram.Location

venue

Optional. Information about the venue.

Type
telegram.Venue

left_chat_member

Optional. Information about the user that left the group. (this member may be the bot itself).

Type
telegram.User

new_chat_title

Optional. A chat title was changed to this value.

Type
str

new_chat_photo

A chat photo was changed to this value. This list is empty if the message does not contain a new chat photo.

Type
List[*telegram.PhotoSize*]

delete_chat_photo

Optional. The chat photo was deleted.

Type
bool

group_chat_created

Optional. The group has been created.

Type
bool

supergroup_chat_created

Optional. The supergroup has been created.

Type
bool

channel_chat_created

Optional. The channel has been created.

Type
bool

message_auto_delete_timer_changed

Optional. Service message: auto-delete timer settings changed in the chat.

New in version 13.4.

Type
telegram.MessageAutoDeleteTimerChanged

migrate_to_chat_id

Optional. The group has been migrated to a supergroup with the specified identifier.

Type
int

migrate_from_chat_id

Optional. The supergroup has been migrated from a group with the specified identifier.

Type
int

pinned_message

Optional. Specified message was pinned.

Type
telegram.Message

invoice

Optional. Information about the invoice.

Type
telegram.Invoice

successful_payment

Optional. Information about the payment.

Type
telegram.SuccessfulPayment

connected_website

Optional. The domain name of the website on which the user has logged in.

Type
str

forward_signature

Optional. Signature of the post author for messages forwarded from channels.

Type
str

forward_sender_name

Optional. Sender's name for messages forwarded from users who disallow adding a link to their account in forwarded messages.

Type
str

author_signature

Optional. Signature of the post author for messages in channels, or the custom title of an anonymous group administrator.

Type
str

passport_data

Optional. Telegram Passport data.

Type
telegram.PassportData

poll

Optional. Message is a native poll, information about the poll.

Type
telegram.Poll

dice

Optional. Message is a dice.

Type
telegram.Dice

via_bot

Optional. Bot through which the message was sent.

Type
telegram.User

proximity_alert_triggered

Optional. Service message. A user in the chat triggered another user's proximity alert while sharing Live Location.

Type
telegram.ProximityAlertTriggered

video_chat_scheduled

Optional. Service message: video chat scheduled.

New in version 20.0.

Type
telegram.VideoChatScheduled

video_chat_started

Optional. Service message: video chat started.

New in version 20.0.

Type
telegram.VideoChatStarted

video_chat_ended

Optional. Service message: video chat ended.

New in version 20.0.

Type
telegram.VideoChatEnded

video_chat_participants_invited

Optional. Service message: new participants invited to a video chat.

New in version 20.0.

Type
telegram.VideoChatParticipantsInvited

web_app_data

Optional. Service message: data sent by a Web App.

New in version 20.0.

Type
telegram.WebAppData

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

bot

Optional. The Bot to use for instance methods.

Type
telegram.Bot

property caption_html

Creates an HTML-formatted string from the markup entities found in the message's caption.

Use this if you want to retrieve the message caption with the caption entities formatted as HTML in the same way the original message was formatted.

Note: Custom emoji entities will currently be ignored by this function. Instead, the supplied replacement for the emoji will be used.

Changed in version 13.10: Spoiler entities are now formatted as HTML.

Returns
Message caption with caption entities formatted as HTML.

Return type
str

property caption_html_urled

Creates an HTML-formatted string from the markup entities found in the message's caption.

Use this if you want to retrieve the message caption with the caption entities formatted as HTML. This also formats *telegram.MessageEntity.URL* as a hyperlink.

Note: Custom emoji entities will currently be ignored by this function. Instead, the supplied replacement for the emoji will be used.

Changed in version 13.10: Spoiler entities are now formatted as HTML.

Returns
Message caption with caption entities formatted as HTML.

Return type
str

property caption_markdown

Creates an Markdown-formatted string from the markup entities found in the message's caption using *telegram.constants.ParseMode.MARKDOWN*.

Use this if you want to retrieve the message caption with the caption entities formatted as Markdown in the same way the original message was formatted.

Note: *'Markdown'* is a legacy mode, retained by Telegram for backward compatibility. You should use *caption_markdown_v2()* instead.

Custom emoji entities will currently be ignored by this function. Instead, the supplied replacement for the emoji will be used.

Returns
Message caption with caption entities formatted as Markdown.

Return type
str

Raises
ValueError – If the message contains underline, strikethrough, spoiler or nested entities.

property caption_markdown_urled

Creates an Markdown-formatted string from the markup entities found in the message's caption using *telegram.constants.ParseMode.MARKDOWN*.

Use this if you want to retrieve the message caption with the caption entities formatted as Markdown. This also formats *telegram.MessageEntity.URL* as a hyperlink.

Note: *'Markdown'* is a legacy mode, retained by Telegram for backward compatibility. You should use *caption_markdown_v2_urled()* instead.

Custom emoji entities will currently be ignored by this function. Instead, the supplied replacement for the emoji will be used.

Returns
Message caption with caption entities formatted as Markdown.

Return type
str

Raises
ValueError – If the message contains underline, strikethrough, spoiler or nested entities.

property caption_markdown_v2

Creates an Markdown-formatted string from the markup entities found in the message's caption using *telegram.constants.ParseMode.MARKDOWN_V2*.

Use this if you want to retrieve the message caption with the caption entities formatted as Markdown in the same way the original message was formatted.

Note: Custom emoji entities will currently be ignored by this function. Instead, the supplied replacement for the emoji will be used.

Changed in version 13.10: Spoiler entities are now formatted as Markdown V2.

Returns
Message caption with caption entities formatted as Markdown.

Return type
str

property caption_markdown_v2_urled

Creates an Markdown-formatted string from the markup entities found in the message's caption using *telegram.constants.ParseMode.MARKDOWN_V2*.

Use this if you want to retrieve the message caption with the caption entities formatted as Markdown. This also formats *telegram.MessageEntity.URL* as a hyperlink.

Note: Custom emoji entities will currently be ignored by this function. Instead, the supplied replacement for the emoji will be used.

Changed in version 13.10: Spoiler entities are now formatted as Markdown V2.

Returns
Message caption with caption entities formatted as Markdown.

Return type
str

property chat_id

Shortcut for *telegram.Chat.id* for *chat*.

Type
int

async copy(*chat_id*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *disable_notification=None*, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *reply_markup=None*, *protect_content=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.copy_message(
    chat_id=chat_id,
    from_chat_id=update.effective_message.chat_id,
    message_id=update.effective_message.message_id,
    *args,
    **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.copy_message()*.

Returns
On success, returns the MessageId of the sent message.

Return type
telegram.MessageId

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

async delete(***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.delete_message(
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.delete_message()*.

Returns
On success, `True` is returned.

Return type
`bool`

async edit_caption(*caption=None*, *reply_markup=None*, *parse_mode=None*, *caption_entities=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.edit_message_caption(
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.edit_message_caption()*.

Note: You can only edit messages that the bot sent itself (i.e. of the `bot.send_*` family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

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

Return type
telegram.Message

async edit_live_location(*latitude=None*, *longitude=None*, *reply_markup=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, ***, *location=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.edit_message_live_location(
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.edit_message_live_location()*.

Note: You can only edit messages that the bot sent itself (i.e. of the `bot.send_*` family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and

might be changed by Telegram.

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

Return type
telegram.Message

async edit_media(*media*, *reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.edit_message_media(
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.edit_message_media()*.

Note: You can only edit messages that the bot sent itself(i.e. of the `bot.send_*` family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

Returns
On success, if edited message is not an inline message, the edited Message is returned, otherwise `True` is returned.

Return type
telegram.Message

async edit_reply_markup(*reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.edit_message_reply_markup(
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.edit_message_reply_markup()*.

Note: You can only edit messages that the bot sent itself (i.e. of the `bot.send_*` family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

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

Return type
telegram.Message

async edit_text(*text*, *parse_mode=None*, *disable_web_page_preview=None*, *reply_markup=None*, *entities=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.edit_message_text(
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.edit_message_text()*.

Note: You can only edit messages that the bot sent itself (i.e. of the `bot.send_*` family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

Returns

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

Return type

telegram.Message

property effective_attachment

If this message is neither a plain text message nor a status update, this gives the attachment that this message was sent with. This may be one of

- *telegram.Audio*
- *telegram.Dice*
- *telegram.Contact*
- *telegram.Document*
- *telegram.Animation*
- *telegram.Game*
- *telegram.Invoice*
- *telegram.Location*
- *telegram.PassportData*
- List[*telegram.PhotoSize*]
- *telegram.Poll*
- *telegram.Sticker*
- *telegram.SuccessfulPayment*
- *telegram.Venue*
- *telegram.Video*
- *telegram.VideoNote*
- *telegram.Voice*

Otherwise `None` is returned.

Changed in version 20.0: *dice*, *passport_data* and *poll* are now also considered to be an attachment.

async forward(*chat_id*, *disable_notification=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.forward_message(
    from_chat_id=update.effective_message.chat_id,
    message_id=update.effective_message.message_id,
    *args,
    **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.forward_message()*.

Note: Since the release of Bot API 5.5 it can be impossible to forward messages from some chats. Use the attributes *telegram.Message.has_protected_content* and *telegram.Chat.has_protected_content* to check this.

As a workaround, it is still possible to use *copy()*. However, this behaviour is undocumented and might be changed by Telegram.

Returns
On success, instance representing the message forwarded.

Return type
telegram.Message

async get_game_high_scores(*user_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.get_game_high_scores(
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.get_game_high_scores()*.

Note: You can only edit messages that the bot sent itself (i.e. of the `bot.send_*` family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

Returns
List[*telegram.GameHighScore*]

property id

Shortcut for *message_id*.

New in version 20.0.

Type
int

property link

Convenience property. If the chat of the message is not a private chat or normal group, returns a t.me link of the message.

Type
str

parse_caption_entities(*types=None*)

Returns a dict that maps *telegram.MessageEntity* to str. It contains entities from this message's caption filtered by their *telegram.MessageEntity.type* attribute as the key, and the text that each entity belongs to as the value of the dict.

Note: This method should always be used instead of the *caption_entities* attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See *parse_entity* for more info.

Parameters

types (List[str], optional) – List of *telegram.MessageEntity* types as strings. If the type attribute of an entity is contained in this list, it will be returned. Defaults to a list of all types. All types can be found as constants in *telegram.MessageEntity*.

Returns

A dictionary of entities mapped to the text that belongs to them, calculated based on UTF-16 codepoints.

Return type

Dict[*telegram.MessageEntity*, str]

parse_caption_entity(*entity*)

Returns the text from a given *telegram.MessageEntity*.

Note: This method is present because Telegram calculates the offset and length in UTF-16 codepoint pairs, which some versions of Python don't handle automatically. (That is, you can't just slice Message.caption with the offset and length.)

Parameters

entity (*telegram.MessageEntity*) – The entity to extract the text from. It must be an entity that belongs to this message.

Returns

The text of the given entity.

Return type

str

Raises

RuntimeError – If the message has no caption.

parse_entities(*types=None*)

Returns a dict that maps *telegram.MessageEntity* to str. It contains entities from this message filtered by their *telegram.MessageEntity.type* attribute as the key, and the text that each entity belongs to as the value of the dict.

Note: This method should always be used instead of the *entities* attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See *parse_entity* for more info.

Parameters

types (List[str], optional) – List of *telegram.MessageEntity* types as strings. If the type attribute of an entity is contained in this list, it will be returned. Defaults to a list of all types. All types can be found as constants in *telegram.MessageEntity*.

Returns
A dictionary of entities mapped to the text that belongs to them, calculated based on UTF-16 codepoints.

Return type
Dict[*telegram.MessageEntity*, str]

parse_entity(*entity*)

Returns the text from a given *telegram.MessageEntity*.

Note: This method is present because Telegram calculates the offset and length in UTF-16 codepoint pairs, which some versions of Python don't handle automatically. (That is, you can't just slice `Message.text` with the offset and length.)

Parameters
entity (*telegram.MessageEntity*) – The entity to extract the text from. It must be an entity that belongs to this message.

Returns
The text of the given entity.

Return type
str

Raises
RuntimeError – If the message has no text.

async pin(*disable_notification=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.pin_chat_message(
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.pin_chat_message()*.

Returns
On success, True is returned.

Return type
bool

async reply_animation(*animation*, *duration=None*, *width=None*, *height=None*, *thumb=None*, *caption=None*, *parse_mode=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *quote=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_animation(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_animation()*.

Keyword Arguments
quote (bool, optional) – If set to True, the animation is sent as an actual reply to this message. If `reply_to_message_id` is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_audio(*audio*, *duration=None*, *performer=None*, *title=None*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *thumb=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *quote=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_audio(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_audio()*.

Keyword Arguments
quote (bool, optional) – If set to True, the audio is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_chat_action(*action*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_chat_action(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_chat_action()*.

New in version 13.2.

Returns
On success, True is returned.

Return type
bool

async reply_contact(*phone_number=None*, *first_name=None*, *last_name=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *vcard=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *contact=None*, *quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_contact(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_contact()*.

Keyword Arguments
quote (bool, optional) – If set to True, the contact is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_copy(*from_chat_id*, *message_id*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *disable_notification=None*, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *reply_markup=None*, *protect_content=None*, ***, *quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.copy_message(
    chat_id=message.chat.id,
    message_id=message_id,
    *args,
    **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.copy_message()*.

Keyword Arguments
quote (bool, optional) – If set to True, the copy is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

New in version 13.1.

Returns
On success, returns the MessageId of the sent message.

Return type
telegram.MessageId

async reply_dice(*disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *emoji=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_dice(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_dice()*.

Keyword Arguments
quote (bool, optional) – If set to True, the dice is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_document(*document*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *thumb=None*, *disable_content_type_detection=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *quote=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_document(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_document()*.

Keyword Arguments
quote (bool, optional) – If set to True, the document is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_game(*game_short_name*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_game(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_game()*.

Keyword Arguments
quote (bool, optional) – If set to True, the game is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

New in version 13.2.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_html(*text*, *disable_web_page_preview=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *entities=None*, *protect_content=None*, ***, *quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_message(
    update.effective_message.chat_id,
    parse_mode=ParseMode.HTML,
    *args,
    **kwargs,
)
```

Sends a message with HTML formatting.

For the documentation of the arguments, please see *telegram.Bot.send_message()*.

Keyword Arguments
quote (bool, optional) – If set to True, the message is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_invoice(*title*, *description*, *payload*, *provider_token*, *currency*, *prices*, *start_parameter=None*, *photo_url=None*, *photo_size=None*, *photo_width=None*, *photo_height=None*, *need_name=None*, *need_phone_number=None*, *need_email=None*, *need_shipping_address=None*, *is_flexible=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *provider_data=None*, *send_phone_number_to_provider=None*, *send_email_to_provider=None*, *allow_sending_without_reply=None*, *max_tip_amount=None*, *suggested_tip_amounts=None*, *protect_content=None*, ***, *quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_invoice(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_invoice()*.

Warning: As of API 5.2 *start_parameter* is an optional argument and therefore the order of the arguments had to be changed. Use keyword arguments to make sure that the arguments are passed correctly.

New in version 13.2.

Changed in version 13.5: As of Bot API 5.2, the parameter *start_parameter* is optional.

Keyword Arguments
quote (bool, optional) – If set to True, the invoice is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_location(*latitude=None*, *longitude=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *live_period=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *location=None*, *quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_location(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_location()*.

Keyword Arguments
quote (bool, optional) – If set to True, the location is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_markdown(*text*, *disable_web_page_preview=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *entities=None*, *protect_content=None*, ***, *quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_message(
    update.effective_message.chat_id,
    parse_mode=ParseMode.MARKDOWN,
    *args,
    **kwargs,
)
```

Sends a message with Markdown version 1 formatting.

For the documentation of the arguments, please see *telegram.Bot.send_message()*.

Note: *'Markdown'* is a legacy mode, retained by Telegram for backward compatibility. You should use *reply_markdown_v2()* instead.

Keyword Arguments

quote (bool, optional) – If set to True, the message is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns

On success, instance representing the message posted.

Return type

telegram.Message

async reply_markdown_v2(*text*, *disable_web_page_preview=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *entities=None*, *protect_content=None*, ***, *quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_message(
    update.effective_message.chat_id,
    parse_mode=ParseMode.MARKDOWN_V2,
    *args,
    **kwargs,
)
```

Sends a message with markdown version 2 formatting.

For the documentation of the arguments, please see *telegram.Bot.send_message()*.

Keyword Arguments

quote (bool, optional) – If set to True, the message is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns

On success, instance representing the message posted.

Return type
telegram.Message

async reply_media_group(*media*, *disable_notification=None*, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *quote=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_media_group(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_media_group()*.

Keyword Arguments
quote (bool, optional) – If set to True, the media group is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns
An array of the sent Messages.

Return type
List[*telegram.Message*]

Raises
telegram.error.TelegramError –

async reply_photo(*photo*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *quote=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_photo(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_photo()*.

Keyword Arguments
quote (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_poll(*question*, *options*, *is_anonymous=None*, *type=None*, *allows_multiple_answers=None*, *correct_option_id=None*, *is_closed=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *explanation=None*, *explanation_parse_mode=None*, *open_period=None*, *close_date=None*, *allow_sending_without_reply=None*, *explanation_entities=None*, *protect_content=None*, ***, *quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_poll(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_poll()*.

Keyword Arguments
quote (bool, optional) – If set to True, the poll is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_sticker(*sticker*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *protect_content=None*, *, *quote=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_sticker(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_sticker()*.

Keyword Arguments
quote (bool, optional) – If set to True, the sticker is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_text(*text*, *parse_mode=None*, *disable_web_page_preview=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *entities=None*, *protect_content=None*, *, *quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_message(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_message()*.

Keyword Arguments
quote (bool, optional) – If set to True, the message is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_venue(*latitude=None*, *longitude=None*, *title=None*, *address=None*, *foursquare_id=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *foursquare_type=None*, *google_place_id=None*, *google_place_type=None*, *allow_sending_without_reply=None*, *protect_content=None*, *, *venue=None*, *quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_venue(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_venue()*.

Keyword Arguments
quote (bool, optional) – If set to True, the venue is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_video(*video*, *duration=None*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *width=None*, *height=None*, *parse_mode=None*, *supports_streaming=None*, *thumb=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *quote=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_video(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_video()*.

Keyword Arguments
quote (bool, optional) – If set to True, the video is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_video_note(*video_note*, *duration=None*, *length=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *thumb=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *filename=None*, *quote=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_video_note(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_video_note()*.

Keyword Arguments
quote (bool, optional) – If set to True, the video note is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_voice(*voice*, *duration=None*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, *, *filename=None*, *quote=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_voice(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_voice()*.

Keyword Arguments

quote (bool, optional) – If set to True, the voice note is sent as an actual reply to this message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns

On success, instance representing the message posted.

Return type

telegram.Message

async set_game_score(*user_id*, *score*, *force=None*, *disable_edit_message=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.set_game_score(
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.set_game_score()*.

Note: You can only edit messages that the bot sent itself (i.e. of the bot.send_* family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

Returns

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

Return type

telegram.Message

async stop_live_location(*reply_markup=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.stop_message_live_location(
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.stop_message_live_location()*.

Note: You can only edit messages that the bot sent itself (i.e. of the bot.send_* family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and

might be changed by Telegram.

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

Return type
telegram.Message

async stop_poll(*reply_markup=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.stop_poll(
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.stop_poll()*.

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

Return type
telegram.Poll

property text_html

Creates an HTML-formatted string from the markup entities found in the message.

Use this if you want to retrieve the message text with the entities formatted as HTML in the same way the original message was formatted.

Note: Custom emoji entities will currently be ignored by this function. Instead, the supplied replacement for the emoji will be used.

Changed in version 13.10: Spoiler entities are now formatted as HTML.

Returns
Message text with entities formatted as HTML.

Return type
str

property text_html_urled

Creates an HTML-formatted string from the markup entities found in the message.

Use this if you want to retrieve the message text with the entities formatted as HTML. This also formats *telegram.MessageEntity.URL* as a hyperlink.

Note: Custom emoji entities will currently be ignored by this function. Instead, the supplied replacement for the emoji will be used.

Changed in version 13.10: Spoiler entities are now formatted as HTML.

Returns
Message text with entities formatted as HTML.

Return type
str

property text_markdown

Creates an Markdown-formatted string from the markup entities found in the message using *telegram.constants.ParseMode.MARKDOWN*.

Use this if you want to retrieve the message text with the entities formatted as Markdown in the same way the original message was formatted.

Note: *'Markdown'* is a legacy mode, retained by Telegram for backward compatibility. You should use *text_markdown_v2()* instead.

Custom emoji entities will currently be ignored by this function. Instead, the supplied replacement for the emoji will be used.

Returns
Message text with entities formatted as Markdown.

Return type
str

Raises
ValueError – If the message contains underline, strikethrough, spoiler or nested entities.

property text_markdown_urled

Creates an Markdown-formatted string from the markup entities found in the message using *telegram.constants.ParseMode.MARKDOWN*.

Use this if you want to retrieve the message text with the entities formatted as Markdown. This also formats *telegram.MessageEntity.URL* as a hyperlink.

Note: *'Markdown'* is a legacy mode, retained by Telegram for backward compatibility. You should use *text_markdown_v2_urled()* instead.

Custom emoji entities will currently be ignored by this function. Instead, the supplied replacement for the emoji will be used.

Returns
Message text with entities formatted as Markdown.

Return type
str

Raises
ValueError – If the message contains underline, strikethrough, spoiler or nested entities.

property text_markdown_v2

Creates an Markdown-formatted string from the markup entities found in the message using *telegram.constants.ParseMode.MARKDOWN_V2*.

Use this if you want to retrieve the message text with the entities formatted as Markdown in the same way the original message was formatted.

Note: Custom emoji entities will currently be ignored by this function. Instead, the supplied replacement for the emoji will be used.

Changed in version 13.10: Spoiler entities are now formatted as Markdown V2.

Returns
Message text with entities formatted as Markdown.

Return type
str

property text_markdown_v2_urled

Creates an Markdown-formatted string from the markup entities found in the message using *telegram.constants.ParseMode.MARKDOWN_V2*.

Use this if you want to retrieve the message text with the entities formatted as Markdown. This also formats *telegram.MessageEntity.URL* as a hyperlink.

Note: Custom emoji entities will currently be ignored by this function. Instead, the supplied replacement for the emoji will be used.

Changed in version 13.10: Spoiler entities are now formatted as Markdown V2.

Returns
Message text with entities formatted as Markdown.

Return type
str

to_dict()

See *telegram.TelegramObject.to_dict()*.

async unpin(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.unpin_chat_message(
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.unpin_chat_message()*.

Returns
On success, True is returned.

Return type
bool

telegram.MessageAutoDeleteTimerChanged

class telegram.**MessageAutoDeleteTimerChanged**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a service message about a change in auto-delete timer settings.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *message_auto_delete_time* is equal.

New in version 13.4.

Parameters

- ***message_auto_delete_time*** (int) – New auto-delete time for messages in the chat.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

message_auto_delete_time

New auto-delete time for messages in the chat.

Type
int

telegram.MessageEntity

class telegram.**MessageEntity**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *type*, *offset* and *length* are equal.

Parameters

- ***type*** (str) – Type of the entity. Can be *MENTION* (@username), *HASHTAG*, *BOT_COMMAND*, *URL*, *EMAIL*, *PHONE_NUMBER*, *BOLD* (bold text), *ITALIC* (italic text), *STRIKETHROUGH*, *SPOILER* (spoiler message), *CODE* (monowidth string), *PRE* (monowidth block), *TEXT_LINK* (for clickable text URLs), *TEXT_MENTION* (for users without usernames), *CUSTOM_EMOJI* (for inline custom emoji stickers).

  New in version 20.0: added inline custom emoji

- ***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, optional) – For *TEXT_LINK* only, url that will be opened after user taps on the text.
- ***user*** (*telegram.User*, optional) – For *TEXT_MENTION* only, the mentioned user.
- ***language*** (str, optional) – For *PRE* only, the programming language of the entity text.
- ***custom_emoji_id*** (str, optional) – For *CUSTOM_EMOJI* only, unique identifier of the custom emoji. Use *telegram.Bot.get_custom_emoji_stickers()* to get full information about the sticker.

  New in version 20.0.

type

Type of the entity.

Type
str

offset

Offset in UTF-16 code units to the start of the entity.

Type
int

length

Length of the entity in UTF-16 code units.

Type
int

url

Optional. Url that will be opened after user taps on the text.

Type
str

user

Optional. The mentioned user.

Type
telegram.User

language

Optional. Programming language of the entity text.

Type
str

custom_emoji_id

Optional. Unique identifier of the custom emoji.

New in version 20.0.

Type
str

ALL_TYPES = [<MessageEntityType.MENTION>, <MessageEntityType.HASHTAG>, <MessageEntityType.CASHTAG>, <MessageEntityType.PHONE_NUMBER>, <MessageEntityType.BOT_COMMAND>, <MessageEntityType.URL>, <MessageEntityType.EMAIL>, <MessageEntityType.BOLD>, <MessageEntityType.ITALIC>, <MessageEntityType.CODE>, <MessageEntityType.PRE>, <MessageEntityType.TEXT_LINK>, <MessageEntityType.TEXT_MENTION>, <MessageEntityType.UNDERLINE>, <MessageEntityType.STRIKETHROUGH>, <MessageEntityType.SPOILER>, <MessageEntityType.CUSTOM_EMOJI>]

A list of all available message entity types.

Type
List[str]

BOLD = 'bold'

telegram.constants.MessageEntityType.BOLD

BOT_COMMAND = 'bot_command'

telegram.constants.MessageEntityType.BOT_COMMAND

CASHTAG = 'cashtag'

telegram.constants.MessageEntityType.CASHTAG

CODE = 'code'

telegram.constants.MessageEntityType.CODE

CUSTOM_EMOJI = 'custom_emoji'

telegram.constants.MessageEntityType.CUSTOM_EMOJI

New in version 20.0.

EMAIL = 'email'

telegram.constants.MessageEntityType.EMAIL

HASHTAG = 'hashtag'

telegram.constants.MessageEntityType.HASHTAG

ITALIC = 'italic'

telegram.constants.MessageEntityType.ITALIC

MENTION = 'mention'

telegram.constants.MessageEntityType.MENTION

PHONE_NUMBER = 'phone_number'

telegram.constants.MessageEntityType.PHONE_NUMBER

PRE = 'pre'

telegram.constants.MessageEntityType.PRE

SPOILER = 'spoiler'

telegram.constants.MessageEntityType.SPOILER

New in version 13.10.

STRIKETHROUGH = 'strikethrough'

telegram.constants.MessageEntityType.STRIKETHROUGH

TEXT_LINK = 'text_link'

telegram.constants.MessageEntityType.TEXT_LINK

TEXT_MENTION = 'text_mention'

telegram.constants.MessageEntityType.TEXT_MENTION

UNDERLINE = 'underline'

telegram.constants.MessageEntityType.UNDERLINE

URL = 'url'

telegram.constants.MessageEntityType.URL

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

telegram.MessageId

class telegram.**MessageId**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a unique message identifier.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *message_id* is equal.

message_id

Unique message identifier

Type

int

telegram.PhotoSize

class telegram.**PhotoSize**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents one size of a photo or a file/sticker thumbnail.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *file_unique_id* is equal.

Parameters

- ***file_id*** (str) – Identifier for this file, which can be used to download or reuse the file.
- ***file_unique_id*** (str) – Unique identifier for this file, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.
- ***width*** (int) – Photo width.
- ***height*** (int) – Photo height.
- ***file_size*** (int, optional) – File size in bytes.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

file_id

Identifier for this file.

Type
str

file_unique_id

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

width

Photo width.

Type
int

height

Photo height.

Type
int

file_size

Optional. File size in bytes.

Type
int

bot

Optional. The Bot to use for instance methods.

Type
telegram.Bot

async get_file(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Convenience wrapper over *telegram.Bot.get_file*

For the documentation of the arguments, please see *telegram.Bot.get_file()*.

Returns
telegram.File

Raises
telegram.error.TelegramError –

telegram.Poll

class telegram.**Poll**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object contains information about a poll.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *id* is equal.

See also:

Pollbot Example

Parameters

- ***id*** (str) – Unique poll identifier.
- ***question*** (str) – Poll question, 1-300 characters.
- ***options*** (List[*PollOption*]) – List of poll options.
- ***is_closed*** (bool) – True, if the poll is closed.
- ***is_anonymous*** (bool) – True, if the poll is anonymous.
- ***type*** (str) – Poll type, currently can be *REGULAR* or *QUIZ*.
- ***allows_multiple_answers*** (bool) – True, if the poll allows multiple answers.
- ***correct_option_id*** (int, 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, 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[*telegram.MessageEntity*], optional) – Special entities like usernames, URLs, bot commands, etc. that appear in the *explanation*.
- ***open_period*** (int, optional) – Amount of time in seconds the poll will be active after creation.
- ***close_date*** (datetime.datetime, optional) – Point in time (Unix timestamp) when the poll will be automatically closed. Converted to datetime.datetime.

id

Unique poll identifier.

Type
str

question

Poll question, 1-300 characters.

Type
str

options

List of poll options.

Type
List[*PollOption*]

total_voter_count

Total number of users that voted in the poll.

Type
int

is_closed

True, if the poll is closed.

Type
bool

is_anonymous

True, if the poll is anonymous.

Type
bool

type

Poll type, currently can be *REGULAR* or *QUIZ*.

Type
str

allows_multiple_answers

True, if the poll allows multiple answers.

Type
bool

correct_option_id

Optional. Identifier of the correct answer option.

Type
int

explanation

Optional. Text that is shown when a user chooses an incorrect answer or taps on the lamp icon in a quiz-style poll.

Type
str

explanation_entities

Special entities like usernames, URLs, bot commands, etc. that appear in the *explanation*. This list is empty if the message does not contain explanation entities.

Changed in version 20.0: This attribute is now always a (possibly empty) list and never None.

Type
List[*telegram.MessageEntity*]

open_period

Optional. Amount of time in seconds the poll will be active after creation.

Type
int

close_date

Optional. Point in time when the poll will be automatically closed.

Type
datetime.datetime

MAX_OPTION_LENGTH = 100

telegram.constants.PollLimit.OPTION_LENGTH

MAX_OPTION_NUMBER = 10

telegram.constants.PollLimit.OPTION_NUMBER

New in version 20.0.

MAX_QUESTION_LENGTH = 300

telegram.constants.PollLimit.QUESTION_LENGTH

QUIZ = 'quiz'

telegram.constants.PollType.QUIZ

REGULAR = 'regular'

telegram.constants.PollType.REGULAR

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

parse_explanation_entities(*types=None*)

Returns a dict that maps *telegram.MessageEntity* to str. It contains entities from this polls explanation filtered by their type attribute as the key, and the text that each entity belongs to as the value of the dict.

Note: This method should always be used instead of the *explanation_entities* attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See *parse_explanation_entity* for more info.

Parameters

types (List[str], optional) – List of MessageEntity types as strings. If the type attribute of an entity is contained in this list, it will be returned. Defaults to *telegram.MessageEntity.ALL_TYPES*.

Returns

A dictionary of entities mapped to the text that belongs to them, calculated based on UTF-16 codepoints.

Return type

Dict[*telegram.MessageEntity*, str]

parse_explanation_entity(*entity*)

Returns the text from a given *telegram.MessageEntity*.

Note: This method is present because Telegram calculates the offset and length in UTF-16 codepoint pairs, which some versions of Python don't handle automatically. (That is, you can't just slice Message.text with the offset and length.)

Parameters

entity (*telegram.MessageEntity*) – The entity to extract the text from. It must be an entity that belongs to this message.

Returns

The text of the given entity.

Return type

str

Raises

RuntimeError – If the poll has no explanation.

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.PollAnswer

class telegram.**PollAnswer**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents an answer of a user in a non-anonymous poll.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *poll_id*, *user* and *option_ids* are equal.

Parameters

- ***poll_id*** (str) – Unique poll identifier.

- ***user*** (*telegram.User*) – The user, who changed the answer to the poll.
- ***option_ids*** (List[int]) – 0-based identifiers of answer options, chosen by the user. May be empty if the user retracted their vote.

poll_id

Unique poll identifier.

Type
str

user

The user, who changed the answer to the poll.

Type
telegram.User

option_ids

Identifiers of answer options, chosen by the user.

Type
List[int]

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

telegram.PollOption

class telegram.**PollOption**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *text* and *voter_count* are equal.

Parameters

- ***text*** (str) – Option text, 1-100 characters.
- ***voter_count*** (int) – Number of users that voted for this option.

text

Option text, 1-100 characters.

Type
str

voter_count

Number of users that voted for this option.

Type
int

MAX_LENGTH = 100

telegram.constants.PollLimit.OPTION_LENGTH

telegram.ProximityAlertTriggered

class telegram.**ProximityAlertTriggered**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *traveler*, *watcher* and *distance* are equal.

Parameters

- ***traveler*** (*telegram.User*) – User that triggered the alert
- ***watcher*** (*telegram.User*) – User that set the alert
- ***distance*** (int) – The distance between the users

traveler

User that triggered the alert

Type
telegram.User

watcher

User that set the alert

Type
telegram.User

distance

The distance between the users

Type
int

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

telegram.ReplyKeyboardMarkup

class telegram.**ReplyKeyboardMarkup**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a custom keyboard with reply options.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their size of *keyboard* and all the buttons are equal.

Example

A user requests to change the bot's language, bot replies to the request with a keyboard to select the new language. Other users in the group don't see the keyboard.

Parameters

- ***keyboard*** (List[List[str | *telegram.KeyboardButton*]]) – Array of button rows, each represented by an Array of *telegram.KeyboardButton* objects.
- ***resize_keyboard*** (bool, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to False, in which case the custom keyboard is always of the same height as the app's standard keyboard.

- ***one_time_keyboard*** (bool, 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.
- ***selective*** (bool, 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 *telegram.Message* object.
  2) If the bot's message is a reply (has `reply_to_message_id`), sender of the original message.

  Defaults to False.
- ***input_field_placeholder*** (str, optional) – The placeholder to be shown in the input field when the keyboard is active; 1-64 characters.

  New in version 13.7.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

keyboard

Array of button rows.

Type
List[List[*telegram.KeyboardButton* | str]]

resize_keyboard

Optional. Requests clients to resize the keyboard.

Type
bool

one_time_keyboard

Optional. Requests clients to hide the keyboard as soon as it's been used.

Type
bool

selective

Optional. Show the keyboard to specific users only.

Type
bool

input_field_placeholder

Optional. The placeholder shown in the input field when the reply is active.

New in version 13.7.

Type
str

classmethod from_button(*button*, *resize_keyboard=False*, *one_time_keyboard=False*, *selective=False*, *input_field_placeholder=None*, ***kwargs*)

Shortcut for:

```
ReplyKeyboardMarkup([[button]], **kwargs)
```

Return a ReplyKeyboardMarkup from a single KeyboardButton.

Parameters

- ***button*** (*telegram.KeyboardButton* | str) – The button to use in the markup.

- ***resize_keyboard*** (bool, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to False, in which case the custom keyboard is always of the same height as the app's standard keyboard.
- ***one_time_keyboard*** (bool, 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.
- ***selective*** (bool, 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 Message object.
  2) If the bot's message is a reply (has `reply_to_message_id`), sender of the original message.

  Defaults to False.
- ***input_field_placeholder*** (str) – Optional. The placeholder shown in the input field when the reply is active.

  New in version 13.7.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

classmethod from_column(*button_column*, *resize_keyboard=False*, *one_time_keyboard=False*, *selective=False*, *input_field_placeholder=None*, ***kwargs*)

Shortcut for:

```
ReplyKeyboardMarkup([[button] for button in button_column], **kwargs)
```

Return a ReplyKeyboardMarkup from a single column of KeyboardButtons.

Parameters

- ***button_column*** (List[*telegram.KeyboardButton* | str]) – The button to use in the markup.
- ***resize_keyboard*** (bool, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to False, in which case the custom keyboard is always of the same height as the app's standard keyboard.
- ***one_time_keyboard*** (bool, 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.
- ***selective*** (bool, 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 Message object.
  2) If the bot's message is a reply (has `reply_to_message_id`), sender of the original message.

  Defaults to False.
- ***input_field_placeholder*** (str) – Optional. The placeholder shown in the input field when the reply is active.

  New in version 13.7.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

classmethod from_row(*button_row*, *resize_keyboard=False*, *one_time_keyboard=False*, *selective=False*, *input_field_placeholder=None*, ***kwargs*)

Shortcut for:

```
ReplyKeyboardMarkup([button_row], **kwargs)
```

Return a ReplyKeyboardMarkup from a single row of KeyboardButtons.

Parameters

- ***button_row*** (List[*telegram.KeyboardButton* | str]) – The button to use in the markup.
- ***resize_keyboard*** (bool, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to False, in which case the custom keyboard is always of the same height as the app's standard keyboard.
- ***one_time_keyboard*** (bool, 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.
- ***selective*** (bool, 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 Message object.
  2) If the bot's message is a reply (has reply_to_message_id), sender of the original message.

  Defaults to False.
- ***input_field_placeholder*** (str) – Optional. The placeholder shown in the input field when the reply is active.

  New in version 13.7.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.ReplyKeyboardRemove

class telegram.**ReplyKeyboardRemove**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

Upon receiving a message with this object, Telegram clients will remove the current custom keyboard and display the default letter-keyboard. By default, custom keyboards are displayed until a new keyboard is sent by a bot. An exception is made for one-time keyboards that are hidden immediately after the user presses a button (see *telegram.ReplyKeyboardMarkup*).

Example

A user votes in a poll, bot returns confirmation message in reply to the vote and removes the keyboard for that user, while still showing the keyboard with poll options to users who haven't voted yet.

Note: User will not be able to summon this keyboard; if you want to hide the keyboard from sight but keep it accessible, use *telegram.ReplyKeyboardMarkup.one_time_keyboard*.

Parameters

- ***selective*** (bool, 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 *telegram.Message* object.
  2) If the bot's message is a reply (has *reply_to_message_id*), sender of the original message.
- ****kwargs*** (dict) – Arbitrary keyword arguments.

remove_keyboard

Requests clients to remove the custom keyboard.

Type
True

selective

Optional. Use this parameter if you want to remove the keyboard for specific users only.

Type
bool

telegram.SentWebAppMessage

class telegram.**SentWebAppMessage**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

Contains information about an inline message sent by a Web App on behalf of a user.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *inline_message_id* are equal.

New in version 20.0.

Parameters
inline_message_id (str, optional) – Identifier of the sent inline message. Available only if there is an *inline keyboard* attached to the message.

inline_message_id

Optional. Identifier of the sent inline message. Available only if there is an *inline keyboard* attached to the message.

Type
str

telegram.TelegramObject

class telegram.**TelegramObject**(**args*, ***kwargs*)

Bases: object

Base class for most Telegram objects.

Objects of this type are subscriptable with strings, where `telegram_object[attribute_name]` is equivalent to `telegram_object.attribute_name`. If the object does not have an attribute with the appropriate name, a KeyError will be raised.

When objects of this type are pickled, the *Bot* attribute associated with the object will be removed. However, when copying the object via copy.deepcopy(), the copy will have the *same* bot instance associated with it, i.e:

```
assert telegram_object.get_bot() is copy.deepcopy(telegram_object).get_bot()
```

Changed in version 20.0: `telegram_object['from']` will look up the key `from_user`. This is to account for special cases like *Message.from_user* that deviate from the official Bot API.

classmethod de_json(*data*, *bot*)

Converts JSON data to a Telegram object.

Parameters

- ***data*** (Dict[str, ...]) – The JSON data.
- ***bot*** (*telegram.Bot*) – The bot associated with this object.

Returns

The Telegram object.

classmethod de_list(*data*, *bot*)

Converts JSON data to a list of Telegram objects.

Parameters

- ***data*** (Dict[str, ...]) – The JSON data.
- ***bot*** (*telegram.Bot*) – The bot associated with these objects.

Returns

A list of Telegram objects.

get_bot()

Returns the *telegram.Bot* instance associated with this object.

See also:

set_bot()

Raises

RuntimeError – If no *telegram.Bot* instance was set for this object.

set_bot(*bot*)

Sets the *telegram.Bot* instance associated with this object.

See also:

get_bot()

Parameters

bot (*telegram.Bot* | None) – The bot instance.

to_dict()

Gives representation of object as dict.

Returns

dict

to_json()

Gives a JSON representation of object.

Returns

str

telegram.Update

class telegram.**Update**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents an incoming update.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *update_id* is equal.

Note: At most one of the optional parameters can be present in any given update.

Parameters

- ***update_id*** (int) – The update's unique identifier. Update identifiers start from a certain positive number and increase sequentially. This ID becomes especially handy if you're using Webhooks, since it allows you to ignore repeated updates or to restore the correct update sequence, should they get out of order. If there are no new updates for at least a week, then identifier of the next update will be chosen randomly instead of sequentially.
- ***message*** (*telegram.Message*, optional) – New incoming message of any kind - text, photo, sticker, etc.
- ***edited_message*** (*telegram.Message*, optional) – New version of a message that is known to the bot and was edited.
- ***channel_post*** (*telegram.Message*, optional) – New incoming channel post of any kind - text, photo, sticker, etc.
- ***edited_channel_post*** (*telegram.Message*, optional) – New version of a channel post that is known to the bot and was edited.
- ***inline_query*** (*telegram.InlineQuery*, optional) – New incoming inline query.
- ***chosen_inline_result*** (*telegram.ChosenInlineResult*, optional) – The result of an inline query that was chosen by a user and sent to their chat partner.
- ***callback_query*** (*telegram.CallbackQuery*, optional) – New incoming callback query.
- ***shipping_query*** (*telegram.ShippingQuery*, optional) – New incoming shipping query. Only for invoices with flexible price.
- ***pre_checkout_query*** (*telegram.PreCheckoutQuery*, optional) – New incoming pre-checkout query. Contains full information about checkout.
- ***poll*** (*telegram.Poll*, optional) – New poll state. Bots receive only updates about stopped polls and polls, which are sent by the bot.
- ***poll_answer*** (*telegram.PollAnswer*, 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*** (*telegram.ChatMemberUpdated*, 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.

  New in version 13.4.
- ***chat_member*** (*telegram.ChatMemberUpdated*, 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 *telegram.ext.Application.run_polling.allowed_updates* to receive these updates (see *telegram.Bot.get_updates()*, *telegram.Bot.set_webhook()*, *telegram.ext.Application.run_polling()* and *telegram.ext.Application.run_webhook()*).

New in version 13.4.

- ***chat_join_request*** (*telegram.ChatJoinRequest*, optional) – A request to join the chat has been sent. The bot must have the *telegram.ChatPermissions.can_invite_users* administrator right in the chat to receive these updates.

  New in version 13.8.

- ****kwargs*** (dict) – Arbitrary keyword arguments.

update_id

The update's unique identifier.

Type
int

message

Optional. New incoming message.

Type
telegram.Message

edited_message

Optional. New version of a message.

Type
telegram.Message

channel_post

Optional. New incoming channel post.

Type
telegram.Message

edited_channel_post

Optional. New version of a channel post.

Type
telegram.Message

inline_query

Optional. New incoming inline query.

Type
telegram.InlineQuery

chosen_inline_result

Optional. The result of an inline query that was chosen by a user.

Type
telegram.ChosenInlineResult

callback_query

Optional. New incoming callback query.

Type
telegram.CallbackQuery

shipping_query

Optional. New incoming shipping query.

Type
telegram.ShippingQuery

pre_checkout_query

Optional. New incoming pre-checkout query.

Type
telegram.PreCheckoutQuery

poll

Optional. New poll state. Bots receive only updates about stopped polls and polls, which are sent by the bot.

Type
telegram.Poll

poll_answer

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.

Type
telegram.PollAnswer

my_chat_member

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.

New in version 13.4.

Type
telegram.ChatMemberUpdated

chat_member

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 *telegram.ext.Application.run_polling.allowed_updates* to receive these updates (see *telegram.Bot.get_updates()*, *telegram.Bot.set_webhook()*, *telegram.ext.Application.run_polling()* and *telegram.ext.Application.run_webhook()*).

New in version 13.4.

Type
telegram.ChatMemberUpdated

chat_join_request

Optional. A request to join the chat has been sent. The bot must have the *telegram.ChatPermissions.can_invite_users* administrator right in the chat to receive these updates.

New in version 13.8.

Type
telegram.ChatJoinRequest

ALL_TYPES = [<UpdateType.MESSAGE>, <UpdateType.EDITED_MESSAGE>, <UpdateType.CHANNEL_POST>, <UpdateType.EDITED_CHANNEL_POST>, <UpdateType.INLINE_QUERY>, <UpdateType.CHOSEN_INLINE_RESULT>, <UpdateType.CALLBACK_QUERY>, <UpdateType.SHIPPING_QUERY>, <UpdateType.PRE_CHECKOUT_QUERY>, <UpdateType.POLL>, <UpdateType.POLL_ANSWER>, <UpdateType.MY_CHAT_MEMBER>, <UpdateType.CHAT_MEMBER>, <UpdateType.CHAT_JOIN_REQUEST>]

A list of all available update types.

New in version 13.5.

Type
List[str]

CALLBACK_QUERY = 'callback_query'

telegram.constants.UpdateType.CALLBACK_QUERY

New in version 13.5.

CHANNEL_POST = 'channel_post'

telegram.constants.UpdateType.CHANNEL_POST

New in version 13.5.

CHAT_JOIN_REQUEST = 'chat_join_request'

telegram.constants.UpdateType.CHAT_JOIN_REQUEST

New in version 13.8.

CHAT_MEMBER = 'chat_member'

telegram.constants.UpdateType.CHAT_MEMBER

New in version 13.5.

CHOSEN_INLINE_RESULT = 'chosen_inline_result'

telegram.constants.UpdateType.CHOSEN_INLINE_RESULT

New in version 13.5.

EDITED_CHANNEL_POST = 'edited_channel_post'

telegram.constants.UpdateType.EDITED_CHANNEL_POST

New in version 13.5.

EDITED_MESSAGE = 'edited_message'

telegram.constants.UpdateType.EDITED_MESSAGE

New in version 13.5.

INLINE_QUERY = 'inline_query'

telegram.constants.UpdateType.INLINE_QUERY

New in version 13.5.

MESSAGE = 'message'

telegram.constants.UpdateType.MESSAGE

New in version 13.5.

MY_CHAT_MEMBER = 'my_chat_member'

telegram.constants.UpdateType.MY_CHAT_MEMBER

New in version 13.5.

POLL = 'poll'

telegram.constants.UpdateType.POLL

New in version 13.5.

POLL_ANSWER = 'poll_answer'

telegram.constants.UpdateType.POLL_ANSWER

New in version 13.5.

PRE_CHECKOUT_QUERY = 'pre_checkout_query'

telegram.constants.UpdateType.PRE_CHECKOUT_QUERY

New in version 13.5.

SHIPPING_QUERY = 'shipping_query'

telegram.constants.UpdateType.SHIPPING_QUERY

New in version 13.5.

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

property effective_chat

The chat that this update was sent in, no matter what kind of update this is. If no chat is associated with this update, this gives None. This is the case, if *inline_query*, *chosen_inline_result*, *callback_query* from inline messages, *shipping_query*, *pre_checkout_query*, *poll* or *poll_answer* is present.

Example

If *message* is present, this will give *telegram.Message.chat*.

Type
telegram.Chat

property effective_message

The message included in this update, no matter what kind of update this is. More precisely, this will be the message contained in *message*, *edited_message*, *channel_post*, *edited_channel_post* or *callback_query* (i.e. *telegram.CallbackQuery.message*) or None, if none of those are present.

Type
telegram.Message

property effective_user

The user that sent this update, no matter what kind of update this is. If no user is associated with this update, this gives None. This is the case if *channel_post*, *edited_channel_post* or *poll* is present.

Example

- If *message* is present, this will give *telegram.Message.from_user*.
- If *poll_answer* is present, this will give *telegram.PollAnswer.user*.

Type
telegram.User

telegram.User

class telegram.**User**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a Telegram user or bot.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *id* is equal.

Changed in version 20.0: The following are now keyword-only arguments in Bot methods: `location`, `filename`, `venue`, `contact`, `{read, write, connect, pool}_timeout` `api_kwargs`. Use a named argument for those, and notice that some positional arguments changed position as a result.

Parameters

- ***id*** (int) – Unique identifier for this user or bot.

- ***is_bot*** (bool) – True, if this user is a bot.
- ***first_name*** (str) – User's or bots first name.
- ***last_name*** (str, optional) – User's or bots last name.
- ***username*** (str, optional) – User's or bots username.
- ***language_code*** (str, optional) – IETF language tag of the user's language.
- ***can_join_groups*** (str, optional) – True, if the bot can be invited to groups. Returned only in *telegram.Bot.get_me* requests.
- ***can_read_all_group_messages*** (str, optional) – True, if privacy mode is disabled for the bot. Returned only in *telegram.Bot.get_me* requests.
- ***supports_inline_queries*** (str, optional) – True, if the bot supports inline queries. Returned only in *telegram.Bot.get_me* requests.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- ***is_premium*** (bool, optional) – True, if this user is a Telegram Premium user.

  New in version 20.0.
- ***added_to_attachment_menu*** (bool, optional) – True, if this user added the bot to the attachment menu.

  New in version 20.0.

id

Unique identifier for this user or bot.

Type
int

is_bot

True, if this user is a bot.

Type
bool

first_name

User's or bot's first name.

Type
str

last_name

Optional. User's or bot's last name.

Type
str

username

Optional. User's or bot's username.

Type
str

language_code

Optional. IETF language tag of the user's language.

Type
str

can_join_groups

Optional. True, if the bot can be invited to groups. Returned only in *telegram.Bot.get_me* requests.

Type
str

can_read_all_group_messages

Optional. True, if privacy mode is disabled for the bot. Returned only in *telegram.Bot.get_me* requests.

Type
str

supports_inline_queries

Optional. True, if the bot supports inline queries. Returned only in *telegram.Bot.get_me* requests.

Type
str

bot

Optional. The Bot to use for instance methods.

Type
telegram.Bot

is_premium

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

New in version 20.0.

Type
bool

added_to_attachment_menu

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

New in version 20.0.

Type
bool

async approve_join_request(*chat_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.approve_chat_join_request(user_id=update.effective_user.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.approve_chat_join_request()*.

New in version 13.8.

Returns
On success, True is returned.

Return type
bool

async copy_message(*chat_id*, *message_id*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *disable_notification=None*, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *reply_markup=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.copy_message(from_chat_id=update.effective_user.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.copy_message()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async decline_join_request(*chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.decline_chat_join_request(user_id=update.effective_user.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.decline_chat_join_request()*.

New in version 13.8.

Returns
On success, `True` is returned.

Return type
`bool`

property full_name

Convenience property. The user's *first_name*, followed by (if available) *last_name*.

Type
`str`

async get_menu_button(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.get_chat_menu_button(chat_id=update.effective_user.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.get_chat_menu_button()*.

See also:

set_menu_button()

New in version 20.0.

Returns
On success, the current menu button is returned.

Return type
telegram.MenuButton

async get_profile_photos(*offset=None*, *limit=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.get_user_profile_photos(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.get_user_profile_photos()*.

property link

Convenience property. If *username* is available, returns a t.me link of the user.

Type
str

mention_button(*name=None*)

Shortcut for:

```
InlineKeyboardButton(text=name, url=f"tg://user?id={update.effective_user.id}
↪")
```

New in version 13.9.

Parameters
name (str) – The name used as a link for the user. Defaults to *full_name*.

Returns
InlineButton with url set to the user mention

Return type
telegram.InlineKeyboardButton

mention_html(*name=None*)

Parameters
name (str) – The name used as a link for the user. Defaults to *full_name*.

Returns
The inline mention for the user as HTML.

Return type
str

mention_markdown(*name=None*)

Note: *'Markdown'* is a legacy mode, retained by Telegram for backward compatibility. You should use *mention_markdown_v2()* instead.

Parameters
name (str) – The name used as a link for the user. Defaults to *full_name*.

Returns
The inline mention for the user as markdown (version 1).

Return type
str

mention_markdown_v2(*name=None*)

Parameters
name (str) – The name used as a link for the user. Defaults to *full_name*.

Returns
The inline mention for the user as markdown (version 2).

Return type
str

property name

Convenience property. If available, returns the user's *username* prefixed with "@". If *username* is not available, returns *full_name*.

Type
str

async **pin_message**(*message_id*, *disable_notification=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.pin_chat_message(chat_id=update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.pin_chat_message()*.

Returns
On success, True is returned.

Return type
bool

async **send_action**(*action*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_chat_action*

async **send_animation**(*animation*, *duration=None*, *width=None*, *height=None*, *thumb=None*, *caption=None*, *parse_mode=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_animation(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_animation()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async **send_audio**(*audio*, *duration=None*, *performer=None*, *title=None*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *thumb=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_audio(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_audio()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async **send_chat_action**(*action*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_chat_action(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_chat_action()*.

Returns
On success.

Return type
True

async send_contact(*phone_number=None*, *first_name=None*, *last_name=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *vcard=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *contact=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_contact(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_contact()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_copy(*from_chat_id*, *message_id*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *disable_notification=None*, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *reply_markup=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.copy_message(chat_id=update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.copy_message()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_dice(*disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *emoji=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_dice(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_dice()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_document(*document*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *thumb=None*, *disable_content_type_detection=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_document(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_document()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_game(*game_short_name*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_game(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_game()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_invoice(*title*, *description*, *payload*, *provider_token*, *currency*, *prices*, *start_parameter=None*, *photo_url=None*, *photo_size=None*, *photo_width=None*, *photo_height=None*, *need_name=None*, *need_phone_number=None*, *need_email=None*, *need_shipping_address=None*, *is_flexible=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *provider_data=None*, *send_phone_number_to_provider=None*, *send_email_to_provider=None*, *allow_sending_without_reply=None*, *max_tip_amount=None*, *suggested_tip_amounts=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_invoice(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_invoice()*.

> **Warning:** As of API 5.2 *start_parameter* is an optional argument and therefore the order of the arguments had to be changed. Use keyword arguments to make sure that the arguments are passed correctly.

Changed in version 13.5: As of Bot API 5.2, the parameter *start_parameter* is optional.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_location(*latitude=None*, *longitude=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *live_period=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, *allow_sending_without_reply=None*, *protect_content=None*, *, *location=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_location(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_location()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_media_group(*media*, *disable_notification=None*, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *protect_content=None*, *, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_media_group(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_media_group()*.

Returns
] On success, instance representing the message posted.

Return type
List[*telegram.Message*

async send_message(*text*, *parse_mode=None*, *disable_web_page_preview=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *entities=None*, *protect_content=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_message(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_message()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_photo(*photo*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, *, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_photo(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_photo()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_poll(*question*, *options*, *is_anonymous=None*, *type=None*, *allows_multiple_answers=None*, *correct_option_id=None*, *is_closed=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *explanation=None*, *explanation_parse_mode=None*, *open_period=None*, *close_date=None*, *allow_sending_without_reply=None*, *explanation_entities=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_poll(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_poll()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_sticker(*sticker*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_sticker(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_sticker()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_venue(*latitude=None*, *longitude=None*, *title=None*, *address=None*, *foursquare_id=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *foursquare_type=None*, *google_place_id=None*, *google_place_type=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *venue=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_venue(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_venue()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_video(*video*, *duration=None*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *width=None*, *height=None*, *parse_mode=None*, *supports_streaming=None*, *thumb=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_video(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_video()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_video_note(*video_note*, *duration=None*, *length=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *thumb=None*, *allow_sending_without_reply=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_video_note(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_video_note()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_voice(*voice*, *duration=None*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, ***, *filename=None*, *read_timeout=None*, *write_timeout=20*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_voice(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.send_voice()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async set_menu_button(*menu_button=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.set_chat_menu_button(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.set_chat_menu_button()*.

See also:

get_menu_button()

New in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

async unpin_all_messages(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.unpin_all_chat_messages(chat_id=update.effective_user.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.unpin_all_chat_messages()*.

Returns
On success, `True` is returned.

Return type
`bool`

async unpin_message(*message_id=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.unpin_chat_message(chat_id=update.effective_user.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.unpin_chat_message()*.

Returns
On success, `True` is returned.

Return type
`bool`

telegram.UserProfilePhotos

class telegram.**UserProfilePhotos**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a user's profile pictures.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *total_count* and *photos* are equal.

Parameters

- ***total_count*** (`int`) – Total number of profile pictures the target user has.
- ***photos*** (List[List[*telegram.PhotoSize*]]) – Requested profile pictures (in up to 4 sizes each).

total_count

Total number of profile pictures.

Type
`int`

photos

Requested profile pictures.

Type
List[List[*telegram.PhotoSize*]]

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.Venue

class telegram.**Venue**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a venue.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *location* and *title* are equal.

Note: Foursquare details and Google Pace details are mutually exclusive. However, this behaviour is undocumented and might be changed by Telegram.

Parameters

- ***location*** (*telegram.Location*) – Venue location.
- ***title*** (str) – Name of the venue.
- ***address*** (str) – Address of the venue.
- ***foursquare_id*** (str, optional) – Foursquare identifier of the venue.
- ***foursquare_type*** (str, optional) – Foursquare type of the venue. (For example, "arts_entertainment/default", "arts_entertainment/aquarium" or "food/icecream".)
- ***google_place_id*** (str, optional) – Google Places identifier of the venue.
- ***google_place_type*** (str, optional) – Google Places type of the venue. (See supported types.)
- *****kwargs*** (dict) – Arbitrary keyword arguments.

location

Venue location.

Type
telegram.Location

title

Name of the venue.

Type
str

address

Address of the venue.

Type
str

foursquare_id

Optional. Foursquare identifier of the venue.

Type
str

foursquare_type

Optional. Foursquare type of the venue.

Type
str

google_place_id

Optional. Google Places identifier of the venue.

Type
str

google_place_type

Optional. Google Places type of the venue.

Type
str

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

telegram.Video

class telegram.**Video**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a video file.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *file_unique_id* is equal.

Parameters

- ***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.
- ***thumb*** (*telegram.PhotoSize*, optional) – Video thumbnail.
- ***file_name*** (str, optional) – Original filename as defined by sender.
- ***mime_type*** (str, optional) – MIME type of a file as defined by sender.
- ***file_size*** (int, optional) – File size in bytes.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

file_id

Identifier for this file.

Type
str

file_unique_id

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

width

Video width as defined by sender.

Type
int

height

Video height as defined by sender.

Type
int

duration

Duration of the video in seconds as defined by sender.

Type
int

thumb

Optional. Video thumbnail.

Type
telegram.PhotoSize

file_name

Optional. Original filename as defined by sender.

Type
str

mime_type

Optional. MIME type of a file as defined by sender.

Type
str

file_size

Optional. File size in bytes.

Type
int

bot

Optional. The Bot to use for instance methods.

Type
telegram.Bot

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

async get_file(***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Convenience wrapper over *telegram.Bot.get_file*

For the documentation of the arguments, please see *telegram.Bot.get_file()*.

Returns
telegram.File

Raises
telegram.error.TelegramError –

telegram.VideoChatEnded

class telegram.**VideoChatEnded**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *duration* are equal.

New in version 13.4.

Changed in version 20.0: This class was renamed from `VoiceChatEnded` in accordance to Bot API 6.0.

Parameters

- ***duration*** (`int`) – Voice chat duration in seconds.
- *****kwargs*** (`dict`) – Arbitrary keyword arguments.

duration

Voice chat duration in seconds.

Type
`int`

telegram.VideoChatParticipantsInvited

class telegram.**VideoChatParticipantsInvited**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *users* are equal.

New in version 13.4.

Changed in version 20.0: This class was renamed from `VoiceChatParticipantsInvited` in accordance to Bot API 6.0.

Parameters

- ***users*** (List[*telegram.User*]) – New members that were invited to the video chat.
- *****kwargs*** (`dict`) – Arbitrary keyword arguments.

users

New members that were invited to the video chat.

Type
List[*telegram.User*]

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.VideoChatScheduled

class telegram.**VideoChatScheduled**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *start_date* are equal.

Changed in version 20.0: This class was renamed from `VoiceChatScheduled` in accordance to Bot API 6.0.

Parameters

- ***start_date*** (datetime.datetime) – Point in time (Unix timestamp) when the video chat is supposed to be started by a chat administrator
- *****kwargs*** (dict) – Arbitrary keyword arguments.

start_date

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

Type
datetime.datetime

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.VideoChatStarted

class telegram.**VideoChatStarted**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a service message about a video chat started in the chat. Currently holds no information.

New in version 13.4.

Changed in version 20.0: This class was renamed from `VoiceChatStarted` in accordance to Bot API 6.0.

telegram.VideoNote

class telegram.**VideoNote**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *file_unique_id* is equal.

Parameters

- ***file_id*** (str) – Identifier for this file, which can be used to download or reuse the file.
- ***file_unique_id*** (str) – Unique identifier for this file, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.
- ***length*** (int) – Video width and height (diameter of the video message) as defined by sender.
- ***duration*** (int) – Duration of the video in seconds as defined by sender.

- ***thumb*** (*telegram.PhotoSize*, optional) – Video thumbnail.
- ***file_size*** (*int*, optional) – File size in bytes.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- *****kwargs*** (*dict*) – Arbitrary keyword arguments.

file_id

Identifier for this file.

Type
str

file_unique_id

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

length

Video width and height as defined by sender.

Type
int

duration

Duration of the video in seconds as defined by sender.

Type
int

thumb

Optional. Video thumbnail.

Type
telegram.PhotoSize

file_size

Optional. File size in bytes.

Type
int

bot

Optional. The Bot to use for instance methods.

Type
telegram.Bot

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

async get_file(***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Convenience wrapper over *telegram.Bot.get_file*

For the documentation of the arguments, please see *telegram.Bot.get_file()*.

Returns
telegram.File

Raises
telegram.error.TelegramError –

telegram.Voice

class telegram.**Voice**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a voice note.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *file_unique_id* is equal.

Parameters

- ***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, optional) – Duration of the audio in seconds as defined by sender.
- ***mime_type*** (str, optional) – MIME type of the file as defined by sender.
- ***file_size*** (int, optional) – File size in bytes.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

file_id

Identifier for this file.

Type
str

file_unique_id

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

duration

Duration of the audio in seconds as defined by sender.

Type
int

mime_type

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

Type
str

file_size

Optional. File size in bytes.

Type
int

bot

Optional. The Bot to use for instance methods.

Type
telegram.Bot

async get_file(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Convenience wrapper over *telegram.Bot.get_file*

For the documentation of the arguments, please see *telegram.Bot.get_file()*.

Returns
telegram.File

Raises
telegram.error.TelegramError –

telegram.WebAppData

class telegram.**WebAppData**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

Contains data sent from a Web App to the bot.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *data* and *button_text* are equal.

See also:

Webappbot Example

New in version 20.0.

Parameters

- ***data*** (str) – The data. Be aware that a bad client can send arbitrary data in this field.
- ***button_text*** (str) – Text of the *web_app* keyboard button, from which the Web App was opened.

data

The data. Be aware that a bad client can send arbitrary data in this field.

Type
str

button_text

Text of the *web_app* keyboard button, from which the Web App was opened.

> **Warning:** Be aware that a bad client can send

arbitrary data in this field.

Type
str

telegram.WebAppInfo

class telegram.**WebAppInfo**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object contains information about a Web App.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *url* are equal.

See also:

Webappbot Example

New in version 20.0.

Parameters

url (str) – An HTTPS URL of a Web App to be opened with additional data as specified in Initializing Web Apps.

url

An HTTPS URL of a Web App to be opened with additional data as specified in Initializing Web Apps.

Type

str

telegram.WebhookInfo

class telegram.**WebhookInfo**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a Telegram WebhookInfo.

Contains information about the current status of a webhook.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *url*, *has_custom_certificate*, *pending_update_count*, *ip_address*, *last_error_date*, *last_error_message*, *max_connections*, *allowed_updates* and *last_synchronization_error_date* are equal.

Changed in version 20.0: *last_synchronization_error_date* is considered as well when comparing objects of this type in terms of equality.

Parameters

- ***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, optional) – Currently used webhook IP address.
- ***last_error_date*** (int, optional) – Unix time for the most recent error that happened when trying to deliver an update via webhook.
- ***last_error_message*** (str, optional) – Error message in human-readable format for the most recent error that happened when trying to deliver an update via webhook.
- ***max_connections*** (int, optional) – Maximum allowed number of simultaneous HTTPS connections to the webhook for update delivery.
- ***allowed_updates*** (List[str], optional) – A list of update types the bot is subscribed to. Defaults to all update types, except *telegram.Update.chat_member*.
- ***last_synchronization_error_date*** (int, optional) – Unix time of the most recent error that happened when trying to synchronize available updates with Telegram datacenters.

  New in version 20.0.

url

Webhook URL.

Type

str

has_custom_certificate

If a custom certificate was provided for webhook.

Type
bool

pending_update_count

Number of updates awaiting delivery.

Type
int

ip_address

Optional. Currently used webhook IP address.

Type
str

last_error_date

Optional. Unix time for the most recent error that happened.

Type
int

last_error_message

Optional. Error message in human-readable format.

Type
str

max_connections

Optional. Maximum allowed number of simultaneous HTTPS connections.

Type
int

allowed_updates

Optional. A list of update types the bot is subscribed to. Defaults to all update types, except *telegram.Update.chat_member*.

Type
List[str]

last_synchronization_error_date

Optional. Unix time of the most recent error that happened when trying to synchronize available updates with Telegram datacenters.

New in version 20.0.

Type
int

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

Stickers

telegram.MaskPosition

class telegram.**MaskPosition**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *point*, *x_shift*, *y_shift* and, *scale* are equal.

Parameters

- ***point*** (str) – The part of the face relative to which the mask should be placed. One of *FOREHEAD*, *EYES*, *MOUTH*, or *CHIN*.
- ***x_shift*** (float) – Shift by X-axis measured in widths of the mask scaled to the face size, from left to right. For example, choosing `-1.0` will place mask just to the left of the default mask position.
- ***y_shift*** (float) – Shift by Y-axis measured in heights of the mask scaled to the face size, from top to bottom. For example, `1.0` will place the mask just below the default mask position.
- ***scale*** (float) – Mask scaling coefficient. For example, `2.0` means double size.

point

The part of the face relative to which the mask should be placed. One of *FOREHEAD*, *EYES*, *MOUTH*, or *CHIN*.

Type
str

x_shift

Shift by X-axis measured in widths of the mask scaled to the face size, from left to right.

Type
float

y_shift

Shift by Y-axis measured in heights of the mask scaled to the face size, from top to bottom.

Type
float

scale

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

Type
float

CHIN = 'chin'

telegram.constants.MaskPosition.CHIN

EYES = 'eyes'

telegram.constants.MaskPosition.EYES

FOREHEAD = 'forehead'

telegram.constants.MaskPosition.FOREHEAD

MOUTH = 'mouth'

telegram.constants.MaskPosition.MOUTH

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

telegram.Sticker

class telegram.**Sticker**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a sticker.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *file_unique_id* is equal.

Note: As of v13.11 *is_video* is a required argument and therefore the order of the arguments had to be changed. Use keyword arguments to make sure that the arguments are passed correctly.

Parameters

- ***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) – 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.

  New in version 13.11.
- ***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*.

  New in version 20.0.
- ***thumb*** (*telegram.PhotoSize*, optional) – Sticker thumbnail in the .WEBP or .JPG format.
- ***emoji*** (str, optional) – Emoji associated with the sticker
- ***set_name*** (str, optional) – Name of the sticker set to which the sticker belongs.
- ***mask_position*** (*telegram.MaskPosition*, optional) – For mask stickers, the position where the mask should be placed.
- ***file_size*** (int, optional) – File size in bytes.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- ***premium_animation*** (*telegram.File*, optional) – For premium regular stickers, premium animation for the sticker.

  New in version 20.0.
- ***custom_emoji*** (str, optional) – For custom emoji stickers, unique identifier of the custom emoji.

  New in version 20.0.
- ***_kwargs*** (dict) – Arbitrary keyword arguments.

file_id

Identifier for this file.

Type

str

file_unique_id

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

width

Sticker width.

Type
int

height

Sticker height.

Type
int

is_animated

True, if the sticker is animated.

Type
bool

is_video

True, if the sticker is a video sticker.

New in version 13.11.

Type
bool

type

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*.

New in version 20.0.

Type
str

thumb

Optional. Sticker thumbnail in the .WEBP or .JPG format.

Type
telegram.PhotoSize

emoji

Optional. Emoji associated with the sticker.

Type
str

set_name

Optional. Name of the sticker set to which the sticker belongs.

Type
str

mask_position

Optional. For mask stickers, the position where the mask should be placed.

Type
telegram.MaskPosition

file_size

Optional. File size in bytes.

Type
int

bot

Optional. The Bot to use for instance methods.

Type
telegram.Bot

premium_animation

Optional. For premium regular stickers, premium animation for the sticker.

New in version 20.0.

Type
telegram.File

custom_emoji

Optional. For custom emoji stickers, unique identifier of the custom emoji.

New in version 20.0.

Type
str

CUSTOM_EMOJI = 'custom_emoji'

telegram.constants.StickerType.CUSTOM_EMOJI

MASK = 'mask'

telegram.constants.StickerType.MASK

REGULAR = 'regular'

telegram.constants.StickerType.REGULAR

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

async get_file(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Convenience wrapper over *telegram.Bot.get_file*

For the documentation of the arguments, please see *telegram.Bot.get_file()*.

Returns
telegram.File

Raises
telegram.error.TelegramError –

telegram.StickerSet

class telegram.**StickerSet**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a sticker set.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *name* is equal.

Note: As of v13.11 *is_video* is a required argument and therefore the order of the arguments had to be changed. Use keyword arguments to make sure that the arguments are passed correctly.

Changed in version 20.0:: The parameter `contains_masks` has been removed. Use *sticker_type* instead.

Parameters

- ***name*** (str) – Sticker set name.
- ***title*** (str) – Sticker set title.
- ***is_animated*** (bool) – True, if the sticker set contains animated stickers.
- ***is_video*** (bool) – True, if the sticker set contains video stickers.

  New in version 13.11.
- ***stickers*** (List[*telegram.Sticker*]) – List of all set stickers.
- ***sticker_type*** (str) – Type of stickers in the set, currently one of *telegram.Sticker.REGULAR*, *telegram.Sticker.MASK*, *telegram.Sticker.CUSTOM_EMOJI*.

  New in version 20.0.
- ***thumb*** (*telegram.PhotoSize*, optional) – Sticker set thumbnail in the `.WEBP`, `.TGS`, or `.WEBM` format.

name

Sticker set name.

Type
str

title

Sticker set title.

Type
str

is_animated

True, if the sticker set contains animated stickers.

Type
bool

is_video

True, if the sticker set contains video stickers.

New in version 13.11.

Type
bool

stickers

List of all set stickers.

Type
List[*telegram.Sticker*]

sticker_type

Type of stickers in the set.

New in version 20.0.

Type
str

thumb

Optional. Sticker set thumbnail in the `.WEBP`, `.TGS` or `.WEBM` format.

Type
telegram.PhotoSize

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

to_dict()

See *telegram.TelegramObject.to_dict()*.

Inline Mode

telegram.ChosenInlineResult

class telegram.**ChosenInlineResult**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

Represents a result of an inline query that was chosen by the user and sent to their chat partner.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *result_id* is equal.

Note:

- In Python `from` is a reserved word use *from_user* instead.
- It is necessary to enable inline feedback via @Botfather in order to receive these objects in updates.

Parameters

- ***result_id*** (`str`) – The unique identifier for the result that was chosen.
- ***from_user*** (*telegram.User*) – The user that chose the result.
- ***location*** (*telegram.Location*, optional) – Sender location, only for bots that require user location.
- ***inline_message_id*** (`str`, 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.
- ***query*** (`str`) – The query that was used to obtain the result.
- *****kwargs*** (`dict`) – Arbitrary keyword arguments.

result_id

The unique identifier for the result that was chosen.

Type
`str`

from_user

The user that chose the result.

Type
telegram.User

location

Optional. Sender location.

Type
telegram.Location

inline_message_id

Optional. Identifier of the sent inline message.

Type
str

query

The query that was used to obtain the result.

Type
str

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

telegram.InlineQuery

class telegram.**InlineQuery**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *id* is equal.

Note: In Python `from` is a reserved word use *from_user* instead.

Changed in version 20.0:

- The following are now keyword-only arguments in Bot methods: `{read, write, connect, pool}_timeout`, *answer.api_kwargs*, `auto_pagination`. Use a named argument for those, and notice that some positional arguments changed position as a result.

Parameters

- ***id*** (str) – Unique identifier for this query.
- ***from_user*** (*telegram.User*) – Sender.
- ***query*** (str) – Text of the query (up to 256 characters).
- ***offset*** (str) – Offset of the results to be returned, can be controlled by the bot.
- ***chat_type*** (str, 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.

  New in version 13.5.
- ***location*** (*telegram.Location*, optional) – Sender location, only for bots that request user location.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

id

Unique identifier for this query.

Type
str

from_user

Sender.

Type
telegram.User

query

Text of the query (up to 256 characters).

Type
str

offset

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

Type
str

location

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

Type
telegram.Location

chat_type

Type of the chat, from which the inline query was sent.

New in version 13.5.

Type
str, optional

MAX_RESULTS = 50

telegram.constants.InlineQueryLimit.RESULTS

New in version 13.2.

MAX_SWITCH_PM_TEXT_LENGTH = 64

telegram.constants.InlineQueryLimit.SWITCH_PM_TEXT_LENGTH

New in version 20.0.

async answer(*results*, *cache_time=None*, *is_personal=None*, *next_offset=None*, *switch_pm_text=None*, *switch_pm_parameter=None*, ***, *current_offset=None*, *auto_pagination=False*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.answer_inline_query(
    update.inline_query.id,
    *args,
    current_offset=self.offset if auto_pagination else None,
    **kwargs
)
```

For the documentation of the arguments, please see *telegram.Bot.answer_inline_query()*.

Changed in version 20.0: Raises ValueError instead of TypeError.

Keyword Arguments
auto_pagination (bool, optional) – If set to True, *offset* will be passed as *current_offset* to *telegram.Bot.answer_inline_query()*. Defaults to False.

Raises
ValueError – If both *current_offset* and *auto_pagination* are supplied.

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

telegram.InlineQueryResult

class telegram.**InlineQueryResult**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

Baseclass for the InlineQueryResult* classes.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *id* is equal.

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

Parameters

- ***type*** (str) – Type of the result.
- ***id*** (str) – Unique identifier for this result, 1-64 Bytes.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

Type of the result.

Type
str

id

Unique identifier for this result, 1-64 Bytes.

Type
str

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.InlineQueryResultArticle

class telegram.**InlineQueryResultArticle**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

This object represents a Telegram InlineQueryResultArticle.

See also:

Inline Example

Parameters

- ***id*** (str) – Unique identifier for this result, 1-64 Bytes.

- ***title*** (str) – Title of the result.
- ***input_message_content*** (*telegram.InputMessageContent*) – Content of the message to be sent.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***url*** (str, optional) – URL of the result.
- ***hide_url*** (bool, optional) – Pass True, if you don't want the URL to be shown in the message.
- ***description*** (str, optional) – Short description of the result.
- ***thumb_url*** (str, optional) – Url of the thumbnail for the result.
- ***thumb_width*** (int, optional) – Thumbnail width.
- ***thumb_height*** (int, optional) – Thumbnail height.
- ****kwargs*** (dict) – Arbitrary keyword arguments.

type

'article'.

Type
str

id

Unique identifier for this result, 1-64 Bytes.

Type
str

title

Title of the result.

Type
str

input_message_content

Content of the message to be sent.

Type
telegram.InputMessageContent

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

url

Optional. URL of the result.

Type
str

hide_url

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

Type
bool

description

Optional. Short description of the result.

Type
str

thumb_url

Optional. Url of the thumbnail for the result.

Type
str

thumb_width

Optional. Thumbnail width.

Type
int

thumb_height

Optional. Thumbnail height.

Type
int

telegram.InlineQueryResultAudio

class telegram.**InlineQueryResultAudio**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

Represents a link to an mp3 audio file. By default, this audio file will be sent by the user. Alternatively, you can use *input_message_content* to send a message with the specified content instead of the audio.

Parameters

- ***id*** (str) – Unique identifier for this result, 1-64 bytes.
- ***audio_url*** (str) – A valid URL for the audio file.
- ***title*** (str) – Title.
- ***performer*** (str, optional) – Performer.
- ***audio_duration*** (str, optional) – Audio duration in seconds.
- ***caption*** (str, optional) – Caption, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***input_message_content*** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the audio.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'audio'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

audio_url

A valid URL for the audio file.

Type
str

title

Title.

Type
str

performer

Optional. Performer.

Type
str

audio_duration

Optional. Audio duration in seconds.

Type
str

caption

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

Type
str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.

Type
str

caption_entities

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

Type
List[*telegram.MessageEntity*]

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

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

Type
telegram.InputMessageContent

telegram.InlineQueryResultCachedAudio

class telegram.**InlineQueryResultCachedAudio**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

Represents a link to an mp3 audio file stored on the Telegram servers. By default, this audio file will be sent by the user. Alternatively, you can use *input_message_content* to send a message with the specified content instead of the audio.

Parameters

- ***id*** (str) – Unique identifier for this result, 1-64 bytes.
- ***audio_file_id*** (str) – A valid file identifier for the audio file.
- ***caption*** (str, optional) – Caption, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***input_message_content*** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the audio.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'audio'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

audio_file_id

A valid file identifier for the audio file.

Type
str

caption

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

Type
str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.

Type
str

caption_entities

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

Type
List[*telegram.MessageEntity*]

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

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

Type
telegram.InputMessageContent

telegram.InlineQueryResultCachedDocument

class telegram.**InlineQueryResultCachedDocument**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

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

Parameters

- ***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, optional) – Short description of the result.
- ***caption*** (str, optional) – Caption of the document to be sent, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption.. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***input_message_content*** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the file.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'document'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

title

Title for the result.

Type
str

document_file_id

A valid file identifier for the file.

Type
str

description

Optional. Short description of the result.

Type
str

caption

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

Type
str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption.. See the constants in *telegram.constants.ParseMode* for the available modes.

Type
str

caption_entities

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

Type
List[*telegram.MessageEntity*]

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

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

Type
telegram.InputMessageContent

telegram.InlineQueryResultCachedGif

class telegram.**InlineQueryResultCachedGif**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

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

Parameters

- ***id*** (str) – Unique identifier for this result, 1-64 bytes.
- ***gif_file_id*** (str) – A valid file identifier for the GIF file.
- ***title*** (str, optional) – Title for the result.caption (str, optional):
- ***caption*** (str, optional) – Caption of the GIF file to be sent, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***input_message_content*** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the gif.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'gif'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

gif_file_id

A valid file identifier for the GIF file.

Type
str

title

Optional. Title for the result.

Type
str

caption

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

Type
str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.

Type
str

caption_entities

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

Type
List[*telegram.MessageEntity*]

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

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

Type
telegram.InputMessageContent

telegram.InlineQueryResultCachedMpeg4Gif

class telegram.**InlineQueryResultCachedMpeg4Gif**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

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

Parameters

- ***id*** (str) – Unique identifier for this result, 1-64 bytes.
- ***mpeg4_file_id*** (str) – A valid file identifier for the MP4 file.
- ***title*** (str, optional) – Title for the result.
- ***caption*** (str, optional) – Caption of the MPEG-4 file to be sent, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***input_message_content*** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the MPEG-4 file.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'mpeg4_gif'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

mpeg4_file_id

A valid file identifier for the MP4 file.

Type
str

title

Optional. Title for the result.

Type
str

caption

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

Type
str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.

Type
str

caption_entities

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

Type
List[*telegram.MessageEntity*]

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

Optional. Content of the message to be sent instead of the MPEG-4 file.

Type
telegram.InputMessageContent

telegram.InlineQueryResultCachedPhoto

class telegram.**InlineQueryResultCachedPhoto**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

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

Parameters

- ***id*** (str) – Unique identifier for this result, 1-64 bytes.
- ***photo_file_id*** (str) – A valid file identifier of the photo.
- ***title*** (str, optional) – Title for the result.
- ***description*** (str, optional) – Short description of the result.
- ***caption*** (str, optional) – Caption of the photo to be sent, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***input_message_content*** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the photo.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'photo'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

photo_file_id

A valid file identifier of the photo.

Type
str

title

Optional. Title for the result.

Type
str

description

Optional. Short description of the result.

Type
str

caption

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

Type
str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.

Type
str

caption_entities

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

Type
List[*telegram.MessageEntity*]

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

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

Type
telegram.InputMessageContent

telegram.InlineQueryResultCachedSticker

class telegram.**InlineQueryResultCachedSticker**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

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

Parameters

- ***id*** (str) – Unique identifier for this result, 1-64 bytes.
- ***sticker_file_id*** (str) – A valid file identifier of the sticker.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***input_message_content*** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the sticker.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'sticker'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

sticker_file_id

A valid file identifier of the sticker.

Type
str

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

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

Type
telegram.InputMessageContent

telegram.InlineQueryResultCachedVideo

class telegram.**InlineQueryResultCachedVideo**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

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

Parameters

- ***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, optional) – Short description of the result.
- ***caption*** (str, optional) – Caption of the video to be sent, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***input_message_content*** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the video.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'video'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

video_file_id

A valid file identifier for the video file.

Type
str

title

Title for the result.

Type
str

description

Optional. Short description of the result.

Type
str

caption

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

Type
str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.

Type
str

caption_entities

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

Type
List[*telegram.MessageEntity*]

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

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

Type
telegram.InputMessageContent

telegram.InlineQueryResultCachedVoice

class telegram.**InlineQueryResultCachedVoice**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

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

Parameters

- ***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, optional) – Caption, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***input_message_content*** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the voice message.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'voice'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

voice_file_id

A valid file identifier for the voice message.

Type
str

title

Voice message title.

Type
str

caption

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

Type
str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.

Type
str

caption_entities

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

Type
List[*telegram.MessageEntity*]

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

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

Type
telegram.InputMessageContent

telegram.InlineQueryResultContact

class telegram.**InlineQueryResultContact**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

Represents a contact with a phone number. By default, this contact will be sent by the user. Alternatively, you can use *input_message_content* to send a message with the specified content instead of the contact.

Parameters

- ***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, optional) – Contact's last name.
- ***vcard*** (str, optional) – Additional data about the contact in the form of a vCard, 0-2048 bytes.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***input_message_content*** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the contact.
- ***thumb_url*** (str, optional) – Url of the thumbnail for the result.
- ***thumb_width*** (int, optional) – Thumbnail width.
- ***thumb_height*** (int, optional) – Thumbnail height.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'contact'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

phone_number

Contact's phone number.

Type
str

first_name

Contact's first name.

Type
str

last_name

Optional. Contact's last name.

Type
str

vcard

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

Type
str

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

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

Type
telegram.InputMessageContent

thumb_url

Optional. Url of the thumbnail for the result.

Type
str

thumb_width

Optional. Thumbnail width.

Type
int

thumb_height

Optional. Thumbnail height.

Type
int

telegram.InlineQueryResultDocument

class telegram.**InlineQueryResultDocument**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

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

Parameters

- ***id*** (str) – Unique identifier for this result, 1-64 bytes.
- ***title*** (str) – Title for the result.
- ***caption*** (str, optional) – Caption of the document to be sent, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- ***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".
- ***description*** (str, optional) – Short description of the result.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***input_message_content*** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the file.
- ***thumb_url*** (str, optional) – URL of the thumbnail (JPEG only) for the file.
- ***thumb_width*** (int, optional) – Thumbnail width.
- ***thumb_height*** (int, optional) – Thumbnail height.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'document'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

title

Title for the result.

Type
str

caption

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

Type
str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.

Type
str

caption_entities

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

Type
List[*telegram.MessageEntity*]

document_url

A valid URL for the file.

Type
str

mime_type

Mime type of the content of the file, either “application/pdf” or “application/zip”.

Type
str

description

Optional. Short description of the result.

Type
str

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

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

Type
telegram.InputMessageContent

thumb_url

Optional. URL of the thumbnail (JPEG only) for the file.

Type
str

thumb_width

Optional. Thumbnail width.

Type
int

thumb_height

Optional. Thumbnail height.

Type
int

telegram.InlineQueryResultGame

class telegram.**InlineQueryResultGame**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

Represents a *telegram.Game*.

Parameters

- ***id*** (str) – Unique identifier for this result, 1-64 bytes.
- ***game_short_name*** (str) – Short name of the game.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'game'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

game_short_name

Short name of the game.

Type
str

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

telegram.InlineQueryResultGif

class telegram.**InlineQueryResultGif**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

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

Parameters

- ***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.
- ***gif_width*** (int, optional) – Width of the GIF.
- ***gif_height*** (int, optional) – Height of the GIF.
- ***gif_duration*** (int, optional) – Duration of the GIF in seconds.
- ***thumb_url*** (str) – URL of the static (JPEG or GIF) or animated (MPEG4) thumbnail for the result.

- ***thumb_mime_type*** (str, optional) – MIME type of the thumbnail, must be one of `'image/jpeg'`, `'image/gif'`, or `'video/mp4'`. Defaults to `'image/jpeg'`.
- ***title*** (str, optional) – Title for the result.
- ***caption*** (str, optional) – Caption of the GIF file to be sent, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***input_message_content*** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the GIF animation.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'gif'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

gif_url

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

Type
str

gif_width

Optional. Width of the GIF.

Type
int

gif_height

Optional. Height of the GIF.

Type
int

gif_duration

Optional. Duration of the GIF in seconds.

Type
int

thumb_url

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

Type
str

thumb_mime_type

Optional. MIME type of the thumbnail.

Type
str

title

Optional. Title for the result.

Type
str

caption

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

Type
str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.

Type
str

caption_entities

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

Type
List[*telegram.MessageEntity*]

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

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

Type
telegram.InputMessageContent

telegram.InlineQueryResultLocation

class telegram.**InlineQueryResultLocation**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

Represents a location on a map. By default, the location will be sent by the user. Alternatively, you can use *input_message_content* to send a message with the specified content instead of the location.

Parameters

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

- ***live_period*** (int, optional) – Period in seconds for which the location can be updated, should be between 60 and 86400.
- ***heading*** (int, 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, optional) – For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and *360* if specified.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***input_message_content*** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the location.
- ***thumb_url*** (str, optional) – Url of the thumbnail for the result.
- ***thumb_width*** (int, optional) – Thumbnail width.
- ***thumb_height*** (int, optional) – Thumbnail height.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'location'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

latitude

Location latitude in degrees.

Type
float

longitude

Location longitude in degrees.

Type
float

title

Location title.

Type
str

horizontal_accuracy

Optional. The radius of uncertainty for the location, measured in meters.

Type
float

live_period

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

Type
int

heading

Optional. For live locations, a direction in which the user is moving, in degrees.

Type
int

proximity_alert_radius

Optional. For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters.

Type
int

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

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

Type
telegram.InputMessageContent

thumb_url

Optional. Url of the thumbnail for the result.

Type
str

thumb_width

Optional. Thumbnail width.

Type
int

thumb_height

Optional. Thumbnail height.

Type
int

telegram.InlineQueryResultMpeg4Gif

class telegram.**InlineQueryResultMpeg4Gif**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

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

Parameters

- ***id*** (str) – Unique identifier for this result, 1-64 bytes.
- ***mpeg4_url*** (str) – A valid URL for the MP4 file. File size must not exceed 1MB.
- ***mpeg4_width*** (int, optional) – Video width.
- ***mpeg4_height*** (int, optional) – Video height.
- ***mpeg4_duration*** (int, optional) – Video duration in seconds.
- ***thumb_url*** (str) – URL of the static thumbnail (jpeg or gif) for the result.

- ***thumb_mime_type*** (str) – Optional. MIME type of the thumbnail, must be one of 'image/jpeg', 'image/gif', or 'video/mp4'. Defaults to 'image/jpeg'.
- ***title*** (str, optional) – Title for the result.
- ***caption*** (str, optional) – Caption of the MPEG-4 file to be sent, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***input_message_content*** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the video animation.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'mpeg4_gif'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

mpeg4_url

A valid URL for the MP4 file. File size must not exceed 1MB.

Type
str

mpeg4_width

Optional. Video width.

Type
int

mpeg4_height

Optional. Video height.

Type
int

mpeg4_duration

Optional. Video duration in seconds.

Type
int

thumb_url

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

Type
str

thumb_mime_type

Optional. MIME type of the thumbnail.

Type
str

title

Optional. Title for the result.

Type
str

caption

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

Type
str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.

Type
str

caption_entities

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

Type
List[*telegram.MessageEntity*]

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

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

Type
telegram.InputMessageContent

telegram.InlineQueryResultPhoto

class telegram.**InlineQueryResultPhoto**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

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

Parameters

- ***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.
- ***thumb_url*** (str) – URL of the thumbnail for the photo.
- ***photo_width*** (int, optional) – Width of the photo.

- ***photo_height*** (int, optional) – Height of the photo.
- ***title*** (str, optional) – Title for the result.
- ***description*** (str, optional) – Short description of the result.
- ***caption*** (str, optional) – Caption of the photo to be sent, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***input_message_content*** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the photo.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'photo'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

photo_url

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

Type
str

thumb_url

URL of the thumbnail for the photo.

Type
str

photo_width

Optional. Width of the photo.

Type
int

photo_height

Optional. Height of the photo.

Type
int

title

Optional. Title for the result.

Type
str

description

Optional. Short description of the result.

Type
str

caption

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

Type
str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.

Type
str

caption_entities

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

Type
List[*telegram.MessageEntity*]

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

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

Type
telegram.InputMessageContent

telegram.InlineQueryResultVenue

class telegram.**InlineQueryResultVenue**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

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

Note: Foursquare details and Google Pace details are mutually exclusive. However, this behaviour is undocumented and might be changed by Telegram.

Parameters

- ***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, optional) – Foursquare identifier of the venue if known.

- ***foursquare_type*** (str, optional) – Foursquare type of the venue, if known. (For example, "arts_entertainment/default", "arts_entertainment/aquarium" or "food/icecream".)
- ***google_place_id*** (str, optional) – Google Places identifier of the venue.
- ***google_place_type*** (str, optional) – Google Places type of the venue. (See supported types.)
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***input_message_content*** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the location.
- ***thumb_url*** (str, optional) – Url of the thumbnail for the result.
- ***thumb_width*** (int, optional) – Thumbnail width.
- ***thumb_height*** (int, optional) – Thumbnail height.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'venue'.

Type
str

id

Unique identifier for this result, 1-64 Bytes.

Type
str

latitude

Latitude of the venue location in degrees.

Type
float

longitude

Longitude of the venue location in degrees.

Type
float

title

Title of the venue.

Type
str

address

Address of the venue.

Type
str

foursquare_id

Optional. Foursquare identifier of the venue if known.

Type
str

foursquare_type

Optional. Foursquare type of the venue, if known.

Type
str

google_place_id

Optional. Google Places identifier of the venue.

Type
str

google_place_type

Optional. Google Places type of the venue.

Type
str

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

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

Type
telegram.InputMessageContent

thumb_url

Optional. Url of the thumbnail for the result.

Type
str

thumb_width

Optional. Thumbnail width.

Type
int

thumb_height

Optional. Thumbnail height.

Type
int

telegram.InlineQueryResultVideo

class telegram.**InlineQueryResultVideo**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

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

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

Parameters

- **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 video url, “text/html” or “video/mp4”.
- **thumb_url** (str) – URL of the thumbnail (JPEG only) for the video.
- **title** (str) – Title for the result.
- **caption** (str, optional) – Caption, 0-*1024* characters after entities parsing.
- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- **caption_entities** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- **video_width** (int, optional) – Video width.
- **video_height** (int, optional) – Video height.
- **video_duration** (int, optional) – Video duration in seconds.
- **description** (str, optional) – Short description of the result.
- **reply_markup** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- **input_message_content** (*telegram.InputMessageContent*, 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).
- ****kwargs*** (dict) – Arbitrary keyword arguments.

type

'video'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

video_url

A valid URL for the embedded video player or video file.

Type
str

mime_type

Mime type of the content of video url, “text/html” or “video/mp4”.

Type
str

thumb_url

URL of the thumbnail (JPEG only) for the video.

Type
str

title

Title for the result.

Type
str

caption

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

Type
str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.

Type
str

caption_entities

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

Type
List[*telegram.MessageEntity*]

video_width

Optional. Video width.

Type
int

video_height

Optional. Video height.

Type
int

video_duration

Optional. Video duration in seconds.

Type
int

description

Optional. Short description of the result.

Type
str

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

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).

Type
telegram.InputMessageContent

telegram.InlineQueryResultVoice

class telegram.**InlineQueryResultVoice**(**args*, ***kwargs*)

Bases: *telegram.InlineQueryResult*

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

Parameters

- ***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, optional) – Caption, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- ***voice_duration*** (int, optional) – Recording duration in seconds.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- ***input_message_content*** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the voice recording.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

'voice'.

Type
str

id

Unique identifier for this result, 1-64 bytes.

Type
str

voice_url

A valid URL for the voice recording.

Type
str

title

Recording title.

Type
str

caption

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

Type
str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.constants.ParseMode* for the available modes.

Type
str

caption_entities

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

Type
List[*telegram.MessageEntity*]

voice_duration

Optional. Recording duration in seconds.

Type
int

reply_markup

Optional. Inline keyboard attached to the message.

Type
telegram.InlineKeyboardMarkup

input_message_content

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

Type
telegram.InputMessageContent

telegram.InputMessageContent

class telegram.**InputMessageContent**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

Base class for Telegram InputMessageContent Objects.

See: *telegram.InputContactMessageContent*, *telegram.InputInvoiceMessageContent*, *telegram.InputLocationMessageContent*, *telegram.InputTextMessageContent* and *telegram.InputVenueMessageContent* for more details.

telegram.InputTextMessageContent

class telegram.**InputTextMessageContent**(**args*, ***kwargs*)

Bases: *telegram.InputMessageContent*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *message_text* is equal.

See also:

Inline Example

Parameters

- ***message_text*** (str) – Text of the message to be sent, 1-*4096* characters after entities parsing.

- ***parse_mode*** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***entities*** (List[*telegram.MessageEntity*], optional) – List of special entities that appear in the caption, which can be specified instead of *parse_mode*.
- ***disable_web_page_preview*** (bool, optional) – Disables link previews for links in the sent message.
- ****kwargs*** (dict) – Arbitrary keyword arguments.

message_text

Text of the message to be sent, 1-*4096* characters after entities parsing.

Type
str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message. See the constants in *telegram.constants.ParseMode* for the available modes.

Type
str

entities

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

Type
List[*telegram.MessageEntity*]

disable_web_page_preview

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

Type
bool

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.InputLocationMessageContent

class telegram.**InputLocationMessageContent**(**args*, ***kwargs*)

Bases: *telegram.InputMessageContent*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *latitude* and *longitude* are equal.

Parameters

- ***latitude*** (float) – Latitude of the location in degrees.
- ***longitude*** (float) – Longitude of the location in degrees.
- ***horizontal_accuracy*** (float, optional) – The radius of uncertainty for the location, measured in meters; 0-*1500*.
- ***live_period*** (int, optional) – Period in seconds for which the location can be updated, should be between 60 and 86400.

- ***heading*** (int, 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, optional) – For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and *360* if specified.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

latitude

Latitude of the location in degrees.

Type
float

longitude

Longitude of the location in degrees.

Type
float

horizontal_accuracy

Optional. The radius of uncertainty for the location, measured in meters.

Type
float

live_period

Optional. Period in seconds for which the location can be updated.

Type
int

heading

Optional. For live locations, a direction in which the user is moving, in degrees.

Type
int

proximity_alert_radius

Optional. For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters.

Type
int

telegram.InputVenueMessageContent

class telegram.**InputVenueMessageContent**(**args*, ***kwargs*)

Bases: *telegram.InputMessageContent*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *latitude*, *longitude* and *title* are equal.

Note: Foursquare details and Google Pace details are mutually exclusive. However, this behaviour is undocumented and might be changed by Telegram.

Parameters

- ***latitude*** (float) – Latitude of the location in degrees.

- ***longitude*** (float) – Longitude of the location in degrees.
- ***title*** (str) – Name of the venue.
- ***address*** (str) – Address of the venue.
- ***foursquare_id*** (str, optional) – Foursquare identifier of the venue, if known.
- ***foursquare_type*** (str, optional) – Foursquare type of the venue, if known. (For example, "arts_entertainment/default", "arts_entertainment/aquarium" or "food/icecream".)
- ***google_place_id*** (str, optional) – Google Places identifier of the venue.
- ***google_place_type*** (str, optional) – Google Places type of the venue. (See supported types.)
- ****kwargs*** (dict) – Arbitrary keyword arguments.

latitude

Latitude of the location in degrees.

Type
float

longitude

Longitude of the location in degrees.

Type
float

title

Name of the venue.

Type
str

address

Address of the venue.

Type
str

foursquare_id

Optional. Foursquare identifier of the venue, if known.

Type
str

foursquare_type

Optional. Foursquare type of the venue, if known.

Type
str

google_place_id

Optional. Google Places identifier of the venue.

Type
str

google_place_type

Optional. Google Places type of the venue.

Type
str

telegram.InputContactMessageContent

class telegram.**InputContactMessageContent**(**args*, ***kwargs*)

Bases: *telegram.InputMessageContent*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *phone_number* is equal.

Parameters

- ***phone_number*** (str) – Contact's phone number.
- ***first_name*** (str) – Contact's first name.
- ***last_name*** (str, optional) – Contact's last name.
- ***vcard*** (str, optional) – Additional data about the contact in the form of a vCard, 0-2048 bytes.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

phone_number

Contact's phone number.

Type
str

first_name

Contact's first name.

Type
str

last_name

Optional. Contact's last name.

Type
str

vcard

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

Type
str

telegram.InputInvoiceMessageContent

class telegram.**InputInvoiceMessageContent**(**args*, ***kwargs*)

Bases: *telegram.InputMessageContent*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *title*, *description*, *payload*, *provider_token*, *currency* and *prices* are equal.

New in version 13.5.

Parameters

- ***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[*telegram.LabeledPrice*]) – Price breakdown, a list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.)
- ***max_tip_amount*** (int, 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], optional) – An 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, optional) – An 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, optional) – 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, optional) – Photo size.
- ***photo_width*** (int, optional) – Photo width.
- ***photo_height*** (int, optional) – Photo height.
- ***need_name*** (bool, optional) – Pass True, if you require the user's full name to complete the order.
- ***need_phone_number*** (bool, optional) – Pass True, if you require the user's phone number to complete the order
- ***need_email*** (bool, optional) – Pass True, if you require the user's email address to complete the order.
- ***need_shipping_address*** (bool, optional) – Pass True, if you require the user's shipping address to complete the order
- ***send_phone_number_to_provider*** (bool, optional) – Pass True, if user's phone number should be sent to provider.
- ***send_email_to_provider*** (bool, optional) – Pass True, if user's email address should be sent to provider.
- ***is_flexible*** (bool, optional) – Pass True, if the final price depends on the shipping method.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

title

Product name. *1- 32* characters.

Type
str

description

Product description. *1- 255* characters.

Type
str

payload

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

Type
str

provider_token

Payment provider token, obtained via @Botfather.

Type
str

currency

Three-letter ISO 4217 currency code, see more on currencies

Type
str

prices

Price breakdown, a list of components.

Type
List[*telegram.LabeledPrice*]

max_tip_amount

Optional. The maximum accepted amount for tips in the smallest units of the currency (integer, not float/double).

Type
int

suggested_tip_amounts

Optional. An array of suggested amounts of tip in the smallest units of the currency (integer, not float/double).

Type
List[int]

provider_data

Optional. An object for data about the invoice, which will be shared with the payment provider.

Type
str

photo_url

Optional. URL of the product photo for the invoice.

Type
str

photo_size

Optional. Photo size.

Type
int

photo_width

Optional. Photo width.

Type
int

photo_height

Optional. Photo height.

Type
int

need_name

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

Type
bool

need_phone_number

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

Type
bool

need_email

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

Type
bool

need_shipping_address

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

Type
bool

send_phone_number_to_provider

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

Type
bool

send_email_to_provider

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

Type
bool

is_flexible

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

Type
bool

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

to_dict()

See *telegram.TelegramObject.to_dict()*.

Payments

telegram.Invoice

class telegram.**Invoice**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object contains basic information about an invoice.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *title*, *description*, *start_parameter*, *currency* and *total_amount* are equal.

Parameters

- ***title*** (str) – Product name.
- ***description*** (str) – Product description.
- ***start_parameter*** (str) – Unique bot deep-linking parameter that can be used to generate this invoice.
- ***currency*** (str) – Three-letter ISO 4217 currency code.
- ***total_amount*** (int) – Total price in the smallest units of the currency (integer, not float/double). For example, for a price of US$ 1.45 pass `amount = 145`. See the `exp` parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- *****kwargs*** (dict) – Arbitrary keyword arguments.

title

Product name.

Type
str

description

Product description.

Type
str

start_parameter

Unique bot deep-linking parameter.

Type
str

currency

Three-letter ISO 4217 currency code.

Type
str

total_amount

Total price in the smallest units of the currency.

Type
int

MAX_DESCRIPTION_LENGTH = 255

telegram.constants.InvoiceLimit.MAX_DESCRIPTION_LENGTH

New in version 20.0.

MAX_PAYLOAD_LENGTH = 128

telegram.constants.InvoiceLimit.MAX_PAYLOAD_LENGTH

New in version 20.0.

MAX_TITLE_LENGTH = 32

telegram.constants.InvoiceLimit.MAX_TITLE_LENGTH

New in version 20.0.

MIN_DESCRIPTION_LENGTH = 1

telegram.constants.InvoiceLimit.MIN_DESCRIPTION_LENGTH

New in version 20.0.

MIN_PAYLOAD_LENGTH = 1

telegram.constants.InvoiceLimit.MIN_PAYLOAD_LENGTH

New in version 20.0.

MIN_TITLE_LENGTH = 1

telegram.constants.InvoiceLimit.MIN_TITLE_LENGTH

New in version 20.0.

telegram.LabeledPrice

class telegram.**LabeledPrice**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *label* and *amount* are equal.

See also:

Paymentbot Example

Parameters

- ***label*** (str) – Portion label.
- ***amount*** (int) – Price of the product in the smallest units of the currency (integer, not float/double). For example, for a price of US$ 1.45 pass `amount = 145`. See the `exp` parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- *****kwargs*** (dict) – Arbitrary keyword arguments.

label

Portion label.

Type
str

amount

Price of the product in the smallest units of the currency.

Type
int

telegram.OrderInfo

class telegram.**OrderInfo**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents information about an order.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *name*, *phone_number*, *email* and *shipping_address* are equal.

Parameters

- ***name*** (str, optional) – User name.
- ***phone_number*** (str, optional) – User's phone number.
- ***email*** (str, optional) – User email.
- ***shipping_address*** (*telegram.ShippingAddress*, optional) – User shipping address.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

name

Optional. User name.

Type
str

phone_number

Optional. User's phone number.

Type
str

email

Optional. User email.

Type
str

shipping_address

Optional. User shipping address.

Type
telegram.ShippingAddress

classmethod **de_json**(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

telegram.PreCheckoutQuery

class telegram.**PreCheckoutQuery**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object contains information about an incoming pre-checkout query.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *id* is equal.

Note: In Python from is a reserved word use *from_user* instead.

Parameters

- ***id*** (str) – Unique query identifier.
- ***from_user*** (*telegram.User*) – User who sent the query.
- ***currency*** (str) – Three-letter ISO 4217 currency code.
- ***total_amount*** (int) – Total price in the smallest units of the currency (integer, not float/double). For example, for a price of US$ 1.45 pass `amount = 145`. See the `exp` parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- ***invoice_payload*** (str) – Bot specified invoice payload.
- ***shipping_option_id*** (str, optional) – Identifier of the shipping option chosen by the user.
- ***order_info*** (*telegram.OrderInfo*, optional) – Order info provided by the user.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

id

Unique query identifier.

Type
str

from_user

User who sent the query.

Type
telegram.User

currency

Three-letter ISO 4217 currency code.

Type
str

total_amount

Total price in the smallest units of the currency.

Type
int

invoice_payload

Bot specified invoice payload.

Type
str

shipping_option_id

Optional. Identifier of the shipping option chosen by the user.

Type
str

order_info

Optional. Order info provided by the user.

Type
telegram.OrderInfo

bot

Optional. The Bot to use for instance methods.

Type
telegram.Bot

async answer(*ok*, *error_message=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.answer_pre_checkout_query(update.pre_checkout_query.id, *args,␣
↪**kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.answer_pre_checkout_query()*.

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

telegram.ShippingAddress

class telegram.**ShippingAddress**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a Telegram ShippingAddress.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *country_code*, *state*, *city*, *street_line1*, *street_line2* and *post_code* are equal.

Parameters

- ***country_code*** (str) – ISO 3166-1 alpha-2 country code.
- ***state*** (str) – State, if applicable.
- ***city*** (str) – City.
- ***street_line1*** (str) – First line for the address.
- ***street_line2*** (str) – Second line for the address.
- ***post_code*** (str) – Address post code.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

country_code

ISO 3166-1 alpha-2 country code.

Type
str

state

State, if applicable.

Type
str

city

City.

Type
str

street_line1

First line for the address.

Type
str

street_line2

Second line for the address.

Type
str

post_code

Address post code.

Type
str

telegram.ShippingOption

class telegram.**ShippingOption**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents one shipping option.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *id* is equal.

See also:

Paymentbot Example

Parameters

- ***id*** (str) – Shipping option identifier.
- ***title*** (str) – Option title.
- ***prices*** (List[*telegram.LabeledPrice*]) – List of price portions.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

id

Shipping option identifier.

Type
str

title

Option title.

Type
str

prices

List of price portions.

Type
List[*telegram.LabeledPrice*]

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.ShippingQuery

class telegram.**ShippingQuery**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object contains information about an incoming shipping query.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *id* is equal.

Note: In Python `from` is a reserved word use *from_user* instead.

Parameters

- ***id*** (`str`) – Unique query identifier.
- ***from_user*** (*telegram.User*) – User who sent the query.
- ***invoice_payload*** (`str`) – Bot specified invoice payload.
- ***shipping_address*** (*telegram.ShippingAddress*) – User specified shipping address.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- *****kwargs*** (`dict`) – Arbitrary keyword arguments.

id

Unique query identifier.

Type
`str`

from_user

User who sent the query.

Type
telegram.User

invoice_payload

Bot specified invoice payload.

Type
`str`

shipping_address

User specified shipping address.

Type
telegram.ShippingAddress

bot

Optional. The Bot to use for instance methods.

Type
telegram.Bot

async **answer**(*ok*, *shipping_options=None*, *error_message=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.answer_shipping_query(update.shipping_query.id, *args, **kwargs)
```

For the documentation of the arguments, please see *telegram.Bot.answer_shipping_query()*.

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

telegram.SuccessfulPayment

class telegram.**SuccessfulPayment**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object contains basic information about a successful payment.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *telegram_payment_charge_id* and *provider_payment_charge_id* are equal.

Parameters

- ***currency*** (str) – Three-letter ISO 4217 currency code.
- ***total_amount*** (int) – Total price in the smallest units of the currency (integer, not float/double). For example, for a price of US$ 1.45 pass `amount = 145`. See the `exp` parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- ***invoice_payload*** (str) – Bot specified invoice payload.
- ***shipping_option_id*** (str, optional) – Identifier of the shipping option chosen by the user.
- ***order_info*** (*telegram.OrderInfo*, optional) – Order info provided by the user.
- ***telegram_payment_charge_id*** (str) – Telegram payment identifier.
- ***provider_payment_charge_id*** (str) – Provider payment identifier.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

currency

Three-letter ISO 4217 currency code.

Type
str

total_amount

Total price in the smallest units of the currency.

Type
int

invoice_payload

Bot specified invoice payload.

Type
str

shipping_option_id

Optional. Identifier of the shipping option chosen by the user.

Type
str

order_info

Optional. Order info provided by the user.

Type
telegram.OrderInfo

telegram_payment_charge_id

Telegram payment identifier.

Type
str

provider_payment_charge_id

Provider payment identifier.

Type
str

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

Games

telegram.Callbackgame

class telegram.**CallbackGame**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

A placeholder, currently holds no information. Use BotFather to set up your game.

telegram.Game

class telegram.**Game**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a game. Use BotFather to create and edit games, their short names will act as unique identifiers.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *title*, *description* and *photo* are equal.

Parameters

- ***title*** (str) – Title of the game.
- ***description*** (str) – Description of the game.
- ***photo*** (List[*telegram.PhotoSize*]) – Photo that will be displayed in the game message in chats.
- ***text*** (str, 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 *telegram.Bot.set_game_score()*, or manually edited using *telegram.Bot.edit_message_text()*. 0-*4096* characters.
- ***text_entities*** (List[*telegram.MessageEntity*], optional) – Special entities that appear in text, such as usernames, URLs, bot commands, etc.
- ***animation*** (*telegram.Animation*, optional) – Animation that will be displayed in the game message in chats. Upload via BotFather.

title

Title of the game.

Type
str

description

Description of the game.

Type
str

photo

Photo that will be displayed in the game message in chats.

Type
List[*telegram.PhotoSize*]

text

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 *telegram.Bot.set_game_score()*, or manually edited using *telegram.Bot.edit_message_text()*.

Type
str

text_entities

Special entities that appear in text, such as usernames, URLs, bot commands, etc. This list is empty if the message does not contain text entities.

Type
List[*telegram.MessageEntity*]

animation

Optional. Animation that will be displayed in the game message in chats. Upload via BotFather.

Type
telegram.Animation

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

parse_text_entities(*types=None*)

Returns a dict that maps *telegram.MessageEntity* to str. It contains entities from this message filtered by their *type* attribute as the key, and the text that each entity belongs to as the value of the dict.

Note: This method should always be used instead of the *text_entities* attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See *parse_text_entity* for more info.

Parameters
types (List[str], optional) – List of *telegram.MessageEntity* types as strings. If the *type* attribute of an entity is contained in this list, it will be returned. Defaults to *telegram.MessageEntity.ALL_TYPES*.

Returns
A dictionary of entities mapped to the text that belongs to them, calculated based on UTF-16 codepoints.

Return type
Dict[*telegram.MessageEntity*, str]

parse_text_entity(*entity*)

Returns the text from a given *telegram.MessageEntity*.

Note: This method is present because Telegram calculates the offset and length in UTF-16 codepoint pairs, which some versions of Python don't handle automatically. (That is, you can't just slice `Message.text` with the offset and length.)

Parameters
entity (*telegram.MessageEntity*) – The entity to extract the text from. It must be an entity that belongs to this message.

Returns
The text of the given entity.

Return type
str

Raises
RuntimeError – If this game has no text.

`to_dict()`
See *telegram.TelegramObject.to_dict()*.

telegram.GameHighScore

`class telegram.GameHighScore(*args, **kwargs)`
Bases: *telegram.TelegramObject*

This object represents one row of the high scores table for a game.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *position*, *user* and *score* are equal.

Parameters
- ***position*** (int) – Position in high score table for the game.
- ***user*** (*telegram.User*) – User.
- ***score*** (int) – Score.

`position`
Position in high score table for the game.

Type
int

`user`
User.

Type
telegram.User

`score`
Score.

Type
int

`classmethod de_json(data, bot)`
See *telegram.TelegramObject.de_json()*.

Passport

telegram.Credentials

class telegram.**Credentials**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

secure_data

Credentials for encrypted data

Type
telegram.SecureData

nonce

Bot-specified nonce

Type
str

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

telegram.DataCredentials

class telegram.**DataCredentials**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

These credentials can be used to decrypt encrypted data from the data field in EncryptedPassportData.

Parameters

- ***data_hash*** (str) – Checksum of encrypted data
- ***secret*** (str) – Secret of encrypted data

hash

Checksum of encrypted data

Type
str

secret

Secret of encrypted data

Type
str

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.EncryptedCredentials

class telegram.**EncryptedCredentials**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

Contains data required for decrypting and authenticating EncryptedPassportElement. See the Telegram Passport Documentation for a complete description of the data decryption and authentication processes.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *data*, *hash* and *secret* are equal.

Note: This object is decrypted only when originating from *telegram.PassportData.decrypted_credentials*.

Parameters

- ***data*** (*telegram.Credentials* or str) – Decrypted data with unique user's nonce, data hashes and secrets used for EncryptedPassportElement decryption and authentication or base64 encrypted data.
- ***hash*** (str) – Base64-encoded data hash for data authentication.
- ***secret*** (str) – Decrypted or encrypted secret used for decryption.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

data

Decrypted data with unique user's nonce, data hashes and secrets used for EncryptedPassportElement decryption and authentication or base64 encrypted data.

Type
telegram.Credentials or str

hash

Base64-encoded data hash for data authentication.

Type
str

secret

Decrypted or encrypted secret used for decryption.

Type
str

property decrypted_data

Lazily decrypt and return credentials data. This object also contains the user specified nonce as *decrypted_data.nonce*.

Raises
telegram.error.PassportDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type
telegram.Credentials

property decrypted_secret

Lazily decrypt and return secret.

Raises
telegram.error.PassportDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type
str

telegram.EncryptedPassportElement

class telegram.**EncryptedPassportElement**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

Contains information about documents or other Telegram Passport elements shared with the bot by the user. The data has been automatically decrypted by python-telegram-bot.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *type*, *data*, *phone_number*, *email*, *files*, *front_side*, *reverse_side* and *selfie* are equal.

Note: This object is decrypted only when originating from *telegram.PassportData.decrypted_data*.

Parameters

- ***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 *telegram.PassportElementErrorUnspecified*.
- ***data*** (*telegram.PersonalDetails* | *telegram.IdDocumentData* | *telegram.ResidentialAddress* | str, optional) – Decrypted or encrypted data, available for “personal_details”, “passport”, “driver_license”, “identity_card”, “identity_passport” and “address” types.
- ***phone_number*** (str, optional) – User’s verified phone number, available only for “phone_number” type.
- ***email*** (str, optional) – User’s verified email address, available only for “email” type.
- ***files*** (List[*telegram.PassportFile*], optional) – Array of encrypted/decrypted files with documents provided by the user, available for “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.
- ***front_side*** (*telegram.PassportFile*, optional) – Encrypted/decrypted file with the front side of the document, provided by the user. Available for “passport”, “driver_license”, “identity_card” and “internal_passport”.
- ***reverse_side*** (*telegram.PassportFile*, optional) – Encrypted/decrypted file with the reverse side of the document, provided by the user. Available for “driver_license” and “identity_card”.
- ***selfie*** (*telegram.PassportFile*, optional) – Encrypted/decrypted file with the selfie of the user holding a document, provided by the user; available for “passport”, “driver_license”, “identity_card” and “internal_passport”.
- ***translation*** (List[*telegram.PassportFile*], optional) – Array of encrypted/decrypted 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.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

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”.

Type
str

hash

Base64-encoded element hash for using in *telegram.PassportElementErrorUnspecified*.

Type
str

data

Optional. Decrypted or encrypted data, available for “personal_details”, “passport”, “driver_license”, “identity_card”, “identity_passport” and “address” types.

Type
telegram.PersonalDetails | *telegram.IdDocumentData* | *telegram.ResidentialAddress* | str

phone_number

Optional. User’s verified phone number, available only for “phone_number” type.

Type
str

email

Optional. User’s verified email address, available only for “email” type.

Type
str

files

Optional. Array of encrypted/decrypted files with documents provided by the user, available for “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.

Type
List[*telegram.PassportFile*]

front_side

Optional. Encrypted/decrypted file with the front side of the document, provided by the user. Available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

Type
telegram.PassportFile

reverse_side

Optional. Encrypted/decrypted file with the reverse side of the document, provided by the user. Available for “driver_license” and “identity_card”.

Type
telegram.PassportFile

selfie

Optional. Encrypted/decrypted file with the selfie of the user holding a document, provided by the user; available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

Type
telegram.PassportFile

translation

Optional. Array of encrypted/decrypted 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.

Type
List[*telegram.PassportFile*]

bot

Optional. The Bot to use for instance methods.

Type
telegram.Bot

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

classmethod de_json_decrypted(*data*, *bot*, *credentials*)

Variant of *telegram.TelegramObject.de_json()* that also takes into account passport credentials.

Parameters

- ***data*** (Dict[str, ...]) – The JSON data.
- ***bot*** (*telegram.Bot*) – The bot associated with this object.
- ***credentials*** (*telegram.FileCredentials*) – The credentials

Return type
telegram.EncryptedPassportElement

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.FileCredentials

class telegram.**FileCredentials**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

These credentials can be used to decrypt encrypted files from the front_side, reverse_side, selfie and files fields in EncryptedPassportData.

Parameters

- ***file_hash*** (str) – Checksum of encrypted file
- ***secret*** (str) – Secret of encrypted file

hash

Checksum of encrypted file

Type
str

secret

Secret of encrypted file

Type
str

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.IdDocumentData

class telegram.**IdDocumentData**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents the data of an identity document.

document_no

Document number.

Type
str

expiry_date

Optional. Date of expiry, in DD.MM.YYYY format.

Type
str

telegram.PassportData

class telegram.**PassportData**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

Contains information about Telegram Passport data shared with the bot by the user.

Note: To be able to decrypt this object, you must pass your `private_key` to either *telegram.ext.Updater* or *telegram.Bot*. Decrypted data is then found in *decrypted_data* and the payload can be found in *decrypted_credentials*'s attribute *telegram.Credentials.nonce*.

Parameters

- ***data*** (List[*telegram.EncryptedPassportElement*]) – Array with encrypted information about documents and other Telegram Passport elements that was shared with the bot.
- ***credentials*** (*telegram.EncryptedCredentials*)) – Encrypted credentials.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

data

Array with encrypted information about documents and other Telegram Passport elements that was shared with the bot.

Type
List[*telegram.EncryptedPassportElement*]

credentials

Encrypted credentials.

Type
telegram.EncryptedCredentials

bot

The Bot to use for instance methods.

Type
telegram.Bot, optional

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

property decrypted_credentials

Lazily decrypt and return credentials that were used
to decrypt the data. This object also contains the user specified payload as *decrypted_data.payload*.

Raises
telegram.error.PassportDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type
telegram.Credentials

property decrypted_data

Lazily decrypt and return information
about documents and other Telegram Passport elements which were shared with the bot.

Raises
telegram.error.PassportDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type
List[*telegram.EncryptedPassportElement*]

to_dict()

See *telegram.TelegramObject.to_dict()*.

telegram.PassportElementError

class telegram.**PassportElementError**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

Baseclass for the PassportElementError* classes.

This object represents an error in the Telegram Passport element which was submitted that should be resolved by the user.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *source* and *type* are equal.

Parameters

- ***source*** (str) – Error source.
- ***type*** (str) – The section of the user's Telegram Passport which has the error.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

source

Error source.

Type
str

type

The section of the user's Telegram Passport which has the error.

Type
str

message

Error message.

Type
str

telegram.PassportElementErrorDataField

class telegram.**PassportElementErrorDataField**(**args*, ***kwargs*)

Bases: *telegram.PassportElementError*

Represents an issue in one of the data fields that was provided by the user. The error is considered resolved when the field's value changes.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *source*, *type*, *field_name*, *data_hash* and *message* are equal.

Parameters

- ***type*** (str) – The section of the user's Telegram Passport which has the error, one of "personal_details", "passport", "driver_license", "identity_card", "internal_passport", "address".
- ***field_name*** (str) – Name of the data field which has the error.
- ***data_hash*** (str) – Base64-encoded data hash.
- ***message*** (str) – Error message.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

The section of the user's Telegram Passport which has the error, one of "personal_details", "passport", "driver_license", "identity_card", "internal_passport", "address".

Type
str

field_name

Name of the data field which has the error.

Type
str

data_hash

Base64-encoded data hash.

Type
str

message

Error message.

Type
str

telegram.PassportElementErrorFile

class telegram.**PassportElementErrorFile**(**args*, ***kwargs*)

Bases: *telegram.PassportElementError*

Represents an issue with a document scan. The error is considered resolved when the file with the document scan changes.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *source*, *type*, *file_hash*, and *message* are equal.

Parameters

- ***type*** (str) – The section of the user's Telegram Passport which has the issue, one of "utility_bill", "bank_statement", "rental_agreement", "passport_registration", "temporary_registration".
- ***file_hash*** (str) – Base64-encoded file hash.
- ***message*** (str) – Error message.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

The section of the user's Telegram Passport which has the issue, one of "utility_bill", "bank_statement", "rental_agreement", "passport_registration", "temporary_registration".

Type
str

file_hash

Base64-encoded file hash.

Type
str

message

Error message.

Type
str

telegram.PassportElementErrorFiles

class telegram.**PassportElementErrorFiles**(**args*, ***kwargs*)

Bases: *telegram.PassportElementError*

Represents an issue with a list of scans. The error is considered resolved when the list of files with the document scans changes.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *source*, *type*, *file_hashes*, and *message* are equal.

Parameters

- ***type*** (str) – The section of the user's Telegram Passport which has the issue, one of "utility_bill", "bank_statement", "rental_agreement", "passport_registration", "temporary_registration".
- ***file_hashes*** (List[str]) – List of base64-encoded file hashes.
- ***message*** (str) – Error message.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

The section of the user's Telegram Passport which has the issue, one of "utility_bill", "bank_statement", "rental_agreement", "passport_registration", "temporary_registration".

Type
str

file_hashes

List of base64-encoded file hashes.

Type
List[str]

message

Error message.

Type
str

telegram.PassportElementErrorFrontSide

class telegram.**PassportElementErrorFrontSide**(**args*, ***kwargs*)

Bases: *telegram.PassportElementError*

Represents an issue with the front side of a document. The error is considered resolved when the file with the front side of the document changes.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *source*, *type*, *file_hash*, and *message* are equal.

Parameters

- ***type*** (str) – The section of the user's Telegram Passport which has the issue, one of "passport", "driver_license", "identity_card", "internal_passport".
- ***file_hash*** (str) – Base64-encoded hash of the file with the front side of the document.
- ***message*** (str) – Error message.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

The section of the user's Telegram Passport which has the issue, one of "passport", "driver_license", "identity_card", "internal_passport".

Type
str

file_hash

Base64-encoded hash of the file with the front side of the document.

Type
str

message

Error message.

Type
str

telegram.PassportElementErrorReverseSide

class telegram.**PassportElementErrorReverseSide**(**args*, ***kwargs*)

Bases: *telegram.PassportElementError*

Represents an issue with the reverse side of a document. The error is considered resolved when the file with the reverse side of the document changes.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *source*, *type*, *file_hash*, and *message* are equal.

Parameters

- ***type*** (str) – The section of the user's Telegram Passport which has the issue, one of "driver_license", "identity_card".
- ***file_hash*** (str) – Base64-encoded hash of the file with the reverse side of the document.
- ***message*** (str) – Error message.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

The section of the user's Telegram Passport which has the issue, one of "driver_license", "identity_card".

Type
str

file_hash

Base64-encoded hash of the file with the reverse side of the document.

Type
str

message

Error message.

Type
str

telegram.PassportElementErrorSelfie

class telegram.**PassportElementErrorSelfie**(**args*, ***kwargs*)

Bases: *telegram.PassportElementError*

Represents an issue with the selfie with a document. The error is considered resolved when the file with the selfie changes.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *source*, *type*, *file_hash*, and *message* are equal.

Parameters

- ***type*** (str) – The section of the user's Telegram Passport which has the issue, one of "passport", "driver_license", "identity_card", "internal_passport".
- ***file_hash*** (str) – Base64-encoded hash of the file with the selfie.
- ***message*** (str) – Error message.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

The section of the user's Telegram Passport which has the issue, one of "passport", "driver_license", "identity_card", "internal_passport".

Type
str

file_hash

Base64-encoded hash of the file with the selfie.

Type
str

message

Error message.

Type
str

telegram.PassportElementErrorTranslationFile

class telegram.**PassportElementErrorTranslationFile**(**args*, ***kwargs*)

Bases: *telegram.PassportElementError*

Represents an issue with one of the files that constitute the translation of a document. The error is considered resolved when the file changes.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *source*, *type*, *file_hash*, and *message* are equal.

Parameters

- ***type*** (str) – Type of element of the user's Telegram Passport which has the issue, one of "passport", "driver_license", "identity_card", "internal_passport", "utility_bill", "bank_statement", "rental_agreement", "passport_registration", "temporary_registration".
- ***file_hash*** (str) – Base64-encoded hash of the file.
- ***message*** (str) – Error message.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

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".

Type
str

file_hash

Base64-encoded hash of the file.

Type
str

message

Error message.

Type
str

telegram.PassportElementErrorTranslationFiles

class telegram.**PassportElementErrorTranslationFiles**(**args*, ***kwargs*)

Bases: *telegram.PassportElementError*

Represents an issue with the translated version of a document. The error is considered resolved when a file with the document translation changes.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *source*, *type*, *file_hashes*, and *message* are equal.

Parameters

- ***type*** (str) – Type of element of the user's Telegram Passport which has the issue, one of "passport", "driver_license", "identity_card", "internal_passport", "utility_bill", "bank_statement", "rental_agreement", "passport_registration", "temporary_registration".
- ***file_hashes*** (List[str]) – List of base64-encoded file hashes.
- ***message*** (str) – Error message.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

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".

Type
str

file_hashes

List of base64-encoded file hashes.

Type
List[str]

message

Error message.

Type
str

telegram.PassportElementErrorUnspecified

class telegram.**PassportElementErrorUnspecified**(**args*, ***kwargs*)

Bases: *telegram.PassportElementError*

Represents an issue in an unspecified place. The error is considered resolved when new data is added.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *source*, *type*, *element_hash*, and *message* are equal.

Parameters

- ***type*** (str) – Type of element of the user's Telegram Passport which has the issue.
- ***element_hash*** (str) – Base64-encoded element hash.
- ***message*** (str) – Error message.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

type

Type of element of the user's Telegram Passport which has the issue.

Type
str

element_hash

Base64-encoded element hash.

Type
str

message

Error message.

Type
str

telegram.PassportFile

class telegram.**PassportFile**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a file uploaded to Telegram Passport. Currently all Telegram Passport files are in JPEG format when decrypted and don't exceed 10MB.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *file_unique_id* is equal.

Parameters

- ***file_id*** (str) – Identifier for this file, which can be used to download or reuse the file.
- ***file_unique_id*** (str) – Unique identifier for this file, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.
- ***file_size*** (int) – File size in bytes.
- ***file_date*** (int) – Unix time when the file was uploaded.
- ***bot*** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- *****kwargs*** (dict) – Arbitrary keyword arguments.

file_id

Identifier for this file.

Type
str

file_unique_id

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

file_size

File size in bytes.

Type
int

file_date

Unix time when the file was uploaded.

Type
int

bot

Optional. The Bot to use for instance methods.

Type
telegram.Bot

classmethod de_json_decrypted(*data*, *bot*, *credentials*)

Variant of *telegram.TelegramObject.de_json()* that also takes into account passport credentials.

Parameters

- ***data*** (Dict[str, ...]) – The JSON data.
- ***bot*** (*telegram.Bot*) – The bot associated with this object.
- ***credentials*** (*telegram.FileCredentials*) – The credentials

Return type
telegram.PassportFile

classmethod de_list_decrypted(*data*, *bot*, *credentials*)

Variant of *telegram.TelegramObject.de_list()* that also takes into account passport credentials.

Parameters

- ***data*** (Dict[str, ...]) – The JSON data.
- ***bot*** (*telegram.Bot*) – The bot associated with these objects.
- ***credentials*** (*telegram.FileCredentials*) – The credentials

Return type
List[*telegram.PassportFile*]

async get_file(**, read_timeout=None, write_timeout=None, connect_timeout=None, pool_timeout=None, api_kwargs=None*)

Wrapper over *telegram.Bot.get_file*. Will automatically assign the correct credentials to the returned *telegram.File* if originating from *telegram.PassportData.decrypted_data*.

For the documentation of the arguments, please see *telegram.Bot.get_file()*.

Returns
telegram.File

Raises
telegram.error.TelegramError –

telegram.PersonalDetails

class telegram.**PersonalDetails**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents personal details.

first_name

First Name.

Type
str

middle_name

Optional. First Name.

Type
str

last_name

Last Name.

Type
str

birth_date

Date of birth in DD.MM.YYYY format.

Type
str

gender

Gender, male or female.

Type
str

country_code

Citizenship (ISO 3166-1 alpha-2 country code).

Type
str

residence_country_code

Country of residence (ISO 3166-1 alpha-2 country code).

Type
str

first_name_native

First Name in the language of the user's country of residence.

Type
str

middle_name_native

Optional. Middle Name in the language of the user's country of residence.

Type
str

last_name_native

Last Name in the language of the user's country of residence.

Type
str

telegram.ResidentialAddress

class telegram.**ResidentialAddress**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents a residential address.

street_line1

First line for the address.

Type
str

street_line2

Optional. Second line for the address.

Type
str

city

City.

Type
str

state

Optional. State.

Type
str

country_code

ISO 3166-1 alpha-2 country code.

Type
str

post_code

Address post code.

Type
str

telegram.SecureData

class telegram.**SecureData**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents the credentials that were used to decrypt the encrypted data. All fields are optional and depend on fields that were requested.

personal_details

Credentials for encrypted personal details.

Type
telegram.SecureValue, optional

passport

Credentials for encrypted passport.

Type
telegram.SecureValue, optional

internal_passport

Credentials for encrypted internal passport.

Type
telegram.SecureValue, optional

driver_license

Credentials for encrypted driver license.

Type
telegram.SecureValue, optional

identity_card

Credentials for encrypted ID card

Type
telegram.SecureValue, optional

address

Credentials for encrypted residential address.

Type
telegram.SecureValue, optional

utility_bill

Credentials for encrypted utility bill.

Type
telegram.SecureValue, optional

bank_statement

Credentials for encrypted bank statement.

Type
telegram.SecureValue, optional

rental_agreement

Credentials for encrypted rental agreement.

Type
telegram.SecureValue, optional

passport_registration

Credentials for encrypted registration from internal passport.

Type
telegram.SecureValue, optional

temporary_registration

Credentials for encrypted temporary registration.

Type
telegram.SecureValue, optional

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

telegram.SecureValue

class telegram.**SecureValue**(**args*, ***kwargs*)

Bases: *telegram.TelegramObject*

This object represents the credentials that were used to decrypt the encrypted value. All fields are optional and depend on the type of field.

data

Credentials for encrypted Telegram Passport data. Available for "personal_details", "passport", "driver_license", "identity_card", "identity_passport" and "address" types.

Type
telegram.DataCredentials, optional

front_side

Credentials for encrypted document's front side. Available for "passport", "driver_license", "identity_card" and "internal_passport".

Type
telegram.FileCredentials, optional

reverse_side

Credentials for encrypted document's reverse side. Available for "driver_license" and "identity_card".

Type
telegram.FileCredentials, optional

selfie

Credentials for encrypted selfie of the user with a document. Can be available for "passport", "driver_license", "identity_card" and "internal_passport".

Type
telegram.FileCredentials, optional

translation

Credentials for an encrypted translation of the document. Available for "passport", "driver_license", "identity_card", "internal_passport", "utility_bill", "bank_statement", "rental_agreement", "passport_registration" and "temporary_registration".

Type
List[*telegram.FileCredentials*], optional

files

Credentials for encrypted files. Available for "utility_bill", "bank_statement", "rental_agreement", "passport_registration" and "temporary_registration" types.

Type
List[*telegram.FileCredentials*], optional

classmethod de_json(*data*, *bot*)

See *telegram.TelegramObject.de_json()*.

to_dict()

See *telegram.TelegramObject.to_dict()*.

10.2 telegram.ext package

10.2.1 telegram.ext.Application

class telegram.ext.**Application**(*, *bot*, *update_queue*, *updater*, *job_queue*, *concurrent_updates*, *persistence*, *context_types*, *post_init*, *post_shutdown*)

Bases: typing.Generic, AbstractAsyncContextManager

This class dispatches all kinds of updates to its registered handlers, and is the entry point to a PTB application.

Tip: This class may not be initialized directly. Use *telegram.ext.ApplicationBuilder* or *builder()* (for convenience).

Instances of this class can be used as asyncio context managers, where

```
async with application:
    # code
```

is roughly equivalent to

```
try:
    await application.initialize()
    # code
finally:
    await application.shutdown()
```

Changed in version 20.0:

- Initialization is now done through the *telegram.ext.ApplicationBuilder*.
- Removed the attribute `groups`.

bot

The bot object that should be passed to the handlers.

Type
telegram.Bot

update_queue

The synchronized queue that will contain the updates.

Type
asyncio.Queue

updater

Optional. The updater used by this application.

Type
telegram.ext.Updater

job_queue

Optional. The *telegram.ext.JobQueue* instance to pass onto handler callbacks.

Type
telegram.ext.JobQueue

chat_data

A dictionary handlers can use to store data for the chat.

Changed in version 20.0: *chat_data* is now read-only

Tip: Manually modifying *chat_data* is almost never needed and unadvisable.

Type
types.MappingProxyType

user_data

A dictionary handlers can use to store data for the user.

Changed in version 20.0: *user_data* is now read-only

Tip: Manually modifying *user_data* is almost never needed and unadvisable.

Type
types.MappingProxyType

bot_data

A dictionary handlers can use to store data for the bot.

Type
dict

persistence

The persistence class to store data that should be persistent over restarts.

Type
telegram.ext.BasePersistence

handlers

A dictionary mapping each handler group to the list of handlers registered to that group.

See also:

add_handler(), *add_handlers()*.

Type
Dict[int, List[*telegram.ext.BaseHandler*]]

error_handlers

A dictionary where the keys are error handlers and the values indicate whether they are to be run blocking.

See also:

add_error_handler()

Type
Dict[coroutine function, bool]

context_types

Specifies the types used by this dispatcher for the `context` argument of handler and job callbacks.

Type
telegram.ext.ContextTypes

post_init

Optional. A callback that will be executed by *Application.run_polling()* and *Application.run_webhook()* after initializing the application via *initialize()*.

Type
coroutine function

post_shutdown

Optional. A callback that will be executed by *Application.run_polling()* and *Application.run_webhook()* after shutting down the application via *shutdown()*.

Type
coroutine function

add_error_handler(*callback*, *block=True*)

Registers an error handler in the Application. This handler will receive every error which happens in your bot. See the docs of *process_error()* for more details on how errors are handled.

Note: Attempts to add the same callback multiple times will be ignored.

See also:

Errorhandler Example

Parameters

- ***callback*** (coroutine function) – The callback function for this error handler. Will be called when an error is raised. Callback signature:

```
async def callback(update: Optional[object], context:␣
↪CallbackContext)
```

  The error that happened will be present in *telegram.ext.CallbackContext.error*.

- ***block*** (bool, optional) – Determines whether the return value of the callback should be awaited before processing the next error handler in *process_error()*. Defaults to True.

add_handler(*handler*, *group=0*)

Register a handler.

TL;DR: Order and priority counts. 0 or 1 handlers per group will be used. End handling of update with *telegram.ext.ApplicationHandlerStop*.

A handler must be an instance of a subclass of *telegram.ext.BaseHandler*. All handlers are organized in groups with a numeric value. The default group is 0. All groups will be evaluated for handling an update, but only 0 or 1 handler per group will be used. If *telegram.ext.ApplicationHandlerStop* is raised from one of the handlers, no further handlers (regardless of the group) will be called.

The priority/order of handlers is determined as follows:

- Priority of the group (lower group number == higher priority)
- The first handler in a group which can handle an update (see *telegram.ext.BaseHandler.check_update*) will be used. Other handlers from the group will not be used. The order in which handlers were added to the group defines the priority.

Warning: Adding persistent *telegram.ext.ConversationHandler* after the application has been initialized is discouraged. This is because the persisted conversation states need to be loaded into memory while the application is already processing updates, which might lead to race conditions and undesired behavior. In particular, current conversation states may be overridden by the loaded data.

Parameters

- ***handler*** (*telegram.ext.BaseHandler*) – A BaseHandler instance.
- ***group*** (int, optional) – The group identifier. Default is 0.

add_handlers(*handlers*, *group=0*)

Registers multiple handlers at once. The order of the handlers in the passed sequence(s) matters. See *add_handler()* for details.

New in version 20.0.

Parameters

- ***handlers*** (List[*telegram.ext.BaseHandler*] | Dict[int, List[*telegram.ext.BaseHandler*]]) – Specify a sequence of handlers *or* a dictionary where the keys are groups and values are handlers.
- ***group*** (int, optional) – Specify which group the sequence of *handlers* should be added to. Defaults to 0.

Example:

```
app.add_handlers(handlers={
    -1: [MessageHandler(...)],
    1: [CallbackQueryHandler(...), CommandHandler(...)]
}
```

static builder()

Convenience method. Returns a new *telegram.ext.ApplicationBuilder*.

New in version 20.0.

property concurrent_updates

The number of concurrent updates that will be processed in parallel. A value of 0 indicates updates are *not* being processed concurrently.

Type
int

create_task(*coroutine*, *update=None*)

Thin wrapper around asyncio.create_task() that handles exceptions raised by the *coroutine* with *process_error()*.

Note:

- If *coroutine* raises an exception, it will be set on the task created by this method even though it's handled by *process_error()*.
- If the application is currently running, tasks created by this method will be awaited with *stop()*.

Parameters

- ***coroutine*** (coroutine function) – The coroutine to run as task.
- ***update*** (object, optional) – If set, will be passed to *process_error()* as additional information for the error handlers. Moreover, the corresponding *chat_data* and *user_data* entries will be updated in the next run of *update_persistence()* after the *coroutine* is finished.

Returns
The created task.

Return type
asyncio.Task

drop_chat_data(*chat_id*)

Drops the corresponding entry from the *chat_data*. Will also be deleted from the persistence on the next run of *update_persistence()*, if applicable.

> **Warning:** When using *concurrent_updates* or the *job_queue*, *process_update()* or *telegram.ext.Job.run()* may re-create this entry due to the asynchronous nature of these features. Please make sure that your program can avoid or handle such situations.

New in version 20.0.

Parameters
chat_id (int) – The chat id to delete. The entry will be deleted even if it is not empty.

drop_user_data(*user_id*)

Drops the corresponding entry from the *user_data*. Will also be deleted from the persistence on the next run of *update_persistence()*, if applicable.

> **Warning:** When using *concurrent_updates* or the *job_queue*, *process_update()* or *telegram.ext.Job.run()* may re-create this entry due to the asynchronous nature of these features. Please make sure that your program can avoid or handle such situations.

New in version 20.0.

Parameters
user_id (int) – The user id to delete. The entry will be deleted even if it is not empty.

async initialize()

Initializes the Application by initializing:

- The *bot*, by calling *telegram.Bot.initialize()*.
- The *updater*, by calling *telegram.ext.Updater.initialize()*.
- The *persistence*, by loading persistent conversations and data.

Does *not* call *post_init* - that is only done by *run_polling()* and *run_webhook()*.

See also:

shutdown()

migrate_chat_data(*message=None*, *old_chat_id=None*, *new_chat_id=None*)

Moves the contents of *chat_data* at key old_chat_id to the key new_chat_id. Also marks the entries to be updated accordingly in the next run of *update_persistence()*.

> **Warning:**
> - Any data stored in *chat_data* at key new_chat_id will be overridden
> - The key *old_chat_id* of *chat_data* will be deleted
> - This does not update the *chat_id* attribute of any scheduled *telegram.ext.Job*.

Warning: When using *concurrent_updates* or the *job_queue*, *process_update()* or *telegram.ext.Job.run()* may re-create the old entry due to the asynchronous nature of these features. Please make sure that your program can avoid or handle such situations.

Parameters

- ***message*** (*telegram.Message*, optional) – A message with either *migrate_from_chat_id* or *migrate_to_chat_id*. Mutually exclusive with passing *old_chat_id* and *new_chat_id*.

  See also:

  telegram.ext.filters.StatusUpdate.MIGRATE

- ***old_chat_id*** (int, optional) – The old chat ID. Mutually exclusive with passing *message*
- ***new_chat_id*** (int, optional) – The new chat ID. Mutually exclusive with passing *message*

Raises

ValueError – Raised if the input is invalid.

async process_error(*update*, *error*, *job=None*, *coroutine=None*)

Processes an error by passing it to all error handlers registered with *add_error_handler()*. If one of the error handlers raises *telegram.ext.ApplicationHandlerStop*, the error will not be handled by other error handlers. Raising *telegram.ext.ApplicationHandlerStop* also stops processing of the update when this method is called by *process_update()*, i.e. no further handlers (even in other groups) will handle the update. All other exceptions raised by an error handler will just be logged.

Changed in version 20.0:

- `dispatch_error` was renamed to *process_error()*.
- Exceptions raised by error handlers are now properly logged.
- *telegram.ext.ApplicationHandlerStop* is no longer reraised but converted into the return value.

Parameters

- ***update*** (object | *telegram.Update*) – The update that caused the error.
- ***error*** (Exception) – The error that was raised.
- ***job*** (*telegram.ext.Job*, optional) – The job that caused the error.

  New in version 20.0.

- ***coroutine*** (coroutine function, optional) – The coroutine that caused the error.

Returns

True, if one of the error handlers raised *telegram.ext.ApplicationHandlerStop*. False, otherwise.

Return type

bool

async process_update(*update*)

Processes a single update and marks the update to be updated by the persistence later. Exceptions raised by handler callbacks will be processed by *process_update()*.

Changed in version 20.0: Persistence is now updated in an interval set by *telegram.ext.BasePersistence.update_interval*.

Parameters

update (*telegram.Update* | object | *telegram.error.TelegramError*) – The update to process.

Raises

RuntimeError – If the application was not initialized.

remove_error_handler(*callback*)

Removes an error handler.

Parameters

callback (coroutine function) – The error handler to remove.

remove_handler(*handler*, *group=0*)

Remove a handler from the specified group.

Parameters

- ***handler*** (*telegram.ext.BaseHandler*) – A *telegram.ext.BaseHandler* instance.
- ***group*** (object, optional) – The group identifier. Default is 0.

run_polling(*poll_interval=0.0*, *timeout=10*, *bootstrap_retries=-1*, *read_timeout=2*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *allowed_updates=None*, *drop_pending_updates=None*, *close_loop=True*, *stop_signals=None*)

Convenience method that takes care of initializing and starting the app, polling updates from Telegram using *telegram.ext.Updater.start_polling()* and a graceful shutdown of the app on exit.

The app will shut down when KeyboardInterrupt or SystemExit is raised. On unix, the app will also shut down on receiving the signals specified by *stop_signals*.

If *post_init* is set, it will be called between *initialize()* and *telegram.ext.Updater.start_polling()*.

If *post_shutdown* is set, it will be called after both *shutdown()* and *telegram.ext.Updater.shutdown()*.

See also:

initialize(), *start()*, *stop()*, *shutdown()* *telegram.ext.Updater.start_polling()*, *run_webhook()*

Parameters

- ***poll_interval*** (float, optional) – Time to wait between polling updates from Telegram in seconds. Default is 0.0.
- ***timeout*** (float, optional) – Passed to *telegram.Bot.get_updates.timeout*. Default is 10 seconds.
- ***bootstrap_retries*** (int, optional) – Whether the bootstrapping phase of the *telegram.ext.Updater* will retry on failures on the Telegram server.
  - < 0 - retry indefinitely (default)
  - 0 - no retries
  - > 0 - retry up to X times
- ***read_timeout*** (float, optional) – Value to pass to *telegram.Bot.get_updates.read_timeout*. Defaults to 2.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.Bot.get_updates.write_timeout*. Defaults to *DEFAULT_NONE*.

- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.Bot.get_updates.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.Bot.get_updates.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***drop_pending_updates*** (bool, optional) – Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is False.
- ***allowed_updates*** (List[str], optional) – Passed to *telegram.Bot.get_updates()*.
- ***close_loop*** (bool, optional) – If True, the current event loop will be closed upon shutdown.

  See also:

  asyncio.loop.close()

- ***stop_signals*** (Sequence[int] | None, optional) – Signals that will shut down the app. Pass None to not use stop signals. Defaults to signal.SIGINT, signal.SIGTERM and signal.SIGABRT on non Windows platforms.

  Caution: Not every asyncio.AbstractEventLoop implements asyncio.loop.add_signal_handler(). Most notably, the standard event loop on Windows, asyncio.ProactorEventLoop, does not implement this method. If this method is not available, stop signals can not be set.

Raises

RuntimeError – If the Application does not have an *telegram.ext.Updater*.

run_webhook(*listen='127.0.0.1'*, *port=80*, *url_path=''*, *cert=None*, *key=None*, *bootstrap_retries=0*, *webhook_url=None*, *allowed_updates=None*, *drop_pending_updates=None*, *ip_address=None*, *max_connections=40*, *close_loop=True*, *stop_signals=None*, *secret_token=None*)

Convenience method that takes care of initializing and starting the app, polling updates from Telegram using *telegram.ext.Updater.start_webhook()* and a graceful shutdown of the app on exit.

The app will shut down when KeyboardInterrupt or SystemExit is raised. On unix, the app will also shut down on receiving the signals specified by *stop_signals*.

If *cert* and *key* are not provided, the webhook will be started directly on `http://listen:port/url_path`, so SSL can be handled by another application. Else, the webhook will be started on `https://listen:port/url_path`. Also calls *telegram.Bot.set_webhook()* as required.

If *post_init* is set, it will be called between *initialize()* and *telegram.ext.Updater.start_webhook()*.

If *post_shutdown* is set, it will be called after both *shutdown()* and *telegram.ext.Updater.shutdown()*.

See also:

initialize(), *start()*, *stop()*, *shutdown()* *telegram.ext.Updater.start_webhook()*, *run_polling()*

Parameters

- ***listen*** (str, optional) – IP-Address to listen on. Defaults to 127.0.0.1.
- ***port*** (int, optional) – Port the bot should be listening on. Must be one of *telegram.constants.SUPPORTED_WEBHOOK_PORTS* unless the bot is running behind a proxy. Defaults to `80`.

- ***url_path*** (str, optional) – Path inside url. Defaults to `` '' ``
- ***cert*** (pathlib.Path | str, optional) – Path to the SSL certificate file.
- ***key*** (pathlib.Path | str, optional) – Path to the SSL key file.
- ***bootstrap_retries*** (int, optional) – Whether the bootstrapping phase of the *telegram.ext.Updater* will retry on failures on the Telegram server.
  - < 0 - retry indefinitely
  - 0 - no retries (default)
  - > 0 - retry up to X times
- ***webhook_url*** (str, optional) – Explicitly specify the webhook url. Useful behind NAT, reverse proxy, etc. Default is derived from *listen*, *port*, *url_path*, *cert*, and *key*.
- ***allowed_updates*** (List[str], optional) – Passed to *telegram.Bot.set_webhook()*.
- ***drop_pending_updates*** (bool, optional) – Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is False.
- ***ip_address*** (str, optional) – Passed to *telegram.Bot.set_webhook()*.
- ***max_connections*** (int, optional) – Passed to *telegram.Bot.set_webhook()*. Defaults to 40.
- ***close_loop*** (bool, optional) – If True, the current event loop will be closed upon shutdown. Defaults to True.

  See also:

  asyncio.loop.close()
- ***stop_signals*** (Sequence[int] | None, optional) – Signals that will shut down the app. Pass None to not use stop signals. Defaults to signal.SIGINT, signal.SIGTERM and signal.SIGABRT.

  Caution: Not every asyncio.AbstractEventLoop implements asyncio.loop.add_signal_handler(). Most notably, the standard event loop on Windows, asyncio.ProactorEventLoop, does not implement this method. If this method is not available, stop signals can not be set.
- ***secret_token*** (str, optional) – Secret token to ensure webhook requests originate from Telegram. See *telegram.Bot.set_webhook.secret_token* for more details.

  When added, the web server started by this call will expect the token to be set in the X-Telegram-Bot-Api-Secret-Token header of an incoming request and will raise a http.HTTPStatus.FORBIDDEN error if either the header isn't set or it is set to a wrong token.

  New in version 20.0.

property running

Indicates if this application is running.

See also:

start(), *stop()*

Type
bool

async shutdown()

Shuts down the Application by shutting down:

- *bot* by calling *telegram.Bot.shutdown()*
- *updater* by calling *telegram.ext.Updater.shutdown()*
- *persistence* by calling *update_persistence()* and *BasePersistence.flush()*

Does *not* call *post_shutdown* - that is only done by *run_polling()* and *run_webhook()*.

See also:

initialize()

Raises

RuntimeError – If the application is still *running*.

async start()

Starts

- a background task that fetches updates from *update_queue* and processes them.
- *job_queue*, if set.
- a background task that calls *update_persistence()* in regular intervals, if *persistence* is set.

Note: This does *not* start fetching updates from Telegram. To fetch updates, you need to either start *updater* manually or use one of *run_polling()* or *run_webhook()*.

See also:

stop()

Raises

RuntimeError – If the application is already running or was not initialized.

async stop()

Stops the process after processing any pending updates or tasks created by *create_task()*. Also stops *job_queue*, if set. Finally, calls *update_persistence()* and *BasePersistence.flush()* on *persistence*, if set.

Warning: Once this method is called, no more updates will be fetched from *update_queue*, even if it's not empty.

See also:

start()

Note: This does *not* stop *updater*. You need to either manually call *telegram.ext.Updater.stop()* or use one of *run_polling()* or *run_webhook()*.

Raises

RuntimeError – If the application is not running.

async update_persistence()

Updates *user_data*, *chat_data*, *bot_data* in *persistence* along with *callback_data_cache* and the conversation states of any persistent *ConversationHandler* registered for this application.

For *user_data*, *chat_data*, only entries used since the last run of this method are updated.

Tip: This method will be called in regular intervals by the application. There is usually no need to call it manually.

Note: Any data is deep copied with `copy.deepcopy()` before handing it over to the persistence in order to avoid race conditions, so all persisted data must be copyable.

See also:

telegram.ext.BasePersistence.update_interval.

10.2.2 telegram.ext.ApplicationBuilder

class telegram.ext.**ApplicationBuilder**

This class serves as initializer for *telegram.ext.Application* via the so called builder pattern. To build a *telegram.ext.Application*, one first initializes an instance of this class. Arguments for the *telegram.ext.Application* to build are then added by subsequently calling the methods of the builder. Finally, the *telegram.ext.Application* is built by calling *build()*. In the simplest case this can look like the following example.

Example

```
application = ApplicationBuilder().token("TOKEN").build()
```

Please see the description of the individual methods for information on which arguments can be set and what the defaults are when not called. When no default is mentioned, the argument will not be used by default.

Note:

- Some arguments are mutually exclusive. E.g. after calling *token()*, you can't set a custom bot with *bot()* and vice versa.
- Unless a custom *telegram.Bot* instance is set via *bot()*, *build()* will use *telegram.ext.ExtBot* for the bot.

---

application_class(*application_class*, *kwargs=None*)

Sets a custom subclass instead of *telegram.ext.Application*. The subclass's `__init__` should look like this

```
def __init__(self, custom_arg_1, custom_arg_2, ..., **kwargs):
    super().__init__(**kwargs)
    self.custom_arg_1 = custom_arg_1
    self.custom_arg_2 = custom_arg_2
```

Parameters

- ***application_class*** (`type`) – A subclass of *telegram.ext.Application*

- ***kwargs*** (Dict[str, object], optional) – Keyword arguments for the initialization. Defaults to an empty dict.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

arbitrary_callback_data(*arbitrary_callback_data*)

Specifies whether *telegram.ext.Application.bot* should allow arbitrary objects as callback data for *telegram.InlineKeyboardButton* and how many keyboards should be cached in memory. If not called, only strings can be used as callback data and no data will be stored in memory.

See also:

Arbitrary callback_data, *arbitrarycallbackdatabot.py*

Parameters
arbitrary_callback_data (bool | int) – If True is passed, the default cache size of 1024 will be used. Pass an integer to specify a different cache size.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

base_file_url(*base_file_url*)

Sets the base file URL for *telegram.ext.Application.bot*. If not called, will default to 'https://api.telegram.org/file/bot'.

See also:

telegram.Bot.base_file_url, Local Bot API Server, *base_url()*

Parameters
base_file_url (str) – The URL.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

base_url(*base_url*)

Sets the base URL for *telegram.ext.Application.bot*. If not called, will default to 'https://api.telegram.org/bot'.

See also:

telegram.Bot.base_url, Local Bot API Server, *base_file_url()*

Parameters
base_url (str) – The URL.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

bot(*bot*)

Sets a *telegram.Bot* instance for *telegram.ext.Application.bot*. Instances of subclasses like *telegram.ext.ExtBot* are also valid.

Parameters
bot (*telegram.Bot*) – The bot.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

build()

Builds a *telegram.ext.Application* with the provided arguments.

Calls *telegram.ext.JobQueue.set_application()* and *telegram.ext.BasePersistence.set_bot()* if appropriate.

Returns
telegram.ext.Application

concurrent_updates(*concurrent_updates*)

Specifies if and how many updates may be processed concurrently instead of one by one.

Warning: Processing updates concurrently is not recommended when stateful handlers like *telegram.ext.ConversationHandler* are used. Only use this if you are sure that your bot does not (explicitly or implicitly) rely on updates being processed sequentially.

Tip: When making requests to the Bot API in an asynchronous fashion (e.g. via *block=False*, *Application.create_task*, *concurrent_updates()* or the *JobQueue*), it can happen that more requests are being made in parallel than there are connections in the pool. If the number of requests is much higher than the number of connections, even setting *pool_timeout()* to a larger value may not always be enough to prevent pool timeouts. You should therefore set *concurrent_updates()*, *connection_pool_size()* and *pool_timeout()* to values that make sense for your setup.

See also:

telegram.ext.Application.concurrent_updates

Parameters
concurrent_updates (`bool` | `int`) – Passing `True` will allow for `256` updates to be processed concurrently. Pass an integer to specify a different number of updates that may be processed concurrently.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

connect_timeout(*connect_timeout*)

Sets the connection attempt timeout for the *connect_timeout* parameter of *telegram.Bot.request*. Defaults to `5.0`.

Parameters
connect_timeout (`float`) – See *telegram.request.HTTPXRequest.connect_timeout* for more information.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

connection_pool_size(*connection_pool_size*)

Sets the size of the connection pool for the *connection_pool_size* parameter of *telegram.Bot.request*. Defaults to 256.

Tip: When making requests to the Bot API in an asynchronous fashion (e.g. via *block=False*, *Application.create_task*, *concurrent_updates()* or the *JobQueue*), it can happen that more requests are being made in parallel than there are connections in the pool. If the number of requests is much higher than the number of connections, even setting *pool_timeout()* to a larger value may not always be enough to prevent pool timeouts. You should therefore set *concurrent_updates()*, *connection_pool_size()* and *pool_timeout()* to values that make sense for your setup.

Parameters
connection_pool_size (int) – The size of the connection pool.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

context_types(*context_types*)

Sets a *telegram.ext.ContextTypes* instance for *telegram.ext.Application.context_types*.

See also:

contexttypesbot.py

Parameters
context_types (*telegram.ext.ContextTypes*) – The context types.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

defaults(*defaults*)

Sets the *telegram.ext.Defaults* instance for *telegram.ext.Application.bot*.

See also:

Adding Defaults

Parameters
defaults (*telegram.ext.Defaults*) – The defaults instance.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

get_updates_connect_timeout(*get_updates_connect_timeout*)

Sets the connection attempt timeout for the *telegram.request.HTTPXRequest.connect_timeout* parameter which is used for the *telegram.Bot.get_updates()* request. Defaults to `5.0`.

Parameters
get_updates_connect_timeout (`float`) – See *telegram.request.HTTPXRequest.connect_timeout* for more information.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

get_updates_connection_pool_size(*get_updates_connection_pool_size*)

Sets the size of the connection pool for the *telegram.request.HTTPXRequest.connection_pool_size* parameter which is used for the *telegram.Bot.get_updates()* request. Defaults to `1`.

Parameters
get_updates_connection_pool_size (`int`) – The size of the connection pool.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

get_updates_pool_timeout(*get_updates_pool_timeout*)

Sets the connection pool's connection freeing timeout for the *pool_timeout* parameter which is used for the *telegram.Bot.get_updates()* request. Defaults to `None`.

Parameters
get_updates_pool_timeout (`float`) – See *telegram.request.HTTPXRequest.pool_timeout* for more information.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

get_updates_proxy_url(*get_updates_proxy_url*)

Sets the proxy for the *telegram.request.HTTPXRequest.proxy_url* parameter which is used for *telegram.Bot.get_updates()*. Defaults to `None`.

Parameters
get_updates_proxy_url (`str`) – The URL to the proxy server. See *telegram.request.HTTPXRequest.proxy_url* for more information.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

get_updates_read_timeout(*get_updates_read_timeout*)

Sets the waiting timeout for the *telegram.request.HTTPXRequest.read_timeout* parameter which is used for the *telegram.Bot.get_updates()* request. Defaults to `5.0`.

Parameters
get_updates_read_timeout (`float`) – See *telegram.request.HTTPXRequest.read_timeout* for more information.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

get_updates_request(*get_updates_request*)

Sets a *telegram.request.BaseRequest* instance for the *get_updates_request* parameter of *telegram.ext.Application.bot*.

See also:

request()

Parameters
get_updates_request (*telegram.request.BaseRequest*) – The request instance.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

get_updates_write_timeout(*get_updates_write_timeout*)

Sets the write operation timeout for the *telegram.request.HTTPXRequest.write_timeout* parameter which is used for the *telegram.Bot.get_updates()* request. Defaults to `5.0`.

Parameters
get_updates_write_timeout (`float`) – See *telegram.request.HTTPXRequest.write_timeout* for more information.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

job_queue(*job_queue*)

Sets a *telegram.ext.JobQueue* instance for *telegram.ext.Application.job_queue*. If not called, a job queue will be instantiated.

See also:

JobQueue, *timerbot.py*

Note:

- *telegram.ext.JobQueue.set_application()* will be called automatically by *build()*.
- The job queue will be automatically started and stopped by *telegram.ext.Application.start()* and *telegram.ext.Application.stop()*, respectively.
- When passing `None`, *telegram.ext.ConversationHandler.conversation_timeout* can not be used, as this uses *telegram.ext.Application.job_queue* internally.

Parameters
job_queue (*telegram.ext.JobQueue*) – The job queue. Pass `None` if you don't want to use a job queue.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

persistence(*persistence*)

Sets a *telegram.ext.BasePersistence* instance for *telegram.ext.Application.persistence*.

Note: When using a persistence, note that all data stored in *context.user_data*, *context.chat_data*, *context.bot_data* and in *telegram.ext.ExtBot.callback_data_cache* must be copyable with `copy.deepcopy()`. This is due to the data being deep copied before handing it over to the persistence in order to avoid race conditions.

See also:

Making your bot persistent, *persistentconversationbot.py*

Warning: If a *telegram.ext.ContextTypes* instance is set via *context_types()*, the persistence instance must use the same types!

Parameters
persistence (*telegram.ext.BasePersistence*) – The persistence instance.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

pool_timeout(*pool_timeout*)

Sets the connection pool's connection freeing timeout for the *pool_timeout* parameter of *telegram.Bot.request*. Defaults to `None`.

Tip: When making requests to the Bot API in an asynchronous fashion (e.g. via *block=False*, *Application.create_task*, *concurrent_updates()* or the *JobQueue*), it can happen that more requests are being made in parallel than there are connections in the pool. If the number of requests is much higher than the number of connections, even setting *pool_timeout()* to a larger value may not always be enough to prevent pool timeouts. You should therefore set *concurrent_updates()*, *connection_pool_size()* and *pool_timeout()* to values that make sense for your setup.

Parameters
pool_timeout (`float`) – See *telegram.request.HTTPXRequest.pool_timeout* for more information.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

post_init(*post_init*)

Sets a callback to be executed by *Application.run_polling()* and *Application.run_webhook()* *after* executing *Application.initialize()* but *before* executing *Updater.start_polling()* or *Updater.start_webhook()*, respectively.

Tip: This can be used for custom startup logic that requires to await coroutines, e.g. setting up the bots commands via *set_my_commands()*.

Example

```
async def post_init(application: Application) -> None:
    await application.bot.set_my_commands([('start', 'Starts the bot')])

application = Application.builder().token("TOKEN").post_init(post_init).
↪build()
```

Parameters

post_init (coroutine function) – The custom callback. Must be a coroutine function and must accept exactly one positional argument, which is the *Application*:

```
async def post_init(application: Application) -> None:
```

Returns

The same builder with the updated argument.

Return type

ApplicationBuilder

post_shutdown(*post_shutdown*)

Sets a callback to be executed by *Application.run_polling()* and *Application.run_webhook()* *after* executing *Updater.shutdown()* and *Application.shutdown()*.

Tip: This can be used for custom shutdown logic that requires to await coroutines, e.g. closing a database connection

Example

```
async def post_shutdown(application: Application) -> None:
    await application.bot_data['database'].close()

application = Application.builder()
                         .token("TOKEN")
                         .post_shutdown(post_shutdown)
                         .build()
```

Parameters

post_shutdown (coroutine function) – The custom callback. Must be a coroutine function and must accept exactly one positional argument, which is the *Application*:

```
async def post_shutdown(application: Application) -> None:
```

Returns

The same builder with the updated argument.

Return type

ApplicationBuilder

private_key(*private_key*, *password=None*)

Sets the private key and corresponding password for decryption of telegram passport data for *telegram.ext.Application.bot*.

See also:

passportbot.py, Telegram Passports

Parameters

- ***private_key*** (bytes | str | pathlib.Path) – The private key or the file path of a file that contains the key. In the latter case, the file's content will be read automatically.
- ***password*** (bytes | str | pathlib.Path, optional) – The corresponding password or the file path of a file that contains the password. In the latter case, the file's content will be read automatically.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

proxy_url(*proxy_url*)

Sets the proxy for the *proxy_url* parameter of *telegram.Bot.request*. Defaults to None.

Parameters
proxy_url (str) – The URL to the proxy server. See *telegram.request.HTTPXRequest.proxy_url* for more information.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

rate_limiter(*rate_limiter*)

Sets a *telegram.ext.BaseRateLimiter* instance for the *telegram.ext.ExtBot.rate_limiter* parameter of *telegram.ext.Application.bot*.

Parameters
rate_limiter (*telegram.ext.BaseRateLimiter*) – The rate limiter.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

read_timeout(*read_timeout*)

Sets the waiting timeout for the *read_timeout* parameter of *telegram.Bot.request*. Defaults to 5.0.

Parameters
read_timeout (float) – See *telegram.request.HTTPXRequest.read_timeout* for more information.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

request(*request*)

Sets a *telegram.request.BaseRequest* instance for the *telegram.Bot.request* parameter of *telegram.ext.Application.bot*.

See also:

get_updates_request()

Parameters
request (*telegram.request.BaseRequest*) – The request instance.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

token(*token*)

Sets the token for *telegram.ext.Application.bot*.

Parameters
token (str) – The token.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

update_queue(*update_queue*)

Sets a asyncio.Queue instance for *telegram.ext.Application.update_queue*, i.e. the queue that the application will fetch updates from. Will also be used for the *telegram.ext.Application.updater*. If not called, a queue will be instantiated.

See also:

telegram.ext.Updater.update_queue

Parameters
update_queue (asyncio.Queue) – The queue.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

updater(*updater*)

Sets a *telegram.ext.Updater* instance for *telegram.ext.Application.updater*. The *telegram.ext.Updater.bot* and *telegram.ext.Updater.update_queue* will be used for *telegram.ext.Application.bot* and *telegram.ext.Application.update_queue*, respectively.

Parameters
updater (*telegram.ext.Updater* | None) – The updater instance or None if no updater should be used.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

write_timeout(*write_timeout*)

Sets the write operation timeout for the *write_timeout* parameter of *telegram.Bot.request*. Defaults to 5.0.

Parameters
write_timeout (float) – See *telegram.request.HTTPXRequest.write_timeout* for more information.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

10.2.3 telegram.ext.ApplicationHandlerStop

class telegram.ext.**ApplicationHandlerStop**(*state=None*)

Bases: Exception

Raise this in a handler or an error handler to prevent execution of any other handler (even in different groups).

In order to use this exception in a *telegram.ext.ConversationHandler*, pass the optional *state* parameter instead of returning the next state:

```
async def conversation_callback(update, context):
    ...
    raise ApplicationHandlerStop(next_state)
```

Note: Has no effect, if the handler or error handler is run in a non-blocking way.

Parameters
state (object, optional) – The next state of the conversation.

state
Optional. The next state of the conversation.

Type
object

10.2.4 telegram.ext.CallbackContext

class telegram.ext.**CallbackContext**(*application*, *chat_id=None*, *user_id=None*)

This is a context object passed to the callback called by *telegram.ext.BaseHandler* or by the *telegram.ext.Application* in an error handler added by *telegram.ext.Application.add_error_handler* or to the callback of a *telegram.ext.Job*.

Note: *telegram.ext.Application* will create a single context for an entire update. This means that if you got 2 handlers in different groups and they both get called, they will receive the same *CallbackContext* object (of course with proper attributes like *matches* differing). This allows you to add custom attributes in a lower handler group callback, and then subsequently access those attributes in a higher handler group callback. Note that the attributes on *CallbackContext* might change in the future, so make sure to use a fairly unique name for the attributes.

Warning: Do not combine custom attributes with *telegram.ext.BaseHandler.block* set to False or *telegram.ext.Application.concurrent_updates* set to True. Due to how those work, it will almost certainly execute the callbacks for an update out of order, and the attributes that you think you added will not be present.

This class is a Generic class and accepts four type variables:

1. The type of *bot*. Must be *telegram.Bot* or a subclass of that class.
2. The type of *user_data* (if *user_data* is not None).
3. The type of *chat_data* (if *chat_data* is not None).

4. The type of *bot_data* (if *bot_data* is not None).

See also:

telegram.ext.ContextTypes.DEFAULT_TYPE

Parameters

- ***application*** (*telegram.ext.Application*) – The application associated with this context.
- ***chat_id*** (int, optional) – The ID of the chat associated with this object. Used to provide *chat_data*.

  New in version 20.0.
- ***user_id*** (int, optional) – The ID of the user associated with this object. Used to provide *user_data*.

  New in version 20.0.

coroutine

Optional. Only present in error handlers if the error was caused by a coroutine run with *Application.create_task()* or a handler callback with *block=False*.

Type
coroutine function

matches

Optional. If the associated update originated from a *filters.Regex*, this will contain a list of match objects for every pattern where `re.search(pattern, string)` returned a match. Note that filters short circuit, so combined regex filters will not always be evaluated.

Type
List[re.Match]

args

Optional. Arguments passed to a command if the associated update is handled by *telegram.ext.CommandHandler*, *telegram.ext.PrefixHandler* or *telegram.ext.StringCommandHandler*. It contains a list of the words in the text after the command, using any whitespace string as a delimiter.

Type
List[str]

error

Optional. The error that was raised. Only present when passed to an error handler registered with *telegram.ext.Application.add_error_handler*.

Type
Exception

job

Optional. The job which originated this callback. Only present when passed to the callback of *telegram.ext.Job* or in error handlers if the error is caused by a job.

Changed in version 20.0: *job* is now also present in error handlers if the error is caused by a job.

Type
telegram.ext.Job

property application

The application associated with this context.

Type
telegram.ext.Application

property bot

The bot associated with this context.

Type
telegram.Bot

property bot_data

Optional. An object that can be used to keep any data in. For each update it will be the same *ContextTypes.bot_data*. Defaults to `dict`.

Type
ContextTypes.bot_data

property chat_data

Optional. An object that can be used to keep any data in. For each update from the same chat id it will be the same *ContextTypes.chat_data*. Defaults to `dict`.

> **Warning:** When a group chat migrates to a supergroup, its chat id will change and the `chat_data` needs to be transferred. For details see our wiki page.

Changed in version 20.0: The chat data is now also present in error handlers if the error is caused by a job.

Type
ContextTypes.chat_data

drop_callback_data(*callback_query*)

Deletes the cached data for the specified callback query.

New in version 13.6.

Note: Will *not* raise exceptions in case the data is not found in the cache. *Will* raise `KeyError` in case the callback query can not be found in the cache.

Parameters
callback_query (*telegram.CallbackQuery*) – The callback query.

Raises
KeyError | RuntimeError – `KeyError`, if the callback query can not be found in the cache and `RuntimeError`, if the bot doesn't allow for arbitrary callback data.

classmethod from_error(*update*, *error*, *application*, *job=None*, *coroutine=None*)

Constructs an instance of *telegram.ext.CallbackContext* to be passed to the error handlers.

See also:

telegram.ext.Application.add_error_handler()

Changed in version 20.0: Removed arguments `async_args` and `async_kwargs`.

Parameters

- ***update*** (`object` | *telegram.Update*) – The update associated with the error. May be `None`, e.g. for errors in job callbacks.
- ***error*** (`Exception`) – The error.
- ***application*** (*telegram.ext.Application*) – The application associated with this context.
- ***job*** (*telegram.ext.Job*, optional) – The job associated with the error.

  New in version 20.0.

- ***coroutine*** (coroutine function, optional) – The coroutine function associated with this error if the error was caused by a coroutine run with *Application.create_task()* or a handler callback with *block=False*.

  New in version 20.0.

Returns
telegram.ext.CallbackContext

classmethod from_job(*job*, *application*)

Constructs an instance of *telegram.ext.CallbackContext* to be passed to a job callback.

See also:

telegram.ext.JobQueue()

Parameters

- ***job*** (*telegram.ext.Job*) – The job.
- ***application*** (*telegram.ext.Application*) – The application associated with this context.

Returns
telegram.ext.CallbackContext

classmethod from_update(*update*, *application*)

Constructs an instance of *telegram.ext.CallbackContext* to be passed to the handlers.

See also:

telegram.ext.Application.add_handler()

Parameters

- ***update*** (object | *telegram.Update*) – The update.
- ***application*** (*telegram.ext.Application*) – The application associated with this context.

Returns
telegram.ext.CallbackContext

property job_queue

The *JobQueue* used by the *telegram.ext.Application*.

Type
telegram.ext.JobQueue

property match

The first match from *matches*. Useful if you are only filtering using a single regex filter. Returns None if *matches* is empty.

Type
re.Match

async refresh_data()

If *application* uses persistence, calls *telegram.ext.BasePersistence.refresh_bot_data()* on *bot_data*, *telegram.ext.BasePersistence.refresh_chat_data()* on *chat_data* and *telegram.ext.BasePersistence.refresh_user_data()* on *user_data*, if appropriate.

Will be called by *telegram.ext.Application.process_update()* and *telegram.ext.Job.run()*.

New in version 13.6.

update(*data*)

Updates `self.__slots__` with the passed data.

Parameters

data (Dict[str, object]) – The data.

property update_queue

The asyncio.Queue instance used by the *telegram.ext.Application* and (usually) the *telegram.ext.Updater* associated with this context.

Type

asyncio.Queue

property user_data

Optional. An object that can be used to keep any data in. For each update from the same user it will be the same *ContextTypes.user_data*. Defaults to dict.

Changed in version 20.0: The user data is now also present in error handlers if the error is caused by a job.

Type

ContextTypes.user_data

10.2.5 telegram.ext.ContextTypes

class telegram.ext.**ContextTypes**(*context=<class 'telegram.ext._callbackcontext.CallbackContext'>*, *bot_data=<class 'dict'>*, *chat_data=<class 'dict'>*, *user_data=<class 'dict'>*)

Bases: typing.Generic

Convenience class to gather customizable types of the *telegram.ext.CallbackContext* interface.

See also:

ContextTypes Example

New in version 13.6.

Parameters

- ***context*** (type, optional) – Determines the type of the `context` argument of all (error-)handler callbacks and job callbacks. Must be a subclass of *telegram.ext.CallbackContext*. Defaults to *telegram.ext.CallbackContext*.
- ***bot_data*** (type, optional) – Determines the type of *context.bot_data* of all (error-)handler callbacks and job callbacks. Defaults to dict. Must support instantiating without arguments.
- ***chat_data*** (type, optional) – Determines the type of *context.chat_data* of all (error-)handler callbacks and job callbacks. Defaults to dict. Must support instantiating without arguments.
- ***user_data*** (type, optional) – Determines the type of *context.user_data* of all (error-)handler callbacks and job callbacks. Defaults to dict. Must support instantiating without arguments.

DEFAULT_TYPE

Shortcut for the type annotation for the `context` argument that's correct for the default settings, i.e. if *telegram.ext.ContextTypes* is not used.

Example

```
async def callback(update: Update, context: ContextTypes.DEFAULT_TYPE):
    ...
```

---

alias of *CallbackContext*[*ExtBot*, dict, dict, dict]

property bot_data

The type of *context.bot_data* of all (error-)handler callbacks and job callbacks.

property chat_data

The type of *context.chat_data* of all (error-)handler callbacks and job callbacks.

property context

The type of the context argument of all (error-)handler callbacks and job callbacks.

property user_data

The type of *context.user_data* of all (error-)handler callbacks and job callbacks.

10.2.6 telegram.ext.Defaults

class telegram.ext.**Defaults**(*parse_mode=None*, *disable_notification=None*, *disable_web_page_preview=None*, *quote=None*, *tzinfo=<UTC>*, *block=True*, *allow_sending_without_reply=None*, *protect_content=None*)

Bases: object

Convenience Class to gather all parameters with a (user defined) default value

Changed in version 20.0: Removed the argument and attribute timeout. Specify default timeout behavior for the networking backend directly via *telegram.ext.ApplicationBuilder* instead.

Parameters

- ***parse_mode*** (str, optional) – Send *MARKDOWN* or *HTML*, if you want Telegram apps to show bold, italic, fixed-width text or URLs in your bot's message.
- ***disable_notification*** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***disable_web_page_preview*** (bool, optional) – Disables link previews for links in this message.
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- ***quote*** (bool, optional) – If set to True, the reply is sent as an actual reply to the message. If reply_to_message_id is passed, this parameter will be ignored. Default: True in group chats and False in private chats.
- ***tzinfo*** (*tzinfo*, optional) – A timezone to be used for all date(time) inputs appearing throughout PTB, i.e. if a timezone naive date(time) object is passed somewhere, it will be assumed to be in *tzinfo*. Must be a timezone provided by the pytz module. Defaults to UTC.
- ***block*** (bool, optional) – Default setting for the *BaseHandler.block* parameter of handlers and error handlers registered through *Application.add_handler()* and *Application.add_error_handler()*. Defaults to True.
- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 20.0.

property allow_sending_without_reply

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

Type
`bool`

property block

Optional. Default setting for the *BaseHandler.block* parameter of handlers and error handlers registered through *Application.add_handler()* and *Application.add_error_handler()*.

Type
`bool`

property disable_notification

Optional. Sends the message silently. Users will receive a notification with no sound.

Type
`bool`

property disable_web_page_preview

Optional. Disables link previews for links in this message.

Type
`bool`

property explanation_parse_mode

Optional. Alias for *parse_mode*, used for the corresponding parameter of *telegram.Bot.send_poll()*.

Type
`str`

property parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or URLs in your bot's message.

Type
`str`

property protect_content

Optional. Protects the contents of the sent message from forwarding and saving.

New in version 20.0.

Type
`bool`

property quote

Optional. If set to `True`, the reply is sent as an actual reply to the message. If `reply_to_message_id` is passed, this parameter will be ignored. Default: `True` in group chats and `False` in private chats.

Type
`bool`

property tzinfo

A timezone to be used for all date(time) objects appearing throughout PTB.

Type
tzinfo

10.2.7 telegram.ext.ExtBot

class telegram.ext.**ExtBot**(**args*, ***kwargs*)

Bases: *telegram.Bot*, typing.Generic

This object represents a Telegram Bot with convenience extensions.

> **Warning:** Not to be confused with *telegram.Bot*.

For the documentation of the arguments, methods and attributes, please see *telegram.Bot*.

All API methods of this class have an additional keyword argument `rate_limit_args`. This can be used to pass additional information to the rate limiter, specifically to *telegram.ext.BaseRateLimiter.process_request.rate_limit_args*.

> **Warning:**
>
> - The keyword argument `rate_limit_args` can *not* be used, if *rate_limiter* is `None`.
> - The method *get_updates()* is the only method that does not have the additional argument, as this method will never be rate limited.

See also:

Arbitrary Callback Example, Arbitrary callback_data

New in version 13.6.

Parameters

- ***defaults*** (*telegram.ext.Defaults*, optional) – An object containing default values to be used if not set explicitly in the bot methods.
- ***arbitrary_callback_data*** (`bool` | `int`, optional) – Whether to allow arbitrary objects as callback data for *telegram.InlineKeyboardButton*. Pass an integer to specify the maximum number of objects cached in memory. For more details, please see our wiki. Defaults to `False`.
- ***rate_limiter*** (*telegram.ext.BaseRateLimiter*, optional) – A rate limiter to use for limiting the number of requests made by the bot per time interval.

  New in version 20.0.

arbitrary_callback_data

Whether this bot instance allows to use arbitrary objects as callback data for *telegram.InlineKeyboardButton*.

Type
`bool` | `int`

callback_data_cache

The cache for objects passed as callback data for *telegram.InlineKeyboardButton*.

Type
telegram.ext.CallbackDataCache

property defaults

The *telegram.ext.Defaults* used by this bot, if any.

async initialize()

See *telegram.Bot.initialize()*. Also initializes the *ExtBot.rate_limiter* (if set) by calling *telegram.ext.BaseRateLimiter.initialize()*.

insert_callback_data(*update*)

If this bot allows for arbitrary callback data, this inserts the cached data into all corresponding buttons within this update.

Note: Checks *telegram.Message.via_bot* and *telegram.Message.from_user* to figure out if a) a reply markup exists and b) it was actually sent by this bot. If not, the message will be returned unchanged.

Note that this will fail for channel posts, as *telegram.Message.from_user* is None for those! In the corresponding reply markups, the callback data will be replaced by *telegram.ext.InvalidCallbackData*.

Warning: *In place*, i.e. the passed *telegram.Message* will be changed!

Parameters

update (*telegram.Update*) – The update.

property rate_limiter

The *telegram.ext.BaseRateLimiter* used by this bot, if any.

New in version 20.0.

async shutdown()

See *telegram.Bot.shutdown()*. Also shuts down the *ExtBot.rate_limiter* (if set) by calling *telegram.ext.BaseRateLimiter.shutdown()*.

10.2.8 telegram.ext.Job

class telegram.ext.**Job**(*callback*, *data=None*, *name=None*, *job=None*, *chat_id=None*, *user_id=None*)

Bases: object

This class is a convenience wrapper for the jobs held in a *telegram.ext.JobQueue*. With the current backend APScheduler, *job* holds a apscheduler.job.Job instance.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their id is equal.

Note:

- All attributes and instance methods of *job* are also directly available as attributes/methods of the corresponding *telegram.ext.Job* object.
- If *job* isn't passed on initialization, it must be set manually afterwards for this *telegram.ext.Job* to be useful.

Changed in version 20.0:

- Removed argument and attribute job_queue.
- Renamed Job.context to *Job.data*.

Parameters

- ***callback*** (coroutine function) – The callback function that should be executed by the new job. Callback signature:

```
async def callback(context: CallbackContext)
```

- ***data*** (object, optional) – Additional data needed for the callback function. Can be accessed through *Job.data* in the callback. Defaults to None.
- ***name*** (str, optional) – The name of the new job. Defaults to callback.__name__.
- ***job*** (apscheduler.job.Job, optional) – The APS Job this job is a wrapper for.
- ***chat_id*** (int, optional) – Chat id of the chat that this job is associated with.

  New in version 20.0.
- ***user_id*** (int, optional) – User id of the user that this job is associated with.

  New in version 20.0.

callback

The callback function that should be executed by the new job.

Type
coroutine function

data

Optional. Additional data needed for the callback function.

Type
object

name

Optional. The name of the new job.

Type
str

job

Optional. The APS Job this job is a wrapper for.

Type
apscheduler.job.Job

chat_id

Optional. Chat id of the chat that this job is associated with.

New in version 20.0.

Type
int

user_id

Optional. User id of the user that this job is associated with.

New in version 20.0.

Type
int

property enabled

Whether this job is enabled.

Type
bool

property next_t

Datetime for the next job execution. Datetime is localized according to datetime.datetime.tzinfo. If job is removed or already ran it equals to None.

> **Warning:** This attribute is only available, if the *telegram.ext.JobQueue* this job belongs to is already started. Otherwise APScheduler raises an AttributeError.

Type
datetime.datetime

property removed

Whether this job is due to be removed.

Type
bool

async run(*application*)

Executes the callback function independently of the jobs schedule. Also calls *telegram.ext.Application.update_persistence()*.

Changed in version 20.0: Calls *telegram.ext.Application.update_persistence()*.

Parameters
application (*telegram.ext.Application*) – The application this job is associated with.

schedule_removal()

Schedules this job for removal from the *JobQueue*. It will be removed without executing its callback function again.

10.2.9 telegram.ext.JobQueue

class telegram.ext.**JobQueue**

Bases: object

This class allows you to periodically perform tasks with the bot. It is a convenience wrapper for the APScheduler library.

See also:

telegram.ext.Application.job_queue, *telegram.ext.CallbackContext.job_queue*, Timerbot Example, Job Queue

scheduler

The scheduler.

Changed in version 20.0: Uses AsyncIOScheduler instead of BackgroundScheduler

Type
apscheduler.schedulers.asyncio.AsyncIOScheduler

property application

The application this JobQueue is associated with.

get_jobs_by_name(*name*)

Returns a tuple of all *pending/scheduled* jobs with the given name that are currently in the *JobQueue*.

Returns
Tuple of all *pending* or *scheduled* jobs matching the name.

Return type
Tuple[*Job*]

jobs()

Returns a tuple of all *scheduled* jobs that are currently in the *JobQueue*.

Returns
Tuple of all *scheduled* jobs.

Return type
Tuple[*Job*]

run_custom(*callback*, *job_kwargs*, *data=None*, *name=None*, *chat_id=None*, *user_id=None*)

Creates a new custom defined *Job*.

Parameters

- ***callback*** (coroutine function) – The callback function that should be executed by the new job. Callback signature:

```
async def callback(context: CallbackContext)
```

- ***job_kwargs*** (dict) – Arbitrary keyword arguments. Used as arguments for apscheduler.schedulers.base.BaseScheduler.add_job().
- ***data*** (object, optional) – Additional data needed for the callback function. Can be accessed through *Job.data* in the callback. Defaults to None.

  Changed in version 20.0: Renamed the parameter `context` to *data*.

- ***name*** (str, optional) – The name of the new job. Defaults to callback.__name__.
- ***chat_id*** (int, optional) – Chat id of the chat associated with this job. If passed, the corresponding *chat_data* will be available in the callback.

  New in version 20.0.

- ***user_id*** (int, optional) – User id of the user associated with this job. If passed, the corresponding *user_data* will be available in the callback.

  New in version 20.0.

Returns
The new *Job* instance that has been added to the job queue.

Return type
telegram.ext.Job

run_daily(*callback*, *time*, *days=(0, 1, 2, 3, 4, 5, 6)*, *data=None*, *name=None*, *chat_id=None*, *user_id=None*, *job_kwargs=None*)

Creates a new *Job* that runs on a daily basis and adds it to the queue.

Note: For a note about DST, please see the documentation of APScheduler.

Parameters

- ***callback*** (coroutine function) – The callback function that should be executed by the new job. Callback signature:

```
async def callback(context: CallbackContext)
```

- ***time*** (datetime.time) – Time of day at which the job should run. If the timezone (datetime.time.tzinfo) is None, the default timezone of the bot will be used, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

- ***days*** (Tuple[int], optional) – Defines on which days of the week the job should run (where `0-6` correspond to sunday - saturday). By default, the job will run every day.

  Changed in version 20.0: Changed day of the week mapping of 0-6 from monday-sunday to sunday-saturday.

- ***data*** (object, optional) – Additional data needed for the callback function. Can be accessed through *Job.data* in the callback. Defaults to None.

  Changed in version 20.0: Renamed the parameter `context` to *data*.

- ***name*** (str, optional) – The name of the new job. Defaults to `callback.__name__`.

- ***chat_id*** (int, optional) – Chat id of the chat associated with this job. If passed, the corresponding *chat_data* will be available in the callback.

  New in version 20.0.

- ***user_id*** (int, optional) – User id of the user associated with this job. If passed, the corresponding *user_data* will be available in the callback.

  New in version 20.0.

- ***job_kwargs*** (dict, optional) – Arbitrary keyword arguments to pass to the `apscheduler.schedulers.base.BaseScheduler.add_job()`.

Returns
The new *Job* instance that has been added to the job queue.

Return type
telegram.ext.Job

run_monthly(*callback*, *when*, *day*, *data=None*, *name=None*, *chat_id=None*, *user_id=None*, *job_kwargs=None*)

Creates a new *Job* that runs on a monthly basis and adds it to the queue.

Changed in version 20.0: The `day_is_strict` argument was removed. Instead one can now pass `-1` to the *day* parameter to have the job run on the last day of the month.

Parameters

- ***callback*** (coroutine function) – The callback function that should be executed by the new job. Callback signature:

```
async def callback(context: CallbackContext)
```

- ***when*** (datetime.time) – Time of day at which the job should run. If the timezone (`when.tzinfo`) is None, the default timezone of the bot will be used, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

- ***day*** (int) – Defines the day of the month whereby the job would run. It should be within the range of 1 and 31, inclusive. If a month has fewer days than this number, the job will not run in this month. Passing `-1` leads to the job running on the last day of the month.

- ***data*** (object, optional) – Additional data needed for the callback function. Can be accessed through *Job.data* in the callback. Defaults to None.

  Changed in version 20.0: Renamed the parameter `context` to *data*.

- ***name*** (str, optional) – The name of the new job. Defaults to `callback.__name__`.

- ***chat_id*** (int, optional) – Chat id of the chat associated with this job. If passed, the corresponding *chat_data* will be available in the callback.

  New in version 20.0.

- ***user_id*** (int, optional) – User id of the user associated with this job. If passed, the corresponding *user_data* will be available in the callback.

  New in version 20.0.

- ***job_kwargs*** (dict, optional) – Arbitrary keyword arguments to pass to the apscheduler.schedulers.base.BaseScheduler.add_job().

Returns
The new *Job* instance that has been added to the job queue.

Return type
telegram.ext.Job

run_once(*callback*, *when*, *data=None*, *name=None*, *chat_id=None*, *user_id=None*, *job_kwargs=None*)

Creates a new *Job* instance that runs once and adds it to the queue.

Parameters

- ***callback*** (coroutine function) – The callback function that should be executed by the new job. Callback signature:

```
async def callback(context: CallbackContext)
```

- ***when*** (int | float | datetime.timedelta | datetime.datetime | datetime.time) – Time in or at which the job should run. This parameter will be interpreted depending on its type.
  - int or float will be interpreted as "seconds from now" in which the job should run.
  - datetime.timedelta will be interpreted as "time from now" in which the job should run.
  - datetime.datetime will be interpreted as a specific date and time at which the job should run. If the timezone (datetime.datetime.tzinfo) is None, the default timezone of the bot will be used, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
  - datetime.time will be interpreted as a specific time of day at which the job should run. This could be either today or, if the time has already passed, tomorrow. If the timezone (datetime.time.tzinfo) is None, the default timezone of the bot will be used, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***chat_id*** (int, optional) – Chat id of the chat associated with this job. If passed, the corresponding *chat_data* will be available in the callback.

  New in version 20.0.

- ***user_id*** (int, optional) – User id of the user associated with this job. If passed, the corresponding *user_data* will be available in the callback.

  New in version 20.0.

- ***data*** (object, optional) – Additional data needed for the callback function. Can be accessed through *Job.data* in the callback. Defaults to None.

  Changed in version 20.0: Renamed the parameter `context` to *data*.

- ***name*** (str, optional) – The name of the new job. Defaults to callback.__name__.
- ***job_kwargs*** (dict, optional) – Arbitrary keyword arguments to pass to the apscheduler.schedulers.base.BaseScheduler.add_job().

Returns
The new *Job* instance that has been added to the job queue.

Return type
telegram.ext.Job

run_repeating(*callback*, *interval*, *first=None*, *last=None*, *data=None*, *name=None*, *chat_id=None*, *user_id=None*, *job_kwargs=None*)

Creates a new *Job* instance that runs at specified intervals and adds it to the queue.

Note: For a note about DST, please see the documentation of APScheduler.

Parameters

- ***callback*** (coroutine function) – The callback function that should be executed by the new job. Callback signature:

```
async def callback(context: CallbackContext)
```

- ***interval*** (int | float | datetime.timedelta) – The interval in which the job will run. If it is an int or a float, it will be interpreted as seconds.
- ***first*** (int | float | datetime.timedelta | datetime.datetime | datetime.time, optional) – Time in or at which the job should run. This parameter will be interpreted depending on its type.
  - int or float will be interpreted as "seconds from now" in which the job should run.
  - datetime.timedelta will be interpreted as "time from now" in which the job should run.
  - datetime.datetime will be interpreted as a specific date and time at which the job should run. If the timezone (datetime.datetime.tzinfo) is None, the default timezone of the bot will be used.
  - datetime.time will be interpreted as a specific time of day at which the job should run. This could be either today or, if the time has already passed, tomorrow. If the timezone (datetime.time.tzinfo) is None, the default timezone of the bot will be used, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

  Defaults to *interval*
- ***last*** (int | float | datetime.timedelta | datetime.datetime | datetime.time, optional) – Latest possible time for the job to run. This parameter will be interpreted depending on its type. See *first* for details.

  If *last* is datetime.datetime or datetime.time type and last.tzinfo is None, the default timezone of the bot will be assumed, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

  Defaults to None.
- ***data*** (object, optional) – Additional data needed for the callback function. Can be accessed through *Job.data* in the callback. Defaults to None.

  Changed in version 20.0: Renamed the parameter context to *data*.
- ***name*** (str, optional) – The name of the new job. Defaults to callback.__name__.
- ***chat_id*** (int, optional) – Chat id of the chat associated with this job. If passed, the corresponding *chat_data* will be available in the callback.

  New in version 20.0.
- ***user_id*** (int, optional) – User id of the user associated with this job. If passed, the corresponding *user_data* will be available in the callback.

  New in version 20.0.
- ***job_kwargs*** (dict, optional) – Arbitrary keyword arguments to pass to the apscheduler.schedulers.base.BaseScheduler.add_job().

Returns
The new *Job* instance that has been added to the job queue.

Return type
telegram.ext.Job

set_application(*application*)

Set the application to be used by this JobQueue.

Parameters
application (*telegram.ext.Application*) – The application.

async start()

Starts the *JobQueue*.

async stop(*wait=True*)

Shuts down the *JobQueue*.

Parameters
wait (bool, optional) – Whether to wait until all currently running jobs have finished. Defaults to True.

10.2.10 telegram.ext.Updater

class telegram.ext.**Updater**(*bot*, *update_queue*)

Bases: AbstractAsyncContextManager

This class fetches updates for the bot either via long polling or by starting a webhook server. Received updates are enqueued into the *update_queue* and may be fetched from there to handle them appropriately.

Instances of this class can be used as asyncio context managers, where

```
async with updater:
    # code
```

is roughly equivalent to

```
try:
    await updater.initialize()
    # code
finally:
    await updater.shutdown()
```

Changed in version 20.0:

- Removed argument and attribute user_sig_handler
- The only arguments and attributes are now *bot* and *update_queue* as now the sole purpose of this class is to fetch updates. The entry point to a PTB application is now *telegram.ext.Application*.

Parameters

- ***bot*** (*telegram.Bot*) – The bot used with this Updater.
- ***update_queue*** (asyncio.Queue) – Queue for the updates.

bot

The bot used with this Updater.

Type
telegram.Bot

update_queue

Queue for the updates.

Type
asyncio.Queue

async initialize()

Initializes the Updater & the associated *bot* by calling *telegram.Bot.initialize()*.

See also:

shutdown()

async shutdown()

Shutdown the Updater & the associated *bot* by calling *telegram.Bot.shutdown()*.

See also:

initialize()

Raises
RuntimeError – If the updater is still running.

async start_polling(*poll_interval=0.0*, *timeout=10*, *bootstrap_retries=-1*, *read_timeout=2*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *allowed_updates=None*, *drop_pending_updates=None*, *error_callback=None*)

Starts polling updates from Telegram.

Changed in version 20.0: Removed the `clean` argument in favor of *drop_pending_updates*.

Parameters

- ***poll_interval*** (float, optional) – Time to wait between polling updates from Telegram in seconds. Default is `0.0`.
- ***timeout*** (float, optional) – Passed to *telegram.Bot.get_updates.timeout*. Defaults to `10` seconds.
- ***bootstrap_retries*** (int, optional) – Whether the bootstrapping phase of the *telegram.ext.Updater* will retry on failures on the Telegram server.
  - < 0 - retry indefinitely (default)
  - 0 - no retries
  - > 0 - retry up to X times
- ***read_timeout*** (float, optional) – Value to pass to *telegram.Bot.get_updates.read_timeout*. Defaults to 2.
- ***write_timeout*** (float | None, optional) – Value to pass to *telegram.Bot.get_updates.write_timeout*. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – Value to pass to *telegram.Bot.get_updates.connect_timeout*. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – Value to pass to *telegram.Bot.get_updates.pool_timeout*. Defaults to *DEFAULT_NONE*.
- ***allowed_updates*** (List[str], optional) – Passed to *telegram.Bot.get_updates()*.
- ***drop_pending_updates*** (bool, optional) – Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is False.

  New in version 13.4.

- ***error_callback*** (Callable[[*telegram.error.TelegramError*], None], optional) – Callback to handle *telegram.error.TelegramError* s that occur while calling *telegram.Bot.get_updates()* during polling. Defaults to None, in which case errors will be logged. Callback signature:

```
def callback(error: telegram.error.TelegramError)
```

Note: The *error_callback* must *not* be a coroutine function! If asynchronous behavior of the callback is wanted, please schedule a task from within the callback.

Returns
The update queue that can be filled from the main thread.

Return type
asyncio.Queue

Raises
RuntimeError – If the updater is already running or was not initialized.

async start_webhook(*listen='127.0.0.1'*, *port=80*, *url_path=''*, *cert=None*, *key=None*, *bootstrap_retries=0*, *webhook_url=None*, *allowed_updates=None*, *drop_pending_updates=None*, *ip_address=None*, *max_connections=40*, *secret_token=None*)

Starts a small http server to listen for updates via webhook. If *cert* and *key* are not provided, the webhook will be started directly on `http://listen:port/url_path`, so SSL can be handled by another application. Else, the webhook will be started on `https://listen:port/url_path`. Also calls *telegram.Bot.set_webhook()* as required.

Changed in version 13.4: *start_webhook()* now *always* calls *telegram.Bot.set_webhook()*, so pass `webhook_url` instead of calling `updater.bot.set_webhook(webhook_url)` manually.

Changed in version 20.0: Removed the `clean` argument in favor of *drop_pending_updates* and removed the deprecated argument `force_event_loop`.

Parameters

- ***listen*** (str, optional) – IP-Address to listen on. Defaults to 127.0.0.1.
- ***port*** (int, optional) – Port the bot should be listening on. Must be one of *telegram.constants.SUPPORTED_WEBHOOK_PORTS* unless the bot is running behind a proxy. Defaults to `80`.
- ***url_path*** (str, optional) – Path inside url (http(s)://listen:port/<url_path>). Defaults to `''`.
- ***cert*** (pathlib.Path | str, optional) – Path to the SSL certificate file.
- ***key*** (pathlib.Path | str, optional) – Path to the SSL key file.
- ***drop_pending_updates*** (bool, optional) – Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is False.

  New in version 13.4.

- ***bootstrap_retries*** (int, optional) – Whether the bootstrapping phase of the *telegram.ext.Updater* will retry on failures on the Telegram server.
  - < 0 - retry indefinitely
  - 0 - no retries (default)
  - > 0 - retry up to X times
- ***webhook_url*** (str, optional) – Explicitly specify the webhook url. Useful behind NAT, reverse proxy, etc. Default is derived from *listen*, *port*, *url_path*, *cert*, and *key*.

- ***ip_address*** (str, optional) – Passed to *telegram.Bot.set_webhook()*. Defaults to None.

  New in version 13.4.

- ***allowed_updates*** (List[str], optional) – Passed to *telegram.Bot.set_webhook()*. Defaults to None.

- ***max_connections*** (int, optional) – Passed to *telegram.Bot.set_webhook()*. Defaults to 40.

  New in version 13.6.

- ***secret_token*** (str, optional) – Passed to *telegram.Bot.set_webhook()*. Defaults to None.

  When added, the web server started by this call will expect the token to be set in the X-Telegram-Bot-Api-Secret-Token header of an incoming request and will raise a http.HTTPStatus.FORBIDDEN error if either the header isn't set or it is set to a wrong token.

  New in version 20.0.

Returns
The update queue that can be filled from the main thread.

Return type
queue.Queue

Raises
RuntimeError – If the updater is already running or was not initialized.

async stop()

Stops the polling/webhook.

See also:

start_polling(), *start_webhook()*

Raises
RuntimeError – If the updater is not running.

10.2.11 Handlers

telegram.ext.BaseHandler

class telegram.ext.**BaseHandler**(*callback*, *block=True*)

Bases: typing.Generic, ABC

The base class for all update handlers. Create custom handlers by inheriting from it.

> **Warning:** When setting *block* to False, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

This class is a Generic class and accepts two type variables:

1. The type of the updates that this handler will handle. Must coincide with the type of the first argument of *callback*. *check_update()* must only accept updates of this type.
2. The type of the second argument of *callback*. Must coincide with the type of the parameters *handle_update.context* and *collect_additional_context.context* as well as the second argument of *callback*. Must be either *CallbackContext* or a subclass of that class.

Tip: For this type variable, one should usually provide a `TypeVar` that is also used for the mentioned method arguments. That way, a type checker can check whether this handler fits the definition of the *Application*.

Changed in version 20.0:

- The attribute `run_async` is now *block*.
- This class was previously named `Handler`.

Parameters

- ***callback*** (coroutine function) – The callback function for this handler. Will be called when *check_update()* has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

  The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.

- ***block*** (`bool`, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*. Defaults to `True`.

callback

The callback function for this handler.

Type
coroutine function

block

Determines whether the callback will run in a blocking way..

Type
`bool`

abstract check_update(*update*)

This method is called to determine if an update should be handled by this handler instance. It should always be overridden.

Note: Custom updates types can be handled by the application. Therefore, an implementation of this method should always check the type of *update*.

Parameters
update (`object` | *telegram.Update*) – The update to be tested.

Returns
Either `None` or `False` if the update should not be handled. Otherwise an object that will be passed to *handle_update()* and *collect_additional_context()* when the update gets handled.

collect_additional_context(*context*, *update*, *application*, *check_result*)

Prepares additional arguments for the context. Override if needed.

Parameters

- ***context*** (*telegram.ext.CallbackContext*) – The context object.
- ***update*** (*telegram.Update*) – The update to gather chat/user id from.

- ***application*** (*telegram.ext.Application*) – The calling application.
- ***check_result*** – The result (return value) from *check_update()*.

async handle_update(*update*, *application*, *check_result*, *context*)

This method is called if it was determined that an update should indeed be handled by this instance. Calls *callback* along with its respectful arguments. To work with the *telegram.ext.ConversationHandler*, this method returns the value returned from *callback*. Note that it can be overridden if needed by the subclassing handler.

Parameters

- ***update*** (str | *telegram.Update*) – The update to be handled.
- ***application*** (*telegram.ext.Application*) – The calling application.
- ***check_result*** (object) – The result from *check_update()*.
- ***context*** (*telegram.ext.CallbackContext*) – The context as provided by the application.

telegram.ext.CallbackQueryHandler

class telegram.ext.**CallbackQueryHandler**(*callback*, *pattern=None*, *block=True*)

Bases: *telegram.ext.BaseHandler*

BaseHandler class to handle Telegram *callback queries*. Optionally based on a regex.

Read the documentation of the re module for more information.

Note:

- If your bot allows arbitrary objects as *callback_data*, it may happen that the original *callback_data* for the incoming *telegram.CallbackQuery* can not be found. This is the case when either a malicious client tempered with the *telegram.CallbackQuery.data* or the data was simply dropped from cache or not persisted. In these cases, an instance of *telegram.ext.InvalidCallbackData* will be set as *telegram.CallbackQuery.data*.

  New in version 13.6.

Warning: When setting *block* to False, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

Parameters

- ***callback*** (coroutine function) – The callback function for this handler. Will be called when *check_update()* has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

  The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.

- ***pattern*** (str | re.Pattern | callable | type, optional) – Pattern to test *telegram.CallbackQuery.data* against. If a string or a regex pattern is passed, re.match() is used on *telegram.CallbackQuery.data* to determine if an update should be handled by this handler. If your bot allows arbitrary objects as *callback_data*, non-strings will be accepted. To filter arbitrary objects you may pass:

- a callable, accepting exactly one argument, namely the *telegram.CallbackQuery.data*. It must return `True` or `False`/`None` to indicate, whether the update should be handled.
- a `type`. If *telegram.CallbackQuery.data* is an instance of that type (or a subclass), the update will be handled.

If *telegram.CallbackQuery.data* is `None`, the *telegram.CallbackQuery* update will not be handled.

Changed in version 13.6: Added support for arbitrary callback data.

- ***block*** (`bool`, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*. Defaults to `True`.

callback

The callback function for this handler.

Type
coroutine function

pattern

Optional. Regex pattern, callback or type to test *telegram.CallbackQuery.data* against.

Changed in version 13.6: Added support for arbitrary callback data.

Type
`re.Pattern` | `callable` | `type`

block

Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*.

Type
`bool`

check_update(*update*)

Determines whether an update should be passed to this handler's *callback*.

Parameters
update (*telegram.Update* | `object`) – Incoming update.

Returns
`bool`

collect_additional_context(*context*, *update*, *application*, *check_result*)

Add the result of `re.match(pattern, update.callback_query.data)` to *CallbackContext.matches* as list with one element.

telegram.ext.ChatJoinRequestHandler

class telegram.ext.**ChatJoinRequestHandler**(*callback*, *block=True*)

Bases: *telegram.ext.BaseHandler*

BaseHandler class to handle Telegram updates that contain *telegram.Update.chat_join_request*.

> **Warning:** When setting *block* to `False`, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

New in version 13.8.

Parameters

- ***callback*** (coroutine function) – The callback function for this handler. Will be called when *check_update()* has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

  The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.

- ***block*** (bool, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*. Defaults to True.

callback

The callback function for this handler.

Type
coroutine function

block

Determines whether the callback will run in a blocking way..

Type
bool

check_update(*update*)

Determines whether an update should be passed to this handler's *callback*.

Parameters
update (*telegram.Update* | object) – Incoming update.

Returns
bool

telegram.ext.ChatMemberHandler

class telegram.ext.**ChatMemberHandler**(*callback*, *chat_member_types=-1*, *block=True*)

Bases: *telegram.ext.BaseHandler*

BaseHandler class to handle Telegram updates that contain a chat member update.

Warning: When setting *block* to False, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

See also:

Chat Member Example

New in version 13.4.

Parameters

- ***callback*** (coroutine function) – The callback function for this handler. Will be called when *check_update()* has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

  The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.

- ***chat_member_types*** (int, optional) – Pass one of *MY_CHAT_MEMBER*, *CHAT_MEMBER* or *ANY_CHAT_MEMBER* to specify if this handler should handle only updates with *telegram.Update.my_chat_member*, *telegram.Update.chat_member* or both. Defaults to *MY_CHAT_MEMBER*.
- ***block*** (bool, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*. Defaults to True.

callback

The callback function for this handler.

Type
coroutine function

chat_member_types

Specifies if this handler should handle only updates with *telegram.Update.my_chat_member*, *telegram.Update.chat_member* or both.

Type
int, optional

block

Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*.

Type
bool

ANY_CHAT_MEMBER = 1

Used as a constant to handle both *telegram.Update.my_chat_member* and *telegram.Update.chat_member*.

Type
int

CHAT_MEMBER = 0

Used as a constant to handle only *telegram.Update.chat_member*.

Type
int

MY_CHAT_MEMBER = -1

Used as a constant to handle only *telegram.Update.my_chat_member*.

Type
int

check_update(*update*)

Determines whether an update should be passed to this handler's *callback*.

Parameters
update (*telegram.Update* | object) – Incoming update.

Returns
bool

telegram.ext.ChosenInlineResultHandler

class telegram.ext.**ChosenInlineResultHandler**(*callback*, *block=True*, *pattern=None*)

Bases: *telegram.ext.BaseHandler*

BaseHandler class to handle Telegram updates that contain *telegram.Update.chosen_inline_result*.

Warning: When setting *block* to `False`, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

Parameters

- ***callback*** (coroutine function) – The callback function for this handler. Will be called when *check_update()* has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

  The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.

- ***block*** (`bool`, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*. Defaults to `True`.

- ***pattern*** (`str` | `re.Pattern`, optional) – Regex pattern. If not `None`, `re.match()` is used on *telegram.ChosenInlineResult.result_id* to determine if an update should be handled by this handler. This is accessible in the callback as *telegram.ext.CallbackContext.matches*.

  New in version 13.6.

callback

The callback function for this handler.

Type
coroutine function

block

Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*.

Type
`bool`

pattern

Optional. Regex pattern to test *telegram.ChosenInlineResult.result_id* against.

New in version 13.6.

Type
Pattern

check_update(*update*)

Determines whether an update should be passed to this handler's *callback*.

Parameters
update (*telegram.Update* | `object`) – Incoming update.

Returns
`bool` | `re.match`

collect_additional_context(*context*, *update*, *application*, *check_result*)

This function adds the matched regex pattern result to *telegram.ext.CallbackContext.matches*.

telegram.ext.CommandHandler

class telegram.ext.**CommandHandler**(*command*, *callback*, *filters=None*, *block=True*)

Bases: *telegram.ext.BaseHandler*

BaseHandler class to handle Telegram commands.

Commands are Telegram messages that start with `/`, optionally followed by an `@` and the bot's name and/or some additional text. The handler will add a `list` to the *CallbackContext* named *CallbackContext.args*. It will contain a list of strings, which is the text following the command split on single or consecutive whitespace characters.

By default, the handler listens to messages as well as edited messages. To change this behavior use *~filters.UpdateType.EDITED_MESSAGE* in the filter argument.

Note: *CommandHandler* does *not* handle (edited) channel posts and does *not* handle commands that are part of a caption. Please use *MessageHandler* with a suitable combination of filters (e.g. *telegram.ext.filters.UpdateType.CHANNEL_POSTS*, *telegram.ext.filters.CAPTION* and *telegram.ext.filters.Regex*) to handle those messages.

Warning: When setting *block* to `False`, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

Changed in version 20.0:

- Renamed the attribute `command` to *commands*, which now is always a `frozenset`
- Updating the commands this handler listens to is no longer possible.

Parameters

- ***command*** (`str` | Collection[`str`]) – The command or list of commands this handler should listen for. Case-insensitive. Limitations are the same as described here
- ***callback*** (coroutine function) – The callback function for this handler. Will be called when *check_update()* has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

  The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.

- ***filters*** (*telegram.ext.filters.BaseFilter*, optional) – A filter inheriting from *telegram.ext.filters.BaseFilter*. Standard filters can be found in *telegram.ext.filters*. Filters can be combined using bitwise operators (& for `and`, | for `or`, ~ for `not`)
- ***block*** (`bool`, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*. Defaults to `True`.

Raises

ValueError – When the command is too long or has illegal chars.

commands

The set of commands this handler should listen for.

Type
FrozenSet[str]

callback

The callback function for this handler.

Type
coroutine function

filters

Optional. Only allow updates with these Filters.

Type
telegram.ext.filters.BaseFilter

block

Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*.

Type
bool

check_update(*update*)

Determines whether an update should be passed to this handler's *callback*.

Parameters
update (*telegram.Update* | object) – Incoming update.

Returns
The list of args for the handler.

Return type
list

collect_additional_context(*context*, *update*, *application*, *check_result*)

Add text after the command to *CallbackContext.args* as list, split on single whitespaces and add output of data filters to *CallbackContext* as well.

telegram.ext.ConversationHandler

class telegram.ext.**ConversationHandler**(*entry_points*, *states*, *fallbacks*, *allow_reentry=False*, *per_chat=True*, *per_user=True*, *per_message=False*, *conversation_timeout=None*, *name=None*, *persistent=False*, *map_to_parent=None*, *block=True*)

Bases: *telegram.ext.BaseHandler*

A handler to hold a conversation with a single or multiple users through Telegram updates by managing three collections of other handlers.

Warning: *ConversationHandler* heavily relies on incoming updates being processed one by one. When using this handler, *telegram.ext.Application.concurrent_updates* should be False.

Note: *ConversationHandler* will only accept updates that are (subclass-)instances of *telegram.Update*. This is, because depending on the *per_user* and *per_chat*, *ConversationHandler* relies on *telegram.Update.effective_user* and/or *telegram.Update.effective_chat* in order to determine which conversation an update should belong to. For *per_message=True*, *ConversationHandler* uses *update.callback_query.message.message_id* when *per_chat=True*

and *update.callback_query.inline_message_id* when *per_chat=False*. For a more detailed explanation, please see our FAQ.

Finally, *ConversationHandler*, does *not* handle (edited) channel posts.

The first collection, a `list` named *entry_points*, is used to initiate the conversation, for example with a *telegram.ext.CommandHandler* or *telegram.ext.MessageHandler*.

The second collection, a `dict` named *states*, contains the different conversation steps and one or more associated handlers that should be used if the user sends a message when the conversation with them is currently in that state. Here you can also define a state for *TIMEOUT* to define the behavior when *conversation_timeout* is exceeded, and a state for *WAITING* to define behavior when a new update is received while the previous *block=False* handler is not finished.

The third collection, a `list` named *fallbacks*, is used if the user is currently in a conversation but the state has either no associated handler or the handler that is associated to the state is inappropriate for the update, for example if the update contains a command, but a regular text message is expected. You could use this for a `/cancel` command or to let the user know their message was not recognized.

To change the state of conversation, the callback function of a handler must return the new state after responding to the user. If it does not return anything (returning `None` by default), the state will not change. If an entry point callback function returns `None`, the conversation ends immediately after the execution of this callback function. To end the conversation, the callback function must return *END* or `-1`. To handle the conversation timeout, use handler *TIMEOUT* or `-2`. Finally, *telegram.ext.ApplicationHandlerStop* can be used in conversations as described in its documentation.

Note: In each of the described collections of handlers, a handler may in turn be a *ConversationHandler*. In that case, the child *ConversationHandler* should have the attribute *map_to_parent* which allows returning to the parent conversation at specified states within the child conversation.

Note that the keys in *map_to_parent* must not appear as keys in *states* attribute or else the latter will be ignored. You may map *END* to one of the parents states to continue the parent conversation after the child conversation has ended or even map a state to *END* to end the *parent* conversation from within the child conversation. For an example on nested *ConversationHandler* s, see *nestedconversationbot.py*.

See also:

Conversation Example, Conversation Example 2, Nested Conversation Example, Persistent Conversation Example

Parameters

- ***entry_points*** (List[*telegram.ext.BaseHandler*]) – A list of *BaseHandler* objects that can trigger the start of the conversation. The first handler whose *check_update()* method returns `True` will be used. If all return `False`, the update is not handled.
- ***states*** (Dict[`object`, List[*telegram.ext.BaseHandler*]]) – A `dict` that defines the different states of conversation a user can be in and one or more associated *BaseHandler* objects that should be used in that state. The first handler whose *check_update()* method returns `True` will be used.
- ***fallbacks*** (List[*telegram.ext.BaseHandler*]) – A list of handlers that might be used if the user is in a conversation, but every handler for their current state returned `False` on *check_update()*. The first handler which *check_update()* method returns `True` will be used. If all return `False`, the update is not handled.
- ***allow_reentry*** (`bool`, optional) – If set to `True`, a user that is currently in a conversation can restart the conversation by triggering one of the entry points.
- ***per_chat*** (`bool`, optional) – If the conversation key should contain the Chat's ID. Default is `True`.

- ***per_user*** (bool, optional) – If the conversation key should contain the User's ID. Default is True.
- ***per_message*** (bool, optional) – If the conversation key should contain the Message's ID. Default is False.
- ***conversation_timeout*** (float | datetime.timedelta, optional) – When this handler is inactive more than this timeout (in seconds), it will be automatically ended. If this value is 0 or None (default), there will be no timeout. The last received update and the corresponding *context* will be handled by *ALL* the handler's whose *check_update()* method returns True that are in the state *ConversationHandler.TIMEOUT*.

  Note: Using *conversation_timeout* with nested conversations is currently not supported. You can still try to use it, but it will likely behave differently from what you expect.

- ***name*** (str, optional) – The name for this conversation handler. Required for persistence.
- ***persistent*** (bool, optional) – If the conversation's dict for this handler should be saved. *name* is required and persistence has to be set in *Application*.

  Changed in version 20.0: Was previously named as `persistence`.

- ***map_to_parent*** (Dict[object, object], optional) – A dict that can be used to instruct a child conversation handler to transition into a mapped state on its parent conversation handler in place of a specified nested state.
- ***block*** (bool, optional) – Pass False or True to set a default value for the *BaseHandler.block* setting of all handlers (in *entry_points*, *states* and *fallbacks*). The resolution order for checking if a handler should be run non-blocking is:

  1. *telegram.ext.BaseHandler.block* (if set)
  2. the value passed to this parameter (if any)
  3. *telegram.ext.Defaults.block* (if defaults are used)

  Changed in version 20.0: No longer overrides the handlers settings. Resolution order was changed.

Raises

ValueError – If *persistent* is used but *name* was not set, or when *per_message*, *per_chat*, *per_user* are all False.

block

Determines whether the callback will run in a blocking way.. Always True since conversation handlers handle any non-blocking callbacks internally.

Type

bool

END = -1

Used as a constant to return when a conversation is ended.

Type

int

TIMEOUT = -2

Used as a constant to handle state when a conversation is timed out (exceeded *conversation_timeout*).

Type

int

WAITING = -3

Used as a constant to handle state when a conversation is still waiting on the previous *block=False* handler to finish.

Type
int

property allow_reentry

Determines if a user can restart a conversation with an entry point.

Type
bool

check_update(*update*)

Determines whether an update should be handled by this conversation handler, and if so in which state the conversation currently is.

Parameters
update (*telegram.Update* | object) – Incoming update.

Returns
bool

property conversation_timeout

Optional. When this handler is inactive more than this timeout (in seconds), it will be automatically ended.

Type
float | datetime.timedelta

property entry_points

A list of *BaseHandler* objects that can trigger the start of the conversation.

Type
List[*telegram.ext.BaseHandler*]

property fallbacks

A list of handlers that might be used if the user is in a conversation, but every handler for their current state returned False on *check_update()*.

Type
List[*telegram.ext.BaseHandler*]

async handle_update(*update*, *application*, *check_result*, *context*)

Send the update to the callback for the current state and BaseHandler

Parameters

- ***check_result*** – The result from *check_update()*. For this handler it's a tuple of the conversation state, key, handler, and the handler's check result.
- ***update*** (*telegram.Update*) – Incoming telegram update.
- ***application*** (*telegram.ext.Application*) – Application that originated the update.
- ***context*** (*telegram.ext.CallbackContext*) – The context as provided by the application.

property map_to_parent

Optional. A dict that can be used to instruct a nested *ConversationHandler* to transition into a mapped state on its parent *ConversationHandler* in place of a specified nested state.

Type
Dict[object, object]

property name

Optional. The name for this *ConversationHandler*.

Type
str

property per_chat

If the conversation key should contain the Chat's ID.

Type
bool

property per_message

If the conversation key should contain the message's ID.

Type
bool

property per_user

If the conversation key should contain the User's ID.

Type
bool

property persistent

Optional. If the conversations dict for this handler should be saved. *name* is required and persistence has to be set in *Application*.

Type
bool

property states

A dict that defines the different states of conversation a user can be in and one or more associated *BaseHandler* objects that should be used in that state.

Type
Dict[object, List[*telegram.ext.BaseHandler*]]

telegram.ext.filters Module

This module contains filters for use with *telegram.ext.MessageHandler*, *telegram.ext.CommandHandler*, or *telegram.ext.PrefixHandler*.

Changed in version 20.0:

1. Filters are no longer callable, if you're using a custom filter and are calling an existing filter, then switch to the new syntax: `filters.{filter}.check_update(update)`.
2. Removed the `Filters` class. The filters are now directly attributes/classes of the *filters* module.
3. The names of all filters has been updated:
   - Filter classes which are ready for use, e.g `Filters.all` are now capitalized, e.g `filters.ALL`.
   - Filters which need to be initialized are now in CamelCase. E.g. `filters.User(...)`.
   - Filters which do both (like `Filters.text`) are now split as ready-to-use version `filters.TEXT` and class version `filters.Text(...)`.

telegram.ext.filters.**ALL = filters.ALL**

All Messages.

telegram.ext.filters.**ANIMATION = filters.ANIMATION**

Messages that contain *telegram.Message.animation*.

telegram.ext.filters.**ATTACHMENT = filters.ATTACHMENT**

Messages that contain *telegram.Message.effective_attachment()*.

New in version 13.6.

telegram.ext.filters.**AUDIO = filters.AUDIO**

Messages that contain *telegram.Message.audio*.

class telegram.ext.filters.**BaseFilter**(*name=None*, *data_filter=False*)

Bases: object

Base class for all Filters.

Filters subclassing from this class can combined using bitwise operators:

And:

```
filters.TEXT & filters.Entity(MENTION)
```

Or:

```
filters.AUDIO | filters.VIDEO
```

Exclusive Or:

```
filters.Regex('To Be') ^ filters.Regex('Not 2B')
```

Not:

```
~ filters.COMMAND
```

Also works with more than two filters:

```
filters.TEXT & (filters.Entity(URL) | filters.Entity(TEXT_LINK))
filters.TEXT & (~ filters.FORWARDED)
```

Note: Filters use the same short circuiting logic as python's and, or and not. This means that for example:

```
filters.Regex(r'(a?x)') | filters.Regex(r'(b?x)')
```

With `message.text == 'x'`, will only ever return the matches for the first filter, since the second one is never evaluated.

If you want to create your own filters create a class inheriting from either *MessageFilter* or *UpdateFilter* and implement a `filter()` method that returns a boolean: `True` if the message should be handled, `False` otherwise. Note that the filters work only as class instances, not actual class objects (so remember to initialize your filter classes).

By default, the filters name (what will get printed when converted to a string for display) will be the class name. If you want to overwrite this assign a better name to the *name* class variable.

New in version 20.0: Added the arguments *name* and *data_filter*.

Parameters

- ***name*** (str) – Name for this filter. Defaults to the type of filter.
- ***data_filter*** (bool) – Whether this filter is a data filter. A data filter should return a dict with lists. The dict will be merged with *telegram.ext.CallbackContext*'s internal dict in most cases (depends on the handler).

name

Name for this filter.

Type
str

data_filter

Whether this filter is a data filter.

Type
bool

check_update(*update*)

Checks if the specified update is a message.

telegram.ext.filters.**CAPTION = filters.CAPTION**

Shortcut for *telegram.ext.filters.Caption()*.

Examples

To allow any caption, simply use `MessageHandler(filters.CAPTION, callback_method)`.

---

telegram.ext.filters.**CHAT = filters.CHAT**

This filter filters *any* message that has a *telegram.Message.chat*.

telegram.ext.filters.**COMMAND = filters.COMMAND**

Shortcut for *telegram.ext.filters.Command()*.

Examples

To allow messages starting with a command use `MessageHandler(filters.COMMAND, command_at_start_callback)`.

---

telegram.ext.filters.**CONTACT = filters.CONTACT**

Messages that contain *telegram.Message.contact*.

class telegram.ext.filters.**Caption**(*strings=None*)

Bases: *telegram.ext.filters.MessageFilter*

Messages with a caption. If a list of strings is passed, it filters messages to only allow those whose caption is appearing in the given list.

Examples

```
MessageHandler(filters.Caption(['PTB rocks!', 'PTB'], callback_method_2)
```

See also:

telegram.ext.filters.CAPTION

Parameters

strings (List[str] | Tuple[str], optional) – Which captions to allow. Only exact matches are allowed. If not specified, will allow any message with a caption.

class telegram.ext.filters.**CaptionEntity**(*entity_type*)

Bases: *telegram.ext.filters.MessageFilter*

Filters media messages to only allow those which have a *telegram.MessageEntity* where their *type* matches *entity_type*.

Examples

```
MessageHandler(filters.CaptionEntity("hashtag"), callback_method)
```

Parameters

entity_type (str) – Caption Entity type to check for. All types can be found as constants in *telegram.MessageEntity*.

class telegram.ext.filters.**CaptionRegex**(*pattern*)

Bases: *telegram.ext.filters.MessageFilter*

Filters updates by searching for an occurrence of *pattern* in the message caption.

This filter works similarly to *Regex*, with the only exception being that it applies to the message caption instead of the text.

Examples

Use `MessageHandler(filters.PHOTO & filters.CaptionRegex(r'help'), callback)` to capture all photos with caption containing the word 'help'.

Note: This filter will not work on simple text messages, but only on media with caption.

Parameters

pattern (str | re.Pattern) – The regex pattern.

class telegram.ext.filters.**Chat**(*chat_id=None*, *username=None*, *allow_empty=False*)

Bases: *telegram.ext.filters.MessageFilter*

Filters messages to allow only those which are from a specified chat ID or username.

Examples

```
MessageHandler(filters.Chat(-1234), callback_method)
```

Warning: *chat_ids* will give a *copy* of the saved chat ids as `frozenset`. This is to ensure thread safety. To add/remove a chat, you should use *add_chat_ids()*, and *remove_chat_ids()*. Only update the entire set by `filter.chat_ids = new_set`, if you are entirely sure that it is not causing race conditions, as this will complete replace the current set of allowed chats.

Parameters

- ***chat_id*** (int | Collection[int], optional) – Which chat ID(s) to allow through.
- ***username*** (str | Collection[str], optional) – Which username(s) to allow through. Leading `'@'` s in usernames will be discarded.
- ***allow_empty*** (bool, optional) – Whether updates should be processed, if no chat is specified in *chat_ids* and *usernames*. Defaults to `False`.

chat_ids

Which chat ID(s) to allow through.

Type
set(int)

allow_empty

Whether updates should be processed, if no chat is specified in *chat_ids* and *usernames*.

Type
bool

Raises
RuntimeError – If chat_id and username are both present.

add_chat_ids(*chat_id*)

Add one or more chats to the allowed chat ids.

Parameters
chat_id (int | Collection[int]) – Which chat ID(s) to allow through.

remove_chat_ids(*chat_id*)

Remove one or more chats from allowed chat ids.

Parameters
chat_id (int | Collection[int]) – Which chat ID(s) to disallow through.

add_usernames(*username*)

Add one or more chats to the allowed usernames.

Parameters
username (str | Collection[str]) – Which username(s) to allow through. Leading '@' s in usernames will be discarded.

remove_usernames(*username*)

Remove one or more chats from allowed usernames.

Parameters
username (str | Collection[str]) – Which username(s) to disallow through. Leading '@' s in usernames will be discarded.

property usernames

Which username(s) to allow through.

> **Warning:** *usernames* will give a *copy* of the saved usernames as frozenset. This is to ensure thread safety. To add/remove a user, you should use *add_usernames()*, and *remove_usernames()*. Only update the entire set by filter.usernames = new_set, if you are entirely sure that it is not causing race conditions, as this will complete replace the current set of allowed users.

Returns
frozenset(str)

class telegram.ext.filters.**ChatType**

Bases: object

Subset for filtering the type of chat.

Examples

Use these filters like: filters.ChatType.CHANNEL or filters.ChatType.SUPERGROUP etc.

Caution: `filters.ChatType` itself is *not* a filter, but just a convenience namespace.

CHANNEL = filters.ChatType.CHANNEL

Updates from channel.

GROUP = filters.ChatType.GROUP

Updates from group.

GROUPS = filters.ChatType.GROUPS

Update from group *or* supergroup.

PRIVATE = filters.ChatType.PRIVATE

Update from private chats.

SUPERGROUP = filters.ChatType.SUPERGROUP

Updates from supergroup.

class telegram.ext.filters.**Command**(*only_start=True*)

Bases: *telegram.ext.filters.MessageFilter*

Messages with a *telegram.MessageEntity.BOT_COMMAND*. By default, only allows messages *starting* with a bot command. Pass `False` to also allow messages that contain a bot command *anywhere* in the text.

Examples

```
MessageHandler(filters.Command(False), command_anywhere_callback)
```

See also:

telegram.ext.filters.COMMAND.

Note: *telegram.ext.filters.TEXT* also accepts messages containing a command.

Parameters

only_start (`bool`, optional) – Whether to only allow messages that *start* with a bot command. Defaults to `True`.

class telegram.ext.filters.**Dice**(*values=None*, *emoji=None*)

Bases: *telegram.ext.filters.MessageFilter*

Dice Messages. If an integer or a list of integers is passed, it filters messages to only allow those whose dice value is appearing in the given list.

New in version 13.4.

Examples

To allow any dice message, simply use `MessageHandler(filters.Dice.ALL, callback_method)`.

To allow any dice message, but with value 3 *or* 4, use `MessageHandler(filters.Dice([3, 4]), callback_method)`

To allow only dice messages with the emoji , but any value, use `MessageHandler(filters.Dice.DICE, callback_method)`.

To allow only dice messages with the emoji and with value 6, use `MessageHandler(filters.Dice.Darts(6), callback_method)`.

To allow only dice messages with the emoji and with value 5 *or* 6, use `MessageHandler(filters.Dice.Football([5, 6]), callback_method)`.

Note: Dice messages don't have text. If you want to filter either text or dice messages, use `filters.TEXT | filters.Dice.ALL`.

Parameters

values (`int` | Collection[`int`], optional) – Which values to allow. If not specified, will allow the specified dice message.

ALL = filters.Dice.ALL

Dice messages with any value and any emoji.

class Basketball(*values*)

Bases: *telegram.ext.filters.MessageFilter*

Dice messages with the emoji . Supports passing a list of integers.

Parameters

values (`int` | Collection[`int`]) – Which values to allow.

BASKETBALL = filters.Dice.BASKETBALL

Dice messages with the emoji . Matches any dice value.

class Bowling(*values*)

Bases: *telegram.ext.filters.MessageFilter*

Dice messages with the emoji . Supports passing a list of integers.

Parameters

values (`int` | Collection[`int`]) – Which values to allow.

BOWLING = filters.Dice.BOWLING

Dice messages with the emoji . Matches any dice value.

class Darts(*values*)

Bases: *telegram.ext.filters.MessageFilter*

Dice messages with the emoji . Supports passing a list of integers.

Parameters

values (`int` | Collection[`int`]) – Which values to allow.

DARTS = filters.Dice.DARTS

Dice messages with the emoji . Matches any dice value.

class Dice(*values*)

Bases: *telegram.ext.filters.MessageFilter*

Dice messages with the emoji . Supports passing a list of integers.

Parameters

values (`int` | Collection[`int`]) – Which values to allow.

DICE = filters.Dice.DICE

Dice messages with the emoji . Matches any dice value.

class Football(*values*)

Bases: *telegram.ext.filters.MessageFilter*

Dice messages with the emoji . Supports passing a list of integers.

Parameters
values (int | Collection[int]) – Which values to allow.

FOOTBALL = filters.Dice.FOOTBALL

Dice messages with the emoji . Matches any dice value.

class SlotMachine(*values*)

Bases: *telegram.ext.filters.MessageFilter*

Dice messages with the emoji . Supports passing a list of integers.

Parameters
values (int | Collection[int]) – Which values to allow.

SLOT_MACHINE = filters.Dice.SLOT_MACHINE

Dice messages with the emoji . Matches any dice value.

class telegram.ext.filters.**Document**

Bases: object

Subset for messages containing a document/file.

Examples

Use these filters like: `filters.Document.MP3`, `filters.Document.MimeType("text/plain")` etc. Or just use `filters.Document.ALL` for all document messages.

Caution: `filters.Document` itself is *not* a filter, but just a convenience namespace.

ALL = filters.Document.ALL

Messages that contain a *telegram.Message.document*.

class Category(*category*)

Bases: *telegram.ext.filters.MessageFilter*

Filters documents by their category in the mime-type attribute.

Parameters
category (str) – Category of the media you want to filter.

Example

`filters.Document.Category('audio/')` returns `True` for all types of audio sent as a file, for example `'audio/mpeg'` or `'audio/x-wav'`.

Note: This Filter only filters by the mime_type of the document, it doesn't check the validity of the document. The user can manipulate the mime-type of a message and send media with wrong types that don't fit to this handler.

APPLICATION = filters.Document.Category('application/')

Use as `filters.Document.APPLICATION`.

AUDIO = filters.Document.Category('audio/')

Use as `filters.Document.AUDIO`.

IMAGE = filters.Document.Category('image/')

Use as `filters.Document.IMAGE`.

VIDEO = filters.Document.Category('video/')

Use as `filters.Document.VIDEO`.

TEXT = filters.Document.Category('text/')

Use as `filters.Document.TEXT`.

class FileExtension(*file_extension*, *case_sensitive=False*)

Bases: *telegram.ext.filters.MessageFilter*

This filter filters documents by their file ending/extension.

Parameters

- ***file_extension*** (`str` | `None`) – Media file extension you want to filter.
- ***case_sensitive*** (`bool`, optional) – Pass `True` to make the filter case sensitive. Default: `False`.

Example

- `filters.Document.FileExtension("jpg")` filters files with extension `".jpg"`.
- `filters.Document.FileExtension(".jpg")` filters files with extension `"..jpg"`.
- `filters.Document.FileExtension("Dockerfile", case_sensitive=True)` filters files with extension `".Dockerfile"` minding the case.
- `filters.Document.FileExtension(None)` filters files without a dot in the filename.

Note:

- This Filter only filters by the file ending/extension of the document, it doesn't check the validity of document.
- The user can manipulate the file extension of a document and send media with wrong types that don't fit to this handler.
- Case insensitive by default, you may change this with the flag `case_sensitive=True`.
- Extension should be passed without leading dot unless it's a part of the extension.
- Pass `None` to filter files with no extension, i.e. without a dot in the filename.

---

class MimeType(*mimetype*)

Bases: *telegram.ext.filters.MessageFilter*

This Filter filters documents by their mime-type attribute.

Parameters

mimetype (`str`) – The mimetype to filter.

Example

`filters.Document.MimeType('audio/mpeg')` filters all audio in *.mp3* format.

Note: This Filter only filters by the mime_type of the document, it doesn't check the validity of document. The user can manipulate the mime-type of a message and send media with wrong types that don't fit to this handler.

APK = filters.Document.MimeType('application/vnd.android.package-archive')

Use as `filters.Document.APK`.

DOC = filters.Document.MimeType('application/msword')

Use as `filters.Document.DOC`.

DOCX = filters.Document.MimeType('application/vnd.openxmlformats-officedocument.wordprocessingml.document')

Use as `filters.Document.DOCX`.

EXE = filters.Document.MimeType('application/octet-stream')

Use as `filters.Document.EXE`.

MP4 = filters.Document.MimeType('video/mp4')

Use as `filters.Document.MP4`.

GIF = filters.Document.MimeType('image/gif')

Use as `filters.Document.GIF`.

JPG = filters.Document.MimeType('image/jpeg')

Use as `filters.Document.JPG`.

MP3 = filters.Document.MimeType('audio/mpeg')

Use as `filters.Document.MP3`.

PDF = filters.Document.MimeType('application/pdf')

Use as `filters.Document.PDF`.

PY = filters.Document.MimeType('text/x-python')

Use as `filters.Document.PY`.

SVG = filters.Document.MimeType('image/svg+xml')

Use as `filters.Document.SVG`.

TXT = filters.Document.MimeType('text/plain')

Use as `filters.Document.TXT`.

TARGZ = filters.Document.MimeType('application/x-compressed-tar')

Use as `filters.Document.TARGZ`.

WAV = filters.Document.MimeType('audio/x-wav')

Use as `filters.Document.WAV`.

XML = filters.Document.MimeType('text/xml')

Use as `filters.Document.XML`.

ZIP = filters.Document.MimeType('application/zip')

Use as `filters.Document.ZIP`.

class telegram.ext.filters.**Entity**(*entity_type*)

Bases: *telegram.ext.filters.MessageFilter*

Filters messages to only allow those which have a *telegram.MessageEntity* where their *type* matches *entity_type*.

Examples

```
MessageHandler(filters.Entity("hashtag"), callback_method)
```

Parameters

entity_type (str) – Entity type to check for. All types can be found as constants in *telegram.MessageEntity*.

telegram.ext.filters.**FORWARDED = filters.FORWARDED**

Messages that contain *telegram.Message.forward_date*.

class telegram.ext.filters.**ForwardedFrom**(*chat_id=None*, *username=None*, *allow_empty=False*)

Bases: *telegram.ext.filters.MessageFilter*

Filters messages to allow only those which are forwarded from the specified chat ID(s) or username(s) based on *telegram.Message.forward_from* and *telegram.Message.forward_from_chat*.

New in version 13.5.

Examples

```
MessageHandler(filters.ForwardedFrom(chat_id=1234), callback_method)
```

Note: When a user has disallowed adding a link to their account while forwarding their messages, this filter will *not* work since both *telegram.Message.forward_from* and *telegram.Message.forward_from_chat* are `None`. However, this behaviour is undocumented and might be changed by Telegram.

Warning: *chat_ids* will give a *copy* of the saved chat ids as `frozenset`. This is to ensure thread safety. To add/remove a chat, you should use *add_chat_ids()*, and *remove_chat_ids()*. Only update the entire set by `filter.chat_ids = new_set`, if you are entirely sure that it is not causing race conditions, as this will complete replace the current set of allowed chats.

Parameters

- ***chat_id*** (`int` | Collection[`int`], optional) – Which chat/user ID(s) to allow through.
- ***username*** (`str` | Collection[`str`], optional) – Which username(s) to allow through. Leading `'@'` s in usernames will be discarded.
- ***allow_empty*** (`bool`, optional) – Whether updates should be processed, if no chat is specified in *chat_ids* and *usernames*. Defaults to `False`.

chat_ids

Which chat/user ID(s) to allow through.

Type
set(`int`)

allow_empty

Whether updates should be processed, if no chat is specified in *chat_ids* and *usernames*.

Type
`bool`

Raises
RuntimeError – If both `chat_id` and `username` are present.

add_chat_ids(*chat_id*)

Add one or more chats to the allowed chat ids.

Parameters
chat_id (`int` | Collection[`int`]) – Which chat/user ID(s) to allow through.

remove_chat_ids(*chat_id*)

Remove one or more chats from allowed chat ids.

Parameters

chat_id (int | Collection[int]) – Which chat/user ID(s) to disallow through.

add_usernames(*username*)

Add one or more chats to the allowed usernames.

Parameters

username (str | Collection[str]) – Which username(s) to allow through. Leading '@' s in usernames will be discarded.

remove_usernames(*username*)

Remove one or more chats from allowed usernames.

Parameters

username (str | Collection[str]) – Which username(s) to disallow through. Leading '@' s in usernames will be discarded.

property usernames

Which username(s) to allow through.

> **Warning:** *usernames* will give a *copy* of the saved usernames as frozenset. This is to ensure thread safety. To add/remove a user, you should use *add_usernames()*, and *remove_usernames()*. Only update the entire set by `filter.usernames = new_set`, if you are entirely sure that it is not causing race conditions, as this will complete replace the current set of allowed users.

Returns

frozenset(str)

telegram.ext.filters.**GAME = filters.GAME**

Messages that contain *telegram.Message.game*.

telegram.ext.filters.**HAS_PROTECTED_CONTENT = filters.HAS_PROTECTED_CONTENT**

Messages that contain *telegram.Message.has_protected_content*.

New in version 13.9.

telegram.ext.filters.**INVOICE = filters.INVOICE**

Messages that contain *telegram.Message.invoice*.

telegram.ext.filters.**IS_AUTOMATIC_FORWARD = filters.IS_AUTOMATIC_FORWARD**

Messages that contain *telegram.Message.is_automatic_forward*.

New in version 13.9.

telegram.ext.filters.**LOCATION = filters.LOCATION**

Messages that contain *telegram.Message.location*.

class telegram.ext.filters.**Language**(*lang*)

Bases: *telegram.ext.filters.MessageFilter*

Filters messages to only allow those which are from users with a certain language code.

> **Note:** According to official Telegram Bot API documentation, not every single user has the *language_code* attribute. Do not count on this filter working on all users.

Examples

```
MessageHandler(filters.Language("en"), callback_method)
```

Parameters

lang (str | Collection[str]) – Which language code(s) to allow through. This will be matched using `str.startswith` meaning that 'en' will match both 'en_US' and 'en_GB'.

class telegram.ext.filters.**MessageFilter**(*name=None*, *data_filter=False*)

Bases: *telegram.ext.filters.BaseFilter*

Base class for all Message Filters. In contrast to *UpdateFilter*, the object passed to *filter()* is *telegram.Update.effective_message*.

Please see *BaseFilter* for details on how to create custom filters.

name

Name for this filter. Defaults to the type of filter.

Type
str

data_filter

Whether this filter is a data filter. A data filter should return a dict with lists. The dict will be merged with *telegram.ext.CallbackContext*'s internal dict in most cases (depends on the handler).

Type
bool

check_update(*update*)

Checks if the specified update is a message.

abstract filter(*message*)

This method must be overwritten.

Parameters
message (*telegram.Message*) – The message that is tested.

Returns
dict or bool

telegram.ext.filters.**PASSPORT_DATA = filters.PASSPORT_DATA**

Messages that contain *telegram.Message.passport_data*.

telegram.ext.filters.**PHOTO = filters.PHOTO**

Messages that contain *telegram.Message.photo*.

telegram.ext.filters.**POLL = filters.POLL**

Messages that contain *telegram.Message.poll*.

telegram.ext.filters.**REPLY = filters.REPLY**

Messages that contain *telegram.Message.reply_to_message*.

class telegram.ext.filters.**Regex**(*pattern*)

Bases: *telegram.ext.filters.MessageFilter*

Filters updates by searching for an occurrence of *pattern* in the message text. The `re.search()` function is used to determine whether an update should be filtered.

Refer to the documentation of the `re` module for more information.

To get the groups and groupdict matched, see *telegram.ext.CallbackContext.matches*.

Examples

Use `MessageHandler(filters.Regex(r'help'), callback)` to capture all messages that contain the word 'help'. You can also use `MessageHandler(filters.Regex(re.compile(r'help', re.IGNORECASE)), callback)` if you want your pattern to be case insensitive. This approach is recommended if you need to specify flags on your pattern.

Note: Filters use the same short circuiting logic as python's `and`, `or` and `not`. This means that for example:

```
>>> filters.Regex(r'(a?x)') | filters.Regex(r'(b?x)')
```

With a *telegram.Message.text* of *x*, will only ever return the matches for the first filter, since the second one is never evaluated.

Parameters

pattern (`str` | `re.Pattern`) – The regex pattern.

class telegram.ext.filters.**Sticker**

Bases: `object`

Filters messages which contain a sticker.

Examples

Use this filter like: `filters.Sticker.VIDEO`. Or, just use `filters.Sticker.ALL` for any type of sticker.

Caution: `filters.Sticker` itself is *not* a filter, but just a convenience namespace.

ALL = filters.Sticker.ALL

Messages that contain *telegram.Message.sticker*.

ANIMATED = filters.Sticker.ANIMATED

Messages that contain *telegram.Message.sticker* and *is animated*.

New in version 20.0.

STATIC = filters.Sticker.STATIC

Messages that contain *telegram.Message.sticker* and is a static sticker, i.e. does not contain *telegram.Sticker.is_animated* or *telegram.Sticker.is_video*.

New in version 20.0.

VIDEO = filters.Sticker.VIDEO

Messages that contain *telegram.Message.sticker* and is a *video sticker*.

New in version 20.0.

PREMIUM = filters.Sticker.PREMIUM

Messages that contain *telegram.Message.sticker* and have a *premium animation*.

New in version 20.0.

telegram.ext.filters.**SUCCESSFUL_PAYMENT = filters.SUCCESSFUL_PAYMENT**

Messages that contain *telegram.Message.successful_payment*.

class telegram.ext.filters.**SenderChat**(*chat_id=None*, *username=None*, *allow_empty=False*)

Bases: *telegram.ext.filters.MessageFilter*

Filters messages to allow only those which are from a specified sender chat's chat ID or username.

Examples

- To filter for messages sent to a group by a channel with ID `-1234`, use `MessageHandler(filters.SenderChat(-1234), callback_method)`.
- To filter for messages of anonymous admins in a super group with username `@anonymous`, use `MessageHandler(filters.SenderChat(username='anonymous'), callback_method)`.
- To filter for messages sent to a group by *any* channel, use `MessageHandler(filters.SenderChat.CHANNEL, callback_method)`.
- To filter for messages of anonymous admins in *any* super group, use `MessageHandler(filters.SenderChat.SUPERGROUP, callback_method)`.
- To filter for messages forwarded to a discussion group from *any* channel or of anonymous admins in *any* super group, use `MessageHandler(filters.SenderChat.ALL, callback)`

Note: Remember, `sender_chat` is also set for messages in a channel as the channel itself, so when your bot is an admin in a channel and the linked discussion group, you would receive the message twice (once from inside the channel, once inside the discussion group). Since v13.9, the field *telegram.Message.is_automatic_forward* will be `True` for the discussion group message.

See also:

telegram.ext.filters.IS_AUTOMATIC_FORWARD

Warning: *chat_ids* will return a *copy* of the saved chat ids as `frozenset`. This is to ensure thread safety. To add/remove a chat, you should use *add_chat_ids()*, and *remove_chat_ids()*. Only update the entire set by `filter.chat_ids = new_set`, if you are entirely sure that it is not causing race conditions, as this will complete replace the current set of allowed chats.

Parameters

- ***chat_id*** (`int` | Collection[`int`], optional) – Which sender chat chat ID(s) to allow through.
- ***username*** (`str` | Collection[`str`], optional) – Which sender chat username(s) to allow through. Leading `'@'` s in usernames will be discarded.
- ***allow_empty*** (`bool`, optional) – Whether updates should be processed, if no sender chat is specified in *chat_ids* and *usernames*. Defaults to `False`.

chat_ids

Which sender chat chat ID(s) to allow through.

Type
set(`int`)

allow_empty

Whether updates should be processed, if no sender chat is specified in *chat_ids* and *usernames*.

Type
`bool`

Raises
`RuntimeError` – If both `chat_id` and `username` are present.

ALL = filters.SenderChat.ALL

All messages with a *telegram.Message.sender_chat*.

SUPER_GROUP = filters.SenderChat.SUPER_GROUP

Messages whose sender chat is a super group.

CHANNEL = filters.SenderChat.CHANNEL

Messages whose sender chat is a channel.

add_chat_ids(*chat_id*)

Add one or more sender chats to the allowed chat ids.

Parameters

chat_id (int | Collection[int]) – Which sender chat ID(s) to allow through.

remove_chat_ids(*chat_id*)

Remove one or more sender chats from allowed chat ids.

Parameters

chat_id (int | Collection[int]) – Which sender chat ID(s) to disallow through.

add_usernames(*username*)

Add one or more chats to the allowed usernames.

Parameters

username (str | Collection[str]) – Which username(s) to allow through. Leading '@' s in usernames will be discarded.

remove_usernames(*username*)

Remove one or more chats from allowed usernames.

Parameters

username (str | Collection[str]) – Which username(s) to disallow through. Leading '@' s in usernames will be discarded.

property usernames

Which username(s) to allow through.

> **Warning:** *usernames* will give a *copy* of the saved usernames as frozenset. This is to ensure thread safety. To add/remove a user, you should use *add_usernames()*, and *remove_usernames()*. Only update the entire set by filter.usernames = new_set, if you are entirely sure that it is not causing race conditions, as this will complete replace the current set of allowed users.

Returns

frozenset(str)

class telegram.ext.filters.**StatusUpdate**

Bases: object

Subset for messages containing a status update.

Examples

Use these filters like: filters.StatusUpdate.NEW_CHAT_MEMBERS etc. Or use just filters.StatusUpdate.ALL for all status update messages.

> **Caution:** filters.StatusUpdate itself is *not* a filter, but just a convenience namespace.

ALL = filters.StatusUpdate.ALL

Messages that contain any of the below.

CHAT_CREATED = filters.StatusUpdate.CHAT_CREATED

Messages that contain *telegram.Message.group_chat_created*, *telegram.Message.supergroup_chat_created* or *telegram.Message.channel_chat_created*.

CONNECTED_WEBSITE = filters.StatusUpdate.CONNECTED_WEBSITE

Messages that contain *telegram.Message.connected_website*.

DELETE_CHAT_PHOTO = filters.StatusUpdate.DELETE_CHAT_PHOTO

Messages that contain *telegram.Message.delete_chat_photo*.

LEFT_CHAT_MEMBER = filters.StatusUpdate.LEFT_CHAT_MEMBER

Messages that contain *telegram.Message.left_chat_member*.

MESSAGE_AUTO_DELETE_TIMER_CHANGED = filters.StatusUpdate.MESSAGE_AUTO_DELETE_TIMER_CHANGED

Messages that contain *telegram.Message.message_auto_delete_timer_changed*

New in version 13.4.

MIGRATE = filters.StatusUpdate.MIGRATE

Messages that contain *telegram.Message.migrate_from_chat_id* or *telegram.Message.migrate_to_chat_id*.

NEW_CHAT_MEMBERS = filters.StatusUpdate.NEW_CHAT_MEMBERS

Messages that contain *telegram.Message.new_chat_members*.

NEW_CHAT_PHOTO = filters.StatusUpdate.NEW_CHAT_PHOTO

Messages that contain *telegram.Message.new_chat_photo*.

NEW_CHAT_TITLE = filters.StatusUpdate.NEW_CHAT_TITLE

Messages that contain *telegram.Message.new_chat_title*.

PINNED_MESSAGE = filters.StatusUpdate.PINNED_MESSAGE

Messages that contain *telegram.Message.pinned_message*.

PROXIMITY_ALERT_TRIGGERED = filters.StatusUpdate.PROXIMITY_ALERT_TRIGGERED

Messages that contain *telegram.Message.proximity_alert_triggered*.

VIDEO_CHAT_ENDED = filters.StatusUpdate.VIDEO_CHAT_ENDED

Messages that contain *telegram.Message.video_chat_ended*.

New in version 13.4.

Changed in version 20.0: This filter was formerly named VOICE_CHAT_ENDED

VIDEO_CHAT_SCHEDULED = filters.StatusUpdate.VIDEO_CHAT_SCHEDULED

Messages that contain *telegram.Message.video_chat_scheduled*.

New in version 13.5.

Changed in version 20.0: This filter was formerly named VOICE_CHAT_SCHEDULED

VIDEO_CHAT_STARTED = filters.StatusUpdate.VIDEO_CHAT_STARTED

Messages that contain *telegram.Message.video_chat_started*.

New in version 13.4.

Changed in version 20.0: This filter was formerly named VOICE_CHAT_STARTED

VIDEO_CHAT_PARTICIPANTS_INVITED = filters.StatusUpdate.VIDEO_CHAT_PARTICIPANTS_INVITED

Messages that contain *telegram.Message.video_chat_participants_invited*.

New in version 13.4.

Changed in version 20.0: This filter was formerly named VOICE_CHAT_PARTICIPANTS_INVITED

WEB_APP_DATA = filters.StatusUpdate.WEB_APP_DATA

Messages that contain *telegram.Message.web_app_data*.

New in version 20.0.

telegram.ext.filters.**TEXT = filters.TEXT**

Shortcut for *telegram.ext.filters.Text()*.

Examples

To allow any text message, simply use `MessageHandler(filters.TEXT, callback_method)`.

---

class telegram.ext.filters.**Text**(*strings=None*)

Bases: *telegram.ext.filters.MessageFilter*

Text Messages. If a list of strings is passed, it filters messages to only allow those whose text is appearing in the given list.

Examples

A simple use case for passing a list is to allow only messages that were sent by a custom *telegram.ReplyKeyboardMarkup*:

```
buttons = ['Start', 'Settings', 'Back']
markup = ReplyKeyboardMarkup.from_column(buttons)
...
MessageHandler(filters.Text(buttons), callback_method)
```

See also:

telegram.ext.filters.TEXT

Note:

- Dice messages don't have text. If you want to filter either text or dice messages, use `filters.TEXT | filters.Dice.ALL`.
- Messages containing a command are accepted by this filter. Use `filters.TEXT & (~filters.COMMAND)`, if you want to filter only text messages without commands.

Parameters

strings (List[str] | Tuple[str], optional) – Which messages to allow. Only exact matches are allowed. If not specified, will allow any text message.

telegram.ext.filters.**USER = filters.USER**

This filter filters *any* message that has a *telegram.Message.from_user*.

telegram.ext.filters.**USER_ATTACHMENT = filters.USER_ATTACHMENT**

This filter filters *any* message that have a user who added the bot to their *attachment menu* as *telegram.Update.effective_user*.

New in version 20.0.

telegram.ext.filters.**PREMIUM_USER = filters.PREMIUM_USER**

This filter filters *any* message from a *Telegram Premium user* as *telegram.Update.effective_user*.

New in version 20.0.

class telegram.ext.filters.**UpdateFilter**(*name=None*, *data_filter=False*)

Bases: *telegram.ext.filters.BaseFilter*

Base class for all Update Filters. In contrast to *MessageFilter*, the object passed to *filter()* is an instance of *telegram.Update*, which allows to create filters like *telegram.ext.filters.UpdateType.EDITED_MESSAGE*.

Please see *telegram.ext.filters.BaseFilter* for details on how to create custom filters.

name

Name for this filter. Defaults to the type of filter.

Type
str

data_filter

Whether this filter is a data filter. A data filter should return a dict with lists. The dict will be merged with *telegram.ext.CallbackContext*'s internal dict in most cases (depends on the handler).

Type
bool

check_update(*update*)

Checks if the specified update is a message.

abstract filter(*update*)

This method must be overwritten.

Parameters
update (*telegram.Update*) – The update that is tested.

Returns
dict or bool.

class telegram.ext.filters.**UpdateType**

Bases: object

Subset for filtering the type of update.

Examples

Use these filters like: `filters.UpdateType.MESSAGE` or `filters.UpdateType.CHANNEL_POSTS` etc.

Caution: `filters.UpdateType` itself is *not* a filter, but just a convenience namespace.

CHANNEL_POST = filters.UpdateType.CHANNEL_POST

Updates with *telegram.Update.channel_post*.

CHANNEL_POSTS = filters.UpdateType.CHANNEL_POSTS

Updates with either *telegram.Update.channel_post* or *telegram.Update.edited_channel_post*.

EDITED = filters.UpdateType.EDITED

Updates with either *telegram.Update.edited_message* or *telegram.Update.edited_channel_post*.

New in version 20.0.

EDITED_CHANNEL_POST = filters.UpdateType.EDITED_CHANNEL_POST

Updates with *telegram.Update.edited_channel_post*.

EDITED_MESSAGE = filters.UpdateType.EDITED_MESSAGE

Updates with *telegram.Update.edited_message*.

MESSAGE = filters.UpdateType.MESSAGE

Updates with *telegram.Update.message*.

MESSAGES = filters.UpdateType.MESSAGES

Updates with either *telegram.Update.message* or *telegram.Update.edited_message*.

class telegram.ext.filters.**User**(*user_id=None*, *username=None*, *allow_empty=False*)

Bases: *telegram.ext.filters.MessageFilter*

Filters messages to allow only those which are from specified user ID(s) or username(s).

Examples

```
MessageHandler(filters.User(1234), callback_method)
```

Parameters

- ***user_id*** (int | Collection[int], optional) – Which user ID(s) to allow through.
- ***username*** (str | Collection[str], optional) – Which username(s) to allow through. Leading '@' s in usernames will be discarded.
- ***allow_empty*** (bool, optional) – Whether updates should be processed, if no user is specified in *user_ids* and *usernames*. Defaults to False.

Raises

RuntimeError – If user_id and username are both present.

allow_empty

Whether updates should be processed, if no user is specified in *user_ids* and *usernames*.

Type

bool

add_usernames(*username*)

Add one or more chats to the allowed usernames.

Parameters

username (str | Collection[str]) – Which username(s) to allow through. Leading '@' s in usernames will be discarded.

remove_usernames(*username*)

Remove one or more chats from allowed usernames.

Parameters

username (str | Collection[str]) – Which username(s) to disallow through. Leading '@' s in usernames will be discarded.

property usernames

Which username(s) to allow through.

Warning: *usernames* will give a *copy* of the saved usernames as frozenset. This is to ensure thread safety. To add/remove a user, you should use *add_usernames()*, and *remove_usernames()*. Only update the entire set by filter.usernames = new_set, if you are entirely sure that it is not causing race conditions, as this will complete replace the current set of allowed users.

Returns
frozenset(str)

property user_ids

Which user ID(s) to allow through.

> **Warning:** *user_ids* will give a *copy* of the saved user ids as frozenset. This is to ensure thread safety. To add/remove a user, you should use *add_user_ids()*, and *remove_user_ids()*. Only update the entire set by `filter.user_ids = new_set`, if you are entirely sure that it is not causing race conditions, as this will complete replace the current set of allowed users.

Returns
frozenset(int)

add_user_ids(*user_id*)

Add one or more users to the allowed user ids.

Parameters
user_id (int | Collection[int]) – Which user ID(s) to allow through.

remove_user_ids(*user_id*)

Remove one or more users from allowed user ids.

Parameters
user_id (int | Collection[int]) – Which user ID(s) to disallow through.

telegram.ext.filters.**VENUE = filters.VENUE**

Messages that contain *telegram.Message.venue*.

telegram.ext.filters.**VIA_BOT = filters.VIA_BOT**

This filter filters for message that were sent via *any* bot.

telegram.ext.filters.**VIDEO = filters.VIDEO**

Messages that contain *telegram.Message.video*.

telegram.ext.filters.**VIDEO_NOTE = filters.VIDEO_NOTE**

Messages that contain *telegram.Message.video_note*.

telegram.ext.filters.**VOICE = filters.VOICE**

Messages that contain *telegram.Message.voice*.

class telegram.ext.filters.**ViaBot**(*bot_id=None*, *username=None*, *allow_empty=False*)

Bases: *telegram.ext.filters.MessageFilter*

Filters messages to allow only those which are from specified via_bot ID(s) or username(s).

Examples

```
MessageHandler(filters.ViaBot(1234), callback_method)
```

Parameters

- ***bot_id*** (int | Collection[int], optional) – Which bot ID(s) to allow through.
- ***username*** (str | Collection[str], optional) – Which username(s) to allow through. Leading `'@'` s in usernames will be discarded.
- ***allow_empty*** (bool, optional) – Whether updates should be processed, if no user is specified in *bot_ids* and *usernames*. Defaults to False.

Raises
RuntimeError – If bot_id and username are both present.

allow_empty
Whether updates should be processed, if no bot is specified in *bot_ids* and *usernames*.

Type
bool

add_usernames(*username*)
Add one or more chats to the allowed usernames.

Parameters
username (str | Collection[str]) – Which username(s) to allow through. Leading '@' s in usernames will be discarded.

remove_usernames(*username*)
Remove one or more chats from allowed usernames.

Parameters
username (str | Collection[str]) – Which username(s) to disallow through. Leading '@' s in usernames will be discarded.

property usernames
Which username(s) to allow through.

> **Warning:** *usernames* will give a *copy* of the saved usernames as frozenset. This is to ensure thread safety. To add/remove a user, you should use *add_usernames()*, and *remove_usernames()*. Only update the entire set by filter.usernames = new_set, if you are entirely sure that it is not causing race conditions, as this will complete replace the current set of allowed users.

Returns
frozenset(str)

property bot_ids
Which bot ID(s) to allow through.

> **Warning:** *bot_ids* will give a *copy* of the saved bot ids as frozenset. This is to ensure thread safety. To add/remove a bot, you should use *add_bot_ids()*, and *remove_bot_ids()*. Only update the entire set by filter.bot_ids = new_set, if you are entirely sure that it is not causing race conditions, as this will complete replace the current set of allowed bots.

Returns
frozenset(int)

add_bot_ids(*bot_id*)
Add one or more bots to the allowed bot ids.

Parameters
bot_id (int | Collection[int]) – Which bot ID(s) to allow through.

remove_bot_ids(*bot_id*)
Remove one or more bots from allowed bot ids.

Parameters
bot_id (int | Collection[int], optional) – Which bot ID(s) to disallow through.

telegram.ext.InlineQueryHandler

class telegram.ext.**InlineQueryHandler**(*callback*, *pattern=None*, *block=True*, *chat_types=None*)

Bases: *telegram.ext.BaseHandler*

BaseHandler class to handle Telegram updates that contain a *telegram.Update.inline_query*. Optionally based on a regex. Read the documentation of the re module for more information.

Warning:

- When setting *block* to False, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.
- *telegram.InlineQuery.chat_type* will not be set for inline queries from secret chats and may not be set for inline queries coming from third-party clients. These updates won't be handled, if *chat_types* is passed.

See also:

Inlinebot Example

Parameters

- ***callback*** (coroutine function) – The callback function for this handler. Will be called when *check_update()* has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

  The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.

- ***pattern*** (str | re.Pattern, optional) – Regex pattern. If not None, re.match() is used on *telegram.InlineQuery.query* to determine if an update should be handled by this handler.
- ***block*** (bool, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*. Defaults to True.
- ***chat_types*** (List[str], optional) – List of allowed chat types. If passed, will only handle inline queries with the appropriate *telegram.InlineQuery.chat_type*.

  New in version 13.5.

callback

The callback function for this handler.

Type
coroutine function

pattern

Optional. Regex pattern to test *telegram.InlineQuery.query* against.

Type
str | re.Pattern

chat_types

Optional. List of allowed chat types.

New in version 13.5.

Type
List[str]

block

Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*.

Type
bool

check_update(*update*)

Determines whether an update should be passed to this handler's *callback*.

Parameters
update (*telegram.Update* | object) – Incoming update.

Returns
bool | re.match

collect_additional_context(*context*, *update*, *application*, *check_result*)

Add the result of `re.match(pattern, update.inline_query.query)` to *CallbackContext.matches* as list with one element.

telegram.ext.MessageHandler

class telegram.ext.**MessageHandler**(*filters*, *callback*, *block=True*)

Bases: *telegram.ext.BaseHandler*

BaseHandler class to handle Telegram messages. They might contain text, media or status updates.

Warning: When setting *block* to False, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

Parameters

- ***filters*** (*telegram.ext.filters.BaseFilter*) – A filter inheriting from *telegram.ext.filters.BaseFilter*. Standard filters can be found in *telegram.ext.filters*. Filters can be combined using bitwise operators (& for and, | for or, ~ for not). This defaults to all message updates being: *telegram.Update.message*, *telegram.Update.edited_message*, *telegram.Update.channel_post* and *telegram.Update.edited_channel_post*. If you don't want or need any of those pass ~filters.UpdateType.* in the filter argument.
- ***callback*** (coroutine function) – The callback function for this handler. Will be called when *check_update()* has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

  The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.
- ***block*** (bool, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*. Defaults to True.

filters

Only allow updates with these Filters. See *telegram.ext.filters* for a full list of all available filters.

Type
telegram.ext.filters.BaseFilter

callback

The callback function for this handler.

Type
coroutine function

block

Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*.

Type
bool

check_update(*update*)

Determines whether an update should be passed to this handler's *callback*.

Parameters
update (*telegram.Update* | object) – Incoming update.

Returns
bool

collect_additional_context(*context*, *update*, *application*, *check_result*)

Adds possible output of data filters to the *CallbackContext*.

telegram.ext.PollAnswerHandler

class telegram.ext.**PollAnswerHandler**(*callback*, *block=True*)

Bases: *telegram.ext.BaseHandler*

BaseHandler class to handle Telegram updates that contain a *poll answer*.

Warning: When setting *block* to False, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

See also:

Pollbot EXample

Parameters

- ***callback*** (coroutine function) – The callback function for this handler. Will be called when *check_update()* has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

  The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.

- ***block*** (bool, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*. Defaults to True.

callback

The callback function for this handler.

Type
coroutine function

block

Determines whether the callback will run in a blocking way..

Type
bool

check_update(*update*)

Determines whether an update should be passed to this handler's *callback*.

Parameters
update (*telegram.Update* | object) – Incoming update.

Returns
bool

telegram.ext.PollHandler

class telegram.ext.**PollHandler**(*callback*, *block=True*)

Bases: *telegram.ext.BaseHandler*

BaseHandler class to handle Telegram updates that contain a *poll*.

Warning: When setting *block* to False, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

See also:

Pollbot Example

Parameters

- ***callback*** (coroutine function) – The callback function for this handler. Will be called when *check_update()* has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

  The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.

- ***block*** (bool, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*. Defaults to True.

callback

The callback function for this handler.

Type
coroutine function

block

Determines whether the callback will run in a blocking way..

Type
bool

check_update(*update*)

Determines whether an update should be passed to this handler's *callback*.

Parameters
update (*telegram.Update* | object) – Incoming update.

Returns
bool

telegram.ext.PreCheckoutQueryHandler

class telegram.ext.**PreCheckoutQueryHandler**(*callback*, *block=True*)

Bases: *telegram.ext.BaseHandler*

BaseHandler class to handle Telegram *telegram.Update.pre_checkout_query*.

> **Warning:** When setting *block* to False, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

See also:

Paymentbot Example

Parameters

- ***callback*** (coroutine function) – The callback function for this handler. Will be called when *check_update()* has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

  The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.

- ***block*** (bool, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*. Defaults to True.

callback

The callback function for this handler.

Type
coroutine function

block

Determines whether the callback will run in a blocking way..

Type
bool

check_update(*update*)

Determines whether an update should be passed to this handler's *callback*.

Parameters
update (*telegram.Update* | object) – Incoming update.

Returns
bool

telegram.ext.PrefixHandler

class telegram.ext.**PrefixHandler**(*prefix*, *command*, *callback*, *filters=None*, *block=True*)

Bases: *telegram.ext.BaseHandler*

BaseHandler class to handle custom prefix commands.

This is an intermediate handler between *MessageHandler* and *CommandHandler*. It supports configurable commands with the same options as *CommandHandler*. It will respond to every combination of *prefix* and *command*. It will add a `list` to the *CallbackContext* named *CallbackContext.args*, containing a list of strings, which is the text following the command split on single or consecutive whitespace characters.

Examples

Single prefix and command:

```
PrefixHandler("!", "test", callback)  # will respond to '!test'.
```

Multiple prefixes, single command:

```
PrefixHandler(["!", "#"], "test", callback)  # will respond to '!test' and '#test'.
```

Multiple prefixes and commands:

```
PrefixHandler(
    ["!", "#"], ["test", "help"], callback
)  # will respond to '!test', '#test', '!help' and '#help'.
```

By default, the handler listens to messages as well as edited messages. To change this behavior use *~filters.UpdateType.EDITED_MESSAGE*

Note:

- *PrefixHandler* does *not* handle (edited) channel posts.

Warning: When setting *block* to `False`, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

Changed in version 20.0:

- *PrefixHandler* is no longer a subclass of *CommandHandler*.
- Removed the attributes `command` and `prefix`. Instead, the new *commands* contains all commands that this handler listens to as a `frozenset`, which includes the prefixes.
- Updating the prefixes and commands this handler listens to is no longer possible.

Parameters

- ***prefix*** (`str` | Collection[`str`]) – The prefix(es) that will precede *command*.
- ***command*** (`str` | Collection[`str`]) – The command or list of commands this handler should listen for. Case-insensitive.
- ***callback*** (coroutine function) – The callback function for this handler. Will be called when *check_update()* has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.

- ***filters*** (*telegram.ext.filters.BaseFilter*, optional) – A filter inheriting from *telegram.ext.filters.BaseFilter*. Standard filters can be found in *telegram.ext.filters*. Filters can be combined using bitwise operators (& for and, | for or, ~ for not)
- ***block*** (bool, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*. Defaults to True.

commands

The commands that this handler will listen for, i.e. the combinations of *prefix* and *command*.

Type
FrozenSet[str]

callback

The callback function for this handler.

Type
coroutine function

filters

Optional. Only allow updates with these Filters.

Type
telegram.ext.filters.BaseFilter

block

Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*.

Type
bool

check_update(*update*)

Determines whether an update should be passed to this handler's *callback*.

Parameters
update (*telegram.Update* | object) – Incoming update.

Returns
The list of args for the handler.

Return type
list

collect_additional_context(*context*, *update*, *application*, *check_result*)

Add text after the command to *CallbackContext.args* as list, split on single whitespaces and add output of data filters to *CallbackContext* as well.

telegram.ext.ShippingQueryHandler

class telegram.ext.**ShippingQueryHandler**(*callback*, *block=True*)

Bases: *telegram.ext.BaseHandler*

BaseHandler class to handle Telegram *telegram.Update.shipping_query*.

> **Warning:** When setting *block* to `False`, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

See also:

Paymentbot Example

Parameters

- ***callback*** (coroutine function) – The callback function for this handler. Will be called when *check_update()* has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

  The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.

- ***block*** (`bool`, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*. Defaults to `True`.

callback

The callback function for this handler.

Type
coroutine function

block

Determines whether the callback will run in a blocking way..

Type
`bool`

check_update(*update*)

Determines whether an update should be passed to this handler's *callback*.

Parameters
update (*telegram.Update* | `object`) – Incoming update.

Returns
`bool`

telegram.ext.StringCommandHandler

class telegram.ext.**StringCommandHandler**(*command*, *callback*, *block=True*)

Bases: *telegram.ext.BaseHandler*

BaseHandler class to handle string commands. Commands are string updates that start with `/`. The handler will add a `list` to the *CallbackContext* named *CallbackContext.args*. It will contain a list of strings, which is the text following the command split on single whitespace characters.

Note: This handler is not used to handle Telegram *telegram.Update*, but strings manually put in the queue. For example to send messages with the bot using command line or API.

Warning: When setting *block* to `False`, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

Parameters

- ***command*** (`str`) – The command this handler should listen for.
- ***callback*** (coroutine function) – The callback function for this handler. Will be called when *check_update()* has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

  The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.

- ***block*** (`bool`, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*. Defaults to `True`.

command

The command this handler should listen for.

Type
`str`

callback

The callback function for this handler.

Type
coroutine function

block

Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*.

Type
`bool`

check_update(*update*)

Determines whether an update should be passed to this handler's *callback*.

Parameters
update (`object`) – The incoming update.

Returns
List containing the text command split on whitespace.

Return type
List[`str`]

collect_additional_context(*context*, *update*, *application*, *check_result*)

Add text after the command to *CallbackContext.args* as list, split on single whitespaces.

telegram.ext.StringRegexHandler

class telegram.ext.**StringRegexHandler**(*pattern*, *callback*, *block=True*)

Bases: *telegram.ext.BaseHandler*

BaseHandler class to handle string updates based on a regex which checks the update content.

Read the documentation of the re module for more information. The re.match() function is used to determine if an update should be handled by this handler.

Note: This handler is not used to handle Telegram *telegram.Update*, but strings manually put in the queue. For example to send messages with the bot using command line or API.

Warning: When setting *block* to False, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

Parameters

- ***pattern*** (str | re.Pattern) – The regex pattern.
- ***callback*** (coroutine function) – The callback function for this handler. Will be called when *check_update()* has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

  The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.

- ***block*** (bool, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*. Defaults to True.

pattern

The regex pattern.

Type
str | re.Pattern

callback

The callback function for this handler.

Type
coroutine function

block

Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*.

Type
bool

check_update(*update*)

Determines whether an update should be passed to this handler's *callback*.

Parameters
update (object) – The incoming update.

Returns
None | re.match

collect_additional_context(*context*, *update*, *application*, *check_result*)

Add the result of `re.match(pattern, update)` to *CallbackContext.matches* as list with one element.

telegram.ext.TypeHandler

class telegram.ext.**TypeHandler**(*type*, *callback*, *strict=False*, *block=True*)

Bases: *telegram.ext.BaseHandler*

BaseHandler class to handle updates of custom types.

Warning: When setting *block* to `False`, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

Parameters

- ***type*** (`type`) – The `type` of updates this handler should process, as determined by `isinstance`
- ***callback*** (coroutine function) – The callback function for this handler. Will be called when *check_update()* has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

  The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.
- ***strict*** (`bool`, optional) – Use `type` instead of `isinstance`. Default is `False`.
- ***block*** (`bool`, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*. Defaults to `True`.

type

The `type` of updates this handler should process.

Type
`type`

callback

The callback function for this handler.

Type
coroutine function

strict

Use `type` instead of `isinstance`. Default is `False`.

Type
`bool`

block

Determines whether the return value of the callback should be awaited before processing the next handler in *telegram.ext.Application.process_update()*.

Type
`bool`

check_update(*update*)

Determines whether an update should be passed to this handler's *callback*.

Parameters
update (object) – Incoming update.

Returns
bool

10.2.12 Persistence

telegram.ext.BasePersistence

class telegram.ext.**BasePersistence**(*store_data=None*, *update_interval=60*)

Bases: typing.Generic, ABC

Interface class for adding persistence to your bot. Subclass this object for different implementations of a persistent bot.

Attention: The interface provided by this class is intended to be accessed exclusively by *Application*. Calling any of the methods below manually might interfere with the integration of persistence into *Application*.

All relevant methods must be overwritten. This includes:

- *get_bot_data()*
- *update_bot_data()*
- *refresh_bot_data()*
- *get_chat_data()*
- *update_chat_data()*
- *refresh_chat_data()*
- *drop_chat_data()*
- *get_user_data()*
- *update_user_data()*
- *refresh_user_data()*
- *drop_user_data()*
- *get_callback_data()*
- *update_callback_data()*
- *get_conversations()*
- *update_conversation()*
- *flush()*

If you don't actually need one of those methods, a simple pass is enough. For example, if you don't store bot_data, you don't need *get_bot_data()*, *update_bot_data()* or *refresh_bot_data()*.

Note: You should avoid saving *telegram.Bot* instances. This is because if you change e.g. the bots token, this won't propagate to the serialized instances and may lead to exceptions.

To prevent this, the implementation may use *bot* to replace bot instances with a placeholder before serialization and insert *bot* back when loading the data. Since *bot* will be set when the process starts, this will be the up-to-date bot instance.

If the persistence implementation does not take care of this, you should make sure not to store any bot instances in the data that will be persisted. E.g. in case of *telegram.TelegramObject*, one may call *set_bot()* to ensure that shortcuts like *telegram.Message.reply_text()* are available.

This class is a Generic class and accepts three type variables:

1. The type of the second argument of *update_user_data()*, which must coincide with the type of the second argument of *refresh_user_data()* and the values in the dictionary returned by *get_user_data()*.
2. The type of the second argument of *update_chat_data()*, which must coincide with the type of the second argument of *refresh_chat_data()* and the values in the dictionary returned by *get_chat_data()*.
3. The type of the argument of *update_bot_data()*, which must coincide with the type of the argument of *refresh_bot_data()* and the return value of *get_bot_data()*.

Changed in version 20.0:

- The parameters and attributes `store_*_data` were replaced by *store_data*.
- `insert/replace_bot` was dropped. Serialization of bot instances now needs to be handled by the specific implementation - see above note.

Parameters

- ***store_data*** (*PersistenceInput*, optional) – Specifies which kinds of data will be saved by this persistence instance. By default, all available kinds of data will be saved.
- ***update_interval*** (`int` | `float`, optional) – The *Application* will update the persistence in regular intervals. This parameter specifies the time (in seconds) to wait between two consecutive runs of updating the persistence. Defaults to `60` seconds.

  New in version 20.0.

store_data

Specifies which kinds of data will be saved by this persistence instance.

Type
PersistenceInput

bot

The bot associated with the persistence.

Type
telegram.Bot

abstract async drop_chat_data(*chat_id*)

Will be called by the *telegram.ext.Application*, when using *drop_chat_data()*.

New in version 20.0.

Parameters
chat_id (`int`) – The chat id to delete from the persistence.

abstract async drop_user_data(*user_id*)

Will be called by the *telegram.ext.Application*, when using *drop_user_data()*.

New in version 20.0.

Parameters
user_id (`int`) – The user id to delete from the persistence.

abstract async flush()

Will be called by *telegram.ext.Application.stop()*. Gives the persistence a chance to finish up saving or close a database connection gracefully.

Changed in version 20.0: Changed this method into an abstractmethod().

abstract async get_bot_data()

Will be called by *telegram.ext.Application* upon creation with a persistence object. It should return the bot_data if stored, or an empty dict. In the latter case, the dict should produce values corresponding to one of the following:

- dict
- The type from *telegram.ext.ContextTypes.bot_data* if *telegram.ext.ContextTypes* are used.

Returns
The restored bot data.

Return type
Dict[int, dict | *telegram.ext.ContextTypes.bot_data*]

abstract async get_callback_data()

Will be called by *telegram.ext.Application* upon creation with a persistence object. If callback data was stored, it should be returned.

New in version 13.6.

Changed in version 20.0: Changed this method into an abstractmethod().

Returns
Tuple[List[Tuple[str, float, Dict[str, object]]], Dict[str, str]] | None: The restored metadata or None, if no data was stored.

abstract async get_chat_data()

Will be called by *telegram.ext.Application* upon creation with a persistence object. It should return the chat_data if stored, or an empty dict. In the latter case, the dictionary should produce values corresponding to one of the following:

- dict
- The type from *telegram.ext.ContextTypes.chat_data* if *telegram.ext.ContextTypes* is used.

Changed in version 20.0: This method may now return a dict instead of a collections.defaultdict

Returns
The restored chat data.

Return type
Dict[int, dict | *telegram.ext.ContextTypes.chat_data*]

abstract async get_conversations(*name*)

Will be called by *telegram.ext.Application* when a *telegram.ext.ConversationHandler* is added if *telegram.ext.ConversationHandler.persistent* is True. It should return the conversations for the handler with *name* or an empty dict.

Parameters
name (str) – The handlers name.

Returns
The restored conversations for the handler.

Return type
dict

abstract async get_user_data()

Will be called by *telegram.ext.Application* upon creation with a persistence object. It should return the `user_data` if stored, or an empty `dict`. In the latter case, the dictionary should produce values corresponding to one of the following:

- `dict`
- The type from *telegram.ext.ContextTypes.user_data* if *telegram.ext.ContextTypes* is used.

Changed in version 20.0: This method may now return a `dict` instead of a `collections.defaultdict`

Returns
The restored user data.

Return type
Dict[`int`, `dict` | *telegram.ext.ContextTypes.user_data*]

abstract async refresh_bot_data(*bot_data*)

Will be called by the *telegram.ext.Application* before passing the *bot_data* to a callback. Can be used to update data stored in *bot_data* from an external source.

New in version 13.6.

Changed in version 20.0: Changed this method into an `abstractmethod()`.

Parameters
bot_data (`dict` | *telegram.ext.ContextTypes.bot_data*) – The `bot_data`.

abstract async refresh_chat_data(*chat_id*, *chat_data*)

Will be called by the *telegram.ext.Application* before passing the *chat_data* to a callback. Can be used to update data stored in *chat_data* from an external source.

New in version 13.6.

Changed in version 20.0: Changed this method into an `abstractmethod()`.

Parameters

- ***chat_id*** (`int`) – The chat ID this *chat_data* is associated with.
- ***chat_data*** (`dict` | *telegram.ext.ContextTypes.chat_data*) – The `chat_data` of a single chat.

abstract async refresh_user_data(*user_id*, *user_data*)

Will be called by the *telegram.ext.Application* before passing the *user_data* to a callback. Can be used to update data stored in *user_data* from an external source.

New in version 13.6.

Changed in version 20.0: Changed this method into an `abstractmethod()`.

Parameters

- ***user_id*** (`int`) – The user ID this *user_data* is associated with.
- ***user_data*** (`dict` | *telegram.ext.ContextTypes.user_data*) – The `user_data` of a single user.

set_bot(*bot*)

Set the Bot to be used by this persistence instance.

Parameters
bot (*telegram.Bot*) – The bot.

Raises
TypeError – If *PersistenceInput.callback_data* is `True` and the *bot* is not an instance of *telegram.ext.ExtBot*.

abstract async update_bot_data(*data*)

Will be called by the *telegram.ext.Application* after a handler has handled an update.

Parameters

data (dict | *telegram.ext.ContextTypes.bot_data*) – The *telegram.ext.Application.bot_data*.

abstract async update_callback_data(*data*)

Will be called by the *telegram.ext.Application* after a handler has handled an update.

New in version 13.6.

Changed in version 20.0: Changed this method into an abstractmethod().

Parameters

data (Tuple[List[Tuple[str, float, Dict[str, Any]]], Dict[str, str]] | None) – The relevant data to restore *telegram.ext.CallbackDataCache*.

abstract async update_chat_data(*chat_id*, *data*)

Will be called by the *telegram.ext.Application* after a handler has handled an update.

Parameters

- ***chat_id*** (int) – The chat the data might have been changed for.
- ***data*** (dict | *telegram.ext.ContextTypes.chat_data*) – The *telegram.ext.Application.chat_data* [chat_id].

abstract async update_conversation(*name*, *key*, *new_state*)

Will be called when a *telegram.ext.ConversationHandler* changes states. This allows the storage of the new state in the persistence.

Parameters

- ***name*** (str) – The handler's name.
- ***key*** (tuple) – The key the state is changed for.
- ***new_state*** (object) – The new state for the given key.

property update_interval

Time (in seconds) that the *Application* will wait between two consecutive runs of updating the persistence.

New in version 20.0.

Type

float

abstract async update_user_data(*user_id*, *data*)

Will be called by the *telegram.ext.Application* after a handler has handled an update.

Parameters

- ***user_id*** (int) – The user the data might have been changed for.
- ***data*** (dict | *telegram.ext.ContextTypes.user_data*) – The *telegram.ext.Application.user_data* [user_id].

telegram.ext.DictPersistence

class telegram.ext.**DictPersistence**(*store_data=None*, *user_data_json=''*, *chat_data_json=''*, *bot_data_json=''*, *conversations_json=''*, *callback_data_json=''*, *update_interval=60*)

Bases: *telegram.ext.BasePersistence*

Using Python's `dict` and `json` for making your bot persistent.

Attention: The interface provided by this class is intended to be accessed exclusively by *Application*. Calling any of the methods below manually might interfere with the integration of persistence into *Application*.

Note:

- Data managed by *DictPersistence* is in-memory only and will be lost when the bot shuts down. This is, because *DictPersistence* is mainly intended as starting point for custom persistence classes that need to JSON-serialize the stored data before writing them to file/database.
- This implementation of *BasePersistence* does not handle data that cannot be serialized by `json.dumps()`.

Changed in version 20.0: The parameters and attributes `store_*_data` were replaced by *store_data*.

Parameters

- ***store_data*** (*PersistenceInput*, optional) – Specifies which kinds of data will be saved by this persistence instance. By default, all available kinds of data will be saved.
- ***user_data_json*** (`str`, optional) – JSON string that will be used to reconstruct user_data on creating this persistence. Default is "".
- ***chat_data_json*** (`str`, optional) – JSON string that will be used to reconstruct chat_data on creating this persistence. Default is "".
- ***bot_data_json*** (`str`, optional) – JSON string that will be used to reconstruct bot_data on creating this persistence. Default is "".
- ***conversations_json*** (`str`, optional) – JSON string that will be used to reconstruct conversation on creating this persistence. Default is "".
- ***callback_data_json*** (`str`, optional) – JSON string that will be used to reconstruct callback_data on creating this persistence. Default is "".

  New in version 13.6.
- ***update_interval*** (`int` | `float`, optional) – The *Application* will update the persistence in regular intervals. This parameter specifies the time (in seconds) to wait between two consecutive runs of updating the persistence. Defaults to 60 seconds.

  New in version 20.0.

store_data

Specifies which kinds of data will be saved by this persistence instance.

Type
PersistenceInput

property bot_data

The bot_data as a dict.

Type
`dict`

property bot_data_json

The bot_data serialized as a JSON-string.

Type
str

property callback_data

The metadata on the stored callback data.

New in version 13.6.

Type
Tuple[List[Tuple[str, float, Dict[str, object]]], Dict[str, str]]

property callback_data_json

The metadata on the stored callback data as a JSON-string.

New in version 13.6.

Type
str

property chat_data

The chat_data as a dict.

Type
dict

property chat_data_json

The chat_data serialized as a JSON-string.

Type
str

property conversations

The conversations as a dict.

Type
dict

property conversations_json

The conversations serialized as a JSON-string.

Type
str

async drop_chat_data(*chat_id*)

Will delete the specified key from the *chat_data*.

New in version 20.0.

Parameters
chat_id (int) – The chat id to delete from the persistence.

async drop_user_data(*user_id*)

Will delete the specified key from the *user_data*.

New in version 20.0.

Parameters
user_id (int) – The user id to delete from the persistence.

async flush()

Does nothing.

New in version 20.0.

See also:

telegram.ext.BasePersistence.flush()

async get_bot_data()

Returns the bot_data created from the `bot_data_json` or an empty `dict`.

Returns
The restored bot data.

Return type
`dict`

async get_callback_data()

Returns the callback_data created from the `callback_data_json` or `None`.

New in version 13.6.

Returns
The restored metadata or `None`, if no data was stored.

Return type
Tuple[List[Tuple[`str`, `float`, Dict[`str`, `object`]]], Dict[`str`, `str`]]

async get_chat_data()

Returns the chat_data created from the `chat_data_json` or an empty `dict`.

Returns
The restored chat data.

Return type
`dict`

async get_conversations(*name*)

Returns the conversations created from the `conversations_json` or an empty `dict`.

Returns
The restored conversations data.

Return type
`dict`

async get_user_data()

Returns the user_data created from the `user_data_json` or an empty `dict`.

Returns
The restored user data.

Return type
`dict`

async refresh_bot_data(*bot_data*)

Does nothing.

New in version 13.6.

See also:

telegram.ext.BasePersistence.refresh_bot_data()

async refresh_chat_data(*chat_id*, *chat_data*)

Does nothing.

New in version 13.6.

See also:

telegram.ext.BasePersistence.refresh_chat_data()

async refresh_user_data(*user_id*, *user_data*)

Does nothing.

New in version 13.6.

See also:

telegram.ext.BasePersistence.refresh_user_data()

async update_bot_data(*data*)

Will update the bot_data (if changed).

Parameters

data (dict) – The *telegram.ext.Application.bot_data*.

async update_callback_data(*data*)

Will update the callback_data (if changed).

New in version 13.6.

Parameters

data (Tuple[List[Tuple[str, float, Dict[str, object]]], Dict[str, str]]) – The relevant data to restore *telegram.ext.CallbackDataCache*.

async update_chat_data(*chat_id*, *data*)

Will update the chat_data (if changed).

Parameters

- ***chat_id*** (int) – The chat the data might have been changed for.
- ***data*** (dict) – The *telegram.ext.Application.chat_data* [chat_id].

async update_conversation(*name*, *key*, *new_state*)

Will update the conversations for the given handler.

Parameters

- ***name*** (str) – The handler's name.
- ***key*** (tuple) – The key the state is changed for.
- ***new_state*** (tuple | object) – The new state for the given key.

async update_user_data(*user_id*, *data*)

Will update the user_data (if changed).

Parameters

- ***user_id*** (int) – The user the data might have been changed for.
- ***data*** (dict) – The *telegram.ext.Application.user_data* [user_id].

property user_data

The user_data as a dict.

Type

dict

property user_data_json

The user_data serialized as a JSON-string.

Type

str

telegram.ext.PersistenceInput

class telegram.ext.**PersistenceInput**(*bot_data=True*, *chat_data=True*, *user_data=True*, *callback_data=True*)

Bases: NamedTuple

Convenience wrapper to group boolean input for the *store_data* parameter for *BasePersistence*.

Parameters

- ***bot_data*** (bool, optional) – Whether the setting should be applied for bot_data. Defaults to True.
- ***chat_data*** (bool, optional) – Whether the setting should be applied for chat_data. Defaults to True.
- ***user_data*** (bool, optional) – Whether the setting should be applied for user_data. Defaults to True.
- ***callback_data*** (bool, optional) – Whether the setting should be applied for callback_data. Defaults to True.

bot_data

Whether the setting should be applied for bot_data.

Type
bool

chat_data

Whether the setting should be applied for chat_data.

Type
bool

user_data

Whether the setting should be applied for user_data.

Type
bool

callback_data

Whether the setting should be applied for callback_data.

Type
bool

telegram.ext.PicklePersistence

class telegram.ext.**PicklePersistence**(*filepath*, *store_data=None*, *single_file=True*, *on_flush=False*, *update_interval=60*, *context_types=None*)

Bases: *telegram.ext.BasePersistence*

Using python's builtin pickle for making your bot persistent.

Attention: The interface provided by this class is intended to be accessed exclusively by *Application*. Calling any of the methods below manually might interfere with the integration of persistence into *Application*.

Note: This implementation of *BasePersistence* uses the functionality of the pickle module to support serialization of bot instances. Specifically any reference to *bot* will be replaced by a placeholder before

pickling and *bot* will be inserted back when loading the data.

Changed in version 20.0:

- The parameters and attributes `store_*_data` were replaced by *store_data*.
- The parameter and attribute `filename` were replaced by *filepath*.
- *filepath* now also accepts `pathlib.Path` as argument.

Parameters

- ***filepath*** (`str | pathlib.Path`) – The filepath for storing the pickle files. When *single_file* is `False` this will be used as a prefix.
- ***store_data*** (*PersistenceInput*, optional) – Specifies which kinds of data will be saved by this persistence instance. By default, all available kinds of data will be saved.
- ***single_file*** (`bool`, optional) – When `False` will store 5 separate files of *filename_user_data*, *filename_bot_data*, *filename_chat_data*, *filename_callback_data* and *filename_conversations*. Default is `True`.
- ***on_flush*** (`bool`, optional) – When `True` will only save to file when *flush()* is called and keep data in memory until that happens. When `False` will store data on any transaction *and* on call to *flush()*. Default is `False`.
- ***context_types*** (*telegram.ext.ContextTypes*, optional) – Pass an instance of *telegram.ext.ContextTypes* to customize the types used in the `context` interface. If not passed, the defaults documented in *telegram.ext.ContextTypes* will be used.

  New in version 13.6.
- ***update_interval*** (`int | float`, optional) – The *Application* will update the persistence in regular intervals. This parameter specifies the time (in seconds) to wait between two consecutive runs of updating the persistence. Defaults to 60 seconds.

  New in version 20.0.

filepath

The filepath for storing the pickle files. When *single_file* is `False` this will be used as a prefix.

Type
`str | pathlib.Path`

store_data

Specifies which kinds of data will be saved by this persistence instance.

Type
PersistenceInput

single_file

Optional. When `False` will store 5 separate files of *filename_user_data*, *filename_bot_data*, *filename_chat_data*, *filename_callback_data* and *filename_conversations*. Default is `True`.

Type
`bool`

on_flush

When `True` will only save to file when *flush()* is called and keep data in memory until that happens. When `False` will store data on any transaction *and* on call to *flush()*. Default is `False`.

Type
`bool`, optional

context_types

Container for the types used in the `context` interface.

New in version 13.6.

Type
telegram.ext.ContextTypes

async drop_chat_data(*chat_id*)

Will delete the specified key from the `chat_data` and depending on *on_flush* save the pickle file.

New in version 20.0.

Parameters
chat_id (`int`) – The chat id to delete from the persistence.

async drop_user_data(*user_id*)

Will delete the specified key from the `user_data` and depending on *on_flush* save the pickle file.

New in version 20.0.

Parameters
user_id (`int`) – The user id to delete from the persistence.

async flush()

Will save all data in memory to pickle file(s).

async get_bot_data()

Returns the bot_data from the pickle file if it exists or an empty object of type `dict` | *telegram.ext.ContextTypes.bot_data*.

Returns
The restored bot data.

Return type
`dict` | *telegram.ext.ContextTypes.bot_data*

async get_callback_data()

Returns the callback data from the pickle file if it exists or `None`.

New in version 13.6.

Returns
Tuple[List[Tuple[`str`, `float`, Dict[`str`, `object`]]], Dict[`str`, `str`]] | `None`: The restored metadata or `None`, if no data was stored.

async get_chat_data()

Returns the chat_data from the pickle file if it exists or an empty `dict`.

Returns
The restored chat data.

Return type
Dict[`int`, `dict`]

async get_conversations(*name*)

Returns the conversations from the pickle file if it exists or an empty dict.

Parameters
name (`str`) – The handlers name.

Returns
The restored conversations for the handler.

Return type
`dict`

async get_user_data()

Returns the user_data from the pickle file if it exists or an empty dict.

Returns
The restored user data.

Return type
Dict[int, dict]

async refresh_bot_data(*bot_data*)

Does nothing.

New in version 13.6.

See also:

telegram.ext.BasePersistence.refresh_bot_data()

async refresh_chat_data(*chat_id*, *chat_data*)

Does nothing.

New in version 13.6.

See also:

telegram.ext.BasePersistence.refresh_chat_data()

async refresh_user_data(*user_id*, *user_data*)

Does nothing.

New in version 13.6.

See also:

telegram.ext.BasePersistence.refresh_user_data()

async update_bot_data(*data*)

Will update the bot_data and depending on *on_flush* save the pickle file.

Parameters
data (dict | *telegram.ext.ContextTypes.bot_data*) – The *telegram.ext.Application.bot_data*.

async update_callback_data(*data*)

Will update the callback_data (if changed) and depending on *on_flush* save the pickle file.

New in version 13.6.

Parameters
data (Tuple[List[Tuple[str, float, Dict[str, object]]], Dict[str, str]]) – The relevant data to restore *telegram.ext.CallbackDataCache*.

async update_chat_data(*chat_id*, *data*)

Will update the chat_data and depending on *on_flush* save the pickle file.

Parameters

- ***chat_id*** (int) – The chat the data might have been changed for.
- ***data*** (dict) – The *telegram.ext.Application.chat_data* [chat_id].

async update_conversation(*name*, *key*, *new_state*)

Will update the conversations for the given handler and depending on *on_flush* save the pickle file.

Parameters

- ***name*** (str) – The handler's name.
- ***key*** (tuple) – The key the state is changed for.

- ***new_state*** (object) – The new state for the given key.

async update_user_data(*user_id*, *data*)

Will update the user_data and depending on *on_flush* save the pickle file.

Parameters

- ***user_id*** (int) – The user the data might have been changed for.
- ***data*** (dict) – The *telegram.ext.Application.user_data* [user_id].

10.2.13 Arbitrary Callback Data

telegram.ext.CallbackDataCache

class telegram.ext.**CallbackDataCache**(*bot*, *maxsize=1024*, *persistent_data=None*)

Bases: object

A custom cache for storing the callback data of a *telegram.ext.ExtBot*. Internally, it keeps two mappings with fixed maximum size:

- One for mapping the data received in callback queries to the cached objects
- One for mapping the IDs of received callback queries to the cached objects

The second mapping allows to manually drop data that has been cached for keyboards of messages sent via inline mode. If necessary, will drop the least recently used items.

See also:

telegram.ext.ExtBot.callback_data_cache, Arbitrary callback_data, Arbitrary Callback Data Example <examples.arbitrarycallbackdatabot.html>

New in version 13.6.

Parameters

- ***bot*** (*telegram.ext.ExtBot*) – The bot this cache is for.
- ***maxsize*** (int, optional) – Maximum number of items in each of the internal mappings. Defaults to 1024.
- ***persistent_data*** (Tuple[List[Tuple[str, float, Dict[str, object]]], Dict[str, str]], optional) – Data to initialize the cache with, as returned by *telegram.ext.BasePersistence.get_callback_data()*.

bot

The bot this cache is for.

Type
telegram.ext.ExtBot

maxsize

maximum size of the cache.

Type
int

clear_callback_data(*time_cutoff=None*)

Clears the stored callback data.

Parameters
time_cutoff (float | datetime.datetime, optional) – Pass a UNIX timestamp or a datetime.datetime to clear only entries which are older. For timezone naive datetime.datetime objects, the default timezone of the bot will be used, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

clear_callback_queries()

Clears the stored callback query IDs.

drop_data(*callback_query*)

Deletes the data for the specified callback query.

Note: Will *not* raise exceptions in case the callback data is not found in the cache. *Will* raise KeyError in case the callback query can not be found in the cache.

Parameters

callback_query (*telegram.CallbackQuery*) – The callback query.

Raises

KeyError – If the callback query can not be found in the cache

static extract_uuids(*callback_data*)

Extracts the keyboard uuid and the button uuid from the given *callback_data*.

Parameters

callback_data (str) – The *callback_data* as present in the button.

Returns

Tuple of keyboard and button uuid

Return type

(str, str)

property persistence_data

Tuple[List[Tuple[str, float, Dict[str, object]]], Dict[str, str]]: The data that needs to be persisted to allow caching callback data across bot reboots.

process_callback_query(*callback_query*)

Replaces the data in the callback query and the attached messages keyboard with the cached objects, if necessary. If the data could not be found, *telegram.ext.InvalidCallbackData* will be inserted. If *telegram.CallbackQuery.data* or *telegram.CallbackQuery.message* is present, this also saves the callback queries ID in order to be able to resolve it to the stored data.

Note: Also considers inserts data into the buttons of *telegram.Message.reply_to_message* and *telegram.Message.pinned_message* if necessary.

Warning: *In place*, i.e. the passed *telegram.CallbackQuery* will be changed!

Parameters

callback_query (*telegram.CallbackQuery*) – The callback query.

process_keyboard(*reply_markup*)

Registers the reply markup to the cache. If any of the buttons have *callback_data*, stores that data and builds a new keyboard with the correspondingly replaced buttons. Otherwise, does nothing and returns the original reply markup.

Parameters

reply_markup (*telegram.InlineKeyboardMarkup*) – The keyboard.

Returns

The keyboard to be passed to Telegram.

Return type
telegram.InlineKeyboardMarkup

process_message(*message*)

Replaces the data in the inline keyboard attached to the message with the cached objects, if necessary. If the data could not be found, *telegram.ext.InvalidCallbackData* will be inserted.

Note: Checks *telegram.Message.via_bot* and *telegram.Message.from_user* to check if the reply markup (if any) was actually sent by this cache's bot. If it was not, the message will be returned unchanged.

Note that this will fail for channel posts, as *telegram.Message.from_user* is None for those! In the corresponding reply markups the callback data will be replaced by *telegram.ext.InvalidCallbackData*.

Warning:

- Does *not* consider *telegram.Message.reply_to_message* and *telegram.Message.pinned_message*. Pass them to this method separately.
- *In place*, i.e. the passed *telegram.Message* will be changed!

Parameters
message (*telegram.Message*) – The message.

telegram.ext.InvalidCallbackData

class telegram.ext.**InvalidCallbackData**(*callback_data=None*)

Bases: *telegram.error.TelegramError*

Raised when the received callback data has been tempered with or deleted from cache.

New in version 13.6.

Parameters
callback_data (int, optional) – The button data of which the callback data could not be found.

callback_data

Optional. The button data of which the callback data could not be found.

Type
int

10.2.14 Rate Limiting

telegram.ext.BaseRateLimiter

class telegram.ext.**BaseRateLimiter**

Bases: ABC, typing.Generic

Abstract interface class that allows to rate limit the requests that python-telegram-bot sends to the Telegram Bot API. An implementation of this class must implement all abstract methods and properties.

This class is a Generic class and accepts one type variable that specifies the type of the argument *rate_limit_args* of *process_request()* and the methods of *ExtBot*.

Hint: Requests to *get_updates()* are never rate limited.

New in version 20.0.

abstract async initialize()

Initialize resources used by this class. Must be implemented by a subclass.

abstract async process_request(*callback*, *args*, *kwargs*, *endpoint*, *data*, *rate_limit_args*)

Process a request. Must be implemented by a subclass.

This method must call *callback* and return the result of the call. *When* the callback is called is up to the implementation.

Important: This method must only return once the result of *callback* is known!

If a *RetryAfter* error is raised, this method may try to make a new request by calling the callback again.

Warning: This method *should not* handle any other exception raised by *callback*!

There are basically two different approaches how a rate limiter can be implemented:

1. React only if necessary. In this case, the *callback* is called without any precautions. If a *RetryAfter* error is raised, processing requests is halted for the *retry_after* and finally the *callback* is called again. This approach is often amendable for bots that don't have a large user base and/or don't send more messages than they get updates.
2. Throttle all outgoing requests. In this case the implementation makes sure that the requests are spread out over a longer time interval in order to stay below the rate limits. This approach is often amendable for bots that have a large user base and/or send more messages than they get updates.

An implementation can use the information provided by *data*, *endpoint* and *rate_limit_args* to handle each request differently.

Examples

- It is usually desirable to call *telegram.Bot.answer_inline_query()* as quickly as possible, while delaying *telegram.Bot.send_message()* is acceptable.
- There are different rate limits for group chats and private chats.
- When sending broadcast messages to a large number of users, these requests can typically be delayed for a longer time than messages that are direct replies to a user input.

Parameters

- ***callback*** (Callable[..., coroutine]) – The coroutine function that must be called to make the request.
- ***args*** (Tuple[object]) – The positional arguments for the *callback* function.
- ***kwargs*** (Dict[str, object]) – The keyword arguments for the *callback* function.
- ***endpoint*** (str) – The endpoint that the request is made for, e.g. "sendMessage".
- ***data*** (Dict[str, object]) – The parameters that were passed to the method of *ExtBot*. Any api_kwargs are included in this and any *defaults* are already applied.

Example

When calling:

```
await ext_bot.send_message(
    chat_id=1,
    text="Hello world!",
    api_kwargs={"custom": "arg"}
)
```

then *data* will be:

```
{"chat_id": 1, "text": "Hello world!", "custom": "arg"}
```

- ***rate_limit_args*** (None | object) – Custom arguments passed to the methods of *ExtBot*. Can e.g. be used to specify the priority of the request.

Returns
The result of the callback function.

Return type
bool | Dict[str, object] | None

abstract async shutdown()
Stop & clear resources used by this class. Must be implemented by a subclass.

telegram.ext.AIORateLimiter

class telegram.ext.**AIORateLimiter**(*overall_max_rate=30*, *overall_time_period=1*, *group_max_rate=20*, *group_time_period=60*, *max_retries=0*)

Bases: *telegram.ext.BaseRateLimiter*

Implementation of *BaseRateLimiter* using the library aiolimiter.

Important: If you want to use this class, you must install PTB with the optional requirement `rate-limiter`, i.e.

```
pip install python-telegram-bot[rate-limiter]
```

The rate limiting is applied by combining two levels of throttling and *process_request()* roughly boils down to:

```
async with group_limiter(group_id):
    async with overall_limiter:
        await callback(*args, **kwargs)
```

Here, `group_id` is determined by checking if there is a `chat_id` parameter in the *data*. The `overall_limiter` is applied only if a `chat_id` argument is present at all.

Attention:

- Some bot methods accept a `chat_id` parameter in form of a `@username` for supergroups and channels. As we can't know which `@username` corresponds to which integer `chat_id`, these will be treated as different groups, which may lead to exceeding the rate limit.

- As channels can't be differentiated from supergroups by the `@username` or integer `chat_id`, this also applies the group related rate limits to channels.
- A *RetryAfter* exception will halt *all* requests for *retry_after* + 0.1 seconds. This may be stricter than necessary in some cases, e.g. the bot may hit a rate limit in one group but might still be allowed to send messages in another group.

Note: This class is to be understood as minimal effort reference implementation. If you would like to handle rate limiting in a more sophisticated, fine-tuned way, we welcome you to implement your own subclass of *BaseRateLimiter*. Feel free to check out the source code of this class for inspiration.

New in version 20.0.

Parameters

- ***overall_max_rate*** (float) – The maximum number of requests allowed for the entire bot per *overall_time_period*. When set to 0, no rate limiting will be applied. Defaults to `30`.
- ***overall_time_period*** (float) – The time period (in seconds) during which the *overall_max_rate* is enforced. When set to 0, no rate limiting will be applied. Defaults to 1.
- ***group_max_rate*** (float) – The maximum number of requests allowed for requests related to groups and channels per *group_time_period*. When set to 0, no rate limiting will be applied. Defaults to 20.
- ***group_time_period*** (float) – The time period (in seconds) during which the *group_time_period* is enforced. When set to 0, no rate limiting will be applied. Defaults to 60.
- ***max_retries*** (int) – The maximum number of retries to be made in case of a *RetryAfter* exception. If set to 0, no retries will be made. Defaults to `0`.

async initialize()

Does nothing.

async process_request(*callback*, *args*, *kwargs*, *endpoint*, *data*, *rate_limit_args*)

Processes a request by applying rate limiting.

See *telegram.ext.BaseRateLimiter.process_request()* for detailed information on the arguments.

Parameters

rate_limit_args (None | int) – If set, specifies the maximum number of retries to be made in case of a *RetryAfter* exception. Defaults to *AIORateLimiter.max_retries*.

async shutdown()

Does nothing.

10.3 Auxiliary modules

10.3.1 telegram.constants Module

This module contains several constants that are relevant for working with the Bot API.

Unless noted otherwise, all constants in this module were extracted from the Telegram Bots FAQ and Telegram Bots API.

Most of the following constants are related to specific classes or topics and are grouped into enums. If they are related to a specific class, then they are also available as attributes of those classes.

Changed in version 20.0:

- Most of the constants in this module are grouped into enums.

telegram.constants.**BOT_API_VERSION = '6.2'**

Telegram Bot API version supported by this version of *python-telegram-bot*. Also available as *telegram.__bot_api_version__*.

New in version 13.4.

Type
str

telegram.constants.**BOT_API_VERSION_INFO = BotAPIVersion(major=6, minor=2)**

The components can also be accessed by name, so `BOT_API_VERSION_INFO[0]` is equivalent to `BOT_API_VERSION_INFO.major` and so on. Also available as *telegram.__bot_api_version_info__*.

New in version 20.0.

class telegram.constants.**BotCommandScopeType**(*value*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.BotCommandScope*. The enum members of this enumeration are instances of str and can be treated as such.

New in version 20.0.

ALL_CHAT_ADMINISTRATORS = 'all_chat_administrators'

The type of *telegram.BotCommandScopeAllChatAdministrators*.

Type
str

ALL_GROUP_CHATS = 'all_group_chats'

The type of *telegram.BotCommandScopeAllGroupChats*.

Type
str

ALL_PRIVATE_CHATS = 'all_private_chats'

The type of *telegram.BotCommandScopeAllPrivateChats*.

Type
str

CHAT = 'chat'

The type of *telegram.BotCommandScopeChat*.

Type
str

CHAT_ADMINISTRATORS = 'chat_administrators'

The type of *telegram.BotCommandScopeChatAdministrators*.

Type
str

CHAT_MEMBER = 'chat_member'

The type of *telegram.BotCommandScopeChatMember*.

Type
str

DEFAULT = 'default'

The type of *telegram.BotCommandScopeDefault*.

Type
str

class telegram.constants.**CallbackQueryLimit**(*value*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.CallbackQuery*/ *telegram.Bot.answer_callback_query()*. The enum members of this enumeration are instances of int and can be treated as such.

New in version 20.0.

ANSWER_CALLBACK_QUERY_TEXT_LENGTH = 200

Maximum number of characters for the `text` parameter of *telegram.Bot.answer_callback_query()*.

Type
int

class telegram.constants.**ChatAction**(*value*)

Bases: str, enum.Enum

This enum contains the available chat actions for *telegram.Bot.send_chat_action()*. The enum members of this enumeration are instances of str and can be treated as such.

New in version 20.0.

CHOOSE_STICKER = 'choose_sticker'

Chat action indicating that the bot is selecting a sticker.

Type
str

FIND_LOCATION = 'find_location'

Chat action indicating that the bot is selecting a location.

Type
str

RECORD_VIDEO = 'record_video'

Chat action indicating that the bot is recording a video.

Type
str

RECORD_VIDEO_NOTE = 'record_video_note'

Chat action indicating that the bot is recording a video note.

Type
str

RECORD_VOICE = 'record_voice'

Chat action indicating that the bot is recording a voice message.

Type
str

TYPING = 'typing'

A chat indicating the bot is typing.

Type
str

UPLOAD_DOCUMENT = 'upload_document'

Chat action indicating that the bot is uploading a document.

Type
str

UPLOAD_PHOTO = 'upload_photo'

Chat action indicating that the bot is uploading a photo.

Type
str

UPLOAD_VIDEO = 'upload_video'

Chat action indicating that the bot is uploading a video.

Type
str

UPLOAD_VIDEO_NOTE = 'upload_video_note'

Chat action indicating that the bot is uploading a video note.

Type
str

UPLOAD_VOICE = 'upload_voice'

Chat action indicating that the bot is uploading a voice message.

Type
str

class telegram.constants.**ChatID**(*value*)

Bases: enum.IntEnum

This enum contains some special chat IDs. The enum members of this enumeration are instances of int and can be treated as such.

New in version 20.0.

ANONYMOUS_ADMIN = 1087968824

User ID in groups for messages sent by anonymous admins.

Note: *telegram.Message.from_user* will contain this ID for backwards compatibility only. It's recommended to use *telegram.Message.sender_chat* instead.

Type
int

FAKE_CHANNEL = 136817688

User ID in groups when message is sent on behalf of a channel.

Note:

- *telegram.Message.from_user* will contain this ID for backwards compatibility only. It's recommended to use *telegram.Message.sender_chat* instead.
- This value is undocumented and might be changed by Telegram.

Type
int

SERVICE_CHAT = 777000

Telegram service chat, that also acts as sender of channel posts forwarded to discussion groups.

Note: *telegram.Message.from_user* will contain this ID for backwards compatibility only. It's recommended to use *telegram.Message.sender_chat* instead.

Type
int

class telegram.constants.**ChatInviteLinkLimit**(*value*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.ChatInviteLink*/ *telegram.Bot.create_chat_invite_link()*/*telegram.Bot.edit_chat_invite_link()*. The enum members of this enumeration are instances of int and can be treated as such.

New in version 20.0.

MEMBER_LIMIT = 99999

Maximum value allowed for the member_limit parameter of *telegram.Bot.create_chat_invite_link()* and *telegram.Bot.edit_chat_invite_link()*.

Type
int

NAME_LENGTH = 32

Maximum number of characters allowed for the name parameter of *telegram.Bot.create_chat_invite_link()* and *telegram.Bot.edit_chat_invite_link()*.

Type
int

class telegram.constants.**ChatMemberStatus**(*value*)

Bases: str, enum.Enum

This enum contains the available states for *telegram.ChatMember*. The enum members of this enumeration are instances of str and can be treated as such.

New in version 20.0.

ADMINISTRATOR = 'administrator'

A *telegram.ChatMember* who is administrator of the chat.

Type
str

BANNED = 'kicked'

A *telegram.ChatMember* who was banned in the chat.

Type
str

LEFT = 'left'

A *telegram.ChatMember* who has left the chat.

Type
str

MEMBER = 'member'

A *telegram.ChatMember* who is a member of the chat.

Type
str

OWNER = 'creator'

A *telegram.ChatMember* who is the owner of the chat.

Type
str

RESTRICTED = 'restricted'

A *telegram.ChatMember* who was restricted in this chat.

Type
str

class telegram.constants.**ChatType**(*value*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.Chat*. The enum members of this enumeration are instances of str and can be treated as such.

New in version 20.0.

CHANNEL = 'channel'

A *telegram.Chat* that is a channel.

Type
str

GROUP = 'group'

A *telegram.Chat* that is a group.

Type
str

PRIVATE = 'private'

A *telegram.Chat* that is private.

Type
str

SENDER = 'sender'

A *telegram.Chat* that represents the chat of a *telegram.User* sending an *telegram.InlineQuery*.

Type
str

SUPERGROUP = 'supergroup'

A *telegram.Chat* that is a supergroup.

Type
str

class telegram.constants.**CustomEmojiStickerLimit**(*value*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Bot.get_custom_emoji_stickers()*. The enum members of this enumeration are instances of int and can be treated as such.

New in version 20.0.

CUSTOM_EMOJI_IDENTIFIER_LIMIT = 200

Maximum amount of custom emoji identifiers which can be specified for the *custom_emoji_ids* parameter of *telegram.Bot.get_custom_emoji_stickers()*.

Type
int

class telegram.constants.**DiceEmoji**(*value*)

Bases: str, enum.Enum

This enum contains the available emoji for *telegram.Dice*/ *telegram.Bot.send_dice()*. The enum members of this enumeration are instances of str and can be treated as such.

New in version 20.0.

BASKETBALL = ''

A *telegram.Dice* with the emoji .

Type
str

BOWLING = ''

A *telegram.Dice* with the emoji .

Type
str

DARTS = ''

A *telegram.Dice* with the emoji .

Type
str

DICE = ''

A *telegram.Dice* with the emoji .

Type
str

FOOTBALL = ''

A *telegram.Dice* with the emoji .

Type
str

SLOT_MACHINE = ''

A *telegram.Dice* with the emoji .

Type
str

class telegram.constants.**FileSizeLimit**(*value*)

Bases: enum.IntEnum

This enum contains limitations regarding the upload and download of files. The enum members of this enumeration are instances of int and can be treated as such.

New in version 20.0.

FILESIZE_DOWNLOAD = 20000000

Bots can download files of up to 20MB in size.

Type
int

FILESIZE_UPLOAD = 50000000

Bots can upload non-photo files of up to 50MB in size.

Type
int

PHOTOSIZE_UPLOAD = 10000000

Bots can upload photo files of up to 10MB in size.

Type
int

class telegram.constants.**FloodLimit**(*value*)

Bases: enum.IntEnum

This enum contains limitations regarding flood limits. The enum members of this enumeration are instances of int and can be treated as such.

New in version 20.0.

MESSAGES_PER_MINUTE_PER_GROUP = 20

The number of messages that can roughly be sent to a particular group within one minute.

Type
int

MESSAGES_PER_SECOND = 30

The number of messages that can roughly be sent in an interval of 30 seconds across all chats.

Type
int

MESSAGES_PER_SECOND_PER_CHAT = 1

The number of messages that can be sent per second in a particular chat. Telegram may allow short bursts that go over this limit, but eventually you'll begin receiving 429 errors.

Type
int

class telegram.constants.**InlineKeyboardMarkupLimit**(*value*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.InlineKeyboardMarkup*/ *telegram.Bot.send_message()* & friends. The enum members of this enumeration are instances of int and can be treated as such.

New in version 20.0.

BUTTONS_PER_ROW = 8

Maximum number of buttons that can be attached to a message per row.

Note: This value is undocumented and might be changed by Telegram.

Type
int

TOTAL_BUTTON_NUMBER = 100

Maximum number of buttons that can be attached to a message.

Note: This value is undocumented and might be changed by Telegram.

Type
int

class telegram.constants.**InlineQueryLimit**(*value*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.InlineQuery*/ *telegram.Bot.answer_inline_query()*. The enum members of this enumeration are instances of int and can be treated as such.

New in version 20.0.

RESULTS = 50

Maximum number of results that can be passed to *telegram.Bot.answer_inline_query()*.

Type
int

SWITCH_PM_TEXT_LENGTH = 64

Maximum number of characters for the switch_pm_text parameter of *telegram.Bot.answer_inline_query()*.

Type
int

class telegram.constants.**InlineQueryResultType**(*value*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.InlineQueryResult*. The enum members of this enumeration are instances of str and can be treated as such.

New in version 20.0.

ARTICLE = 'article'

Type of *telegram.InlineQueryResultArticle*.

Type
str

AUDIO = 'audio'

Type of *telegram.InlineQueryResultAudio* and *telegram.InlineQueryResultCachedAudio*.

Type
str

CONTACT = 'contact'

Type of *telegram.InlineQueryResultContact*.

Type
str

DOCUMENT = 'document'

Type of *telegram.InlineQueryResultDocument* and *telegram.InlineQueryResultCachedDocument*.

Type
str

GAME = 'game'

Type of *telegram.InlineQueryResultGame*.

Type
str

GIF = 'gif'

Type of *telegram.InlineQueryResultGif* and *telegram.InlineQueryResultCachedGif*.

Type
str

LOCATION = 'location'

Type of *telegram.InlineQueryResultLocation*.

Type
str

MPEG4GIF = 'mpeg4_gif'

Type of *telegram.InlineQueryResultMpeg4Gif* and *telegram.InlineQueryResultCachedMpeg4Gif*.

Type
str

PHOTO = 'photo'

Type of *telegram.InlineQueryResultPhoto* and *telegram.InlineQueryResultCachedPhoto*.

Type
str

STICKER = 'sticker'

Type of and *telegram.InlineQueryResultCachedSticker*.

Type
str

VENUE = 'venue'

Type of *telegram.InlineQueryResultVenue*.

Type
str

VIDEO = 'video'

Type of *telegram.InlineQueryResultVideo* and *telegram.InlineQueryResultCachedVideo*.

Type
str

VOICE = 'voice'

Type of *telegram.InlineQueryResultVoice* and *telegram.InlineQueryResultCachedVoice*.

Type
str

class telegram.constants.**InputMediaType**(*value*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.InputMedia*. The enum members of this enumeration are instances of str and can be treated as such.

New in version 20.0.

ANIMATION = 'animation'

Type of *telegram.InputMediaAnimation*.

Type
str

AUDIO = 'audio'

Type of *telegram.InputMediaAudio*.

Type
str

DOCUMENT = 'document'

Type of *telegram.InputMediaDocument*.

Type
str

PHOTO = 'photo'

Type of *telegram.InputMediaPhoto*.

Type
str

VIDEO = 'video'

Type of *telegram.InputMediaVideo*.

Type
str

class telegram.constants.**InvoiceLimit**(*value*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Bot.create_invoice_link()*. The enum members of this enumeration are instances of int and can be treated as such.

New in version 20.0.

MAX_DESCRIPTION_LENGTH = 255

Maximum number of characters of the invoice description.

Type
int

MAX_PAYLOAD_LENGTH = 128

Maximum amount of bytes for the internal payload.

Type
int

MAX_TITLE_LENGTH = 32

Maximum number of characters of the invoice title.

Type
int

MIN_DESCRIPTION_LENGTH = 1

Minimum number of characters of the invoice description.

Type
int

MIN_PAYLOAD_LENGTH = 1

Minimum amount of bytes for the internal payload.

Type
int

MIN_TITLE_LENGTH = 1

Minimum number of characters of the invoice title.

Type
int

class telegram.constants.**LocationLimit**(*value*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Location*/ *telegram.Bot.send_location()*. The enum members of this enumeration are instances of int and can be treated as such.

New in version 20.0.

HEADING = 360

Maximum value allowed for the direction in which the user is moving, in degrees.

Type
int

HORIZONTAL_ACCURACY = 1500

Maximum radius of uncertainty for the location, measured in meters.

Type
int

PROXIMITY_ALERT_RADIUS = 100000

Maximum distance for proximity alerts about approaching another chat member, in meters.

Type
int

class telegram.constants.**MaskPosition**(*value*)

Bases: str, enum.Enum

This enum contains the available positions for *telegram.MaskPosition*. The enum members of this enumeration are instances of str and can be treated as such.

New in version 20.0.

CHIN = 'chin'

Mask position for a sticker on the chin.

Type
str

EYES = 'eyes'

Mask position for a sticker on the eyes.

Type
str

FOREHEAD = 'forehead'

Mask position for a sticker on the forehead.

Type
str

MOUTH = 'mouth'

Mask position for a sticker on the mouth.

Type
str

class telegram.constants.**MenuButtonType**(*value*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.MenuButton*. The enum members of this enumeration are instances of str and can be treated as such.

New in version 20.0.

COMMANDS = 'commands'

The type of *telegram.MenuButtonCommands*.

Type
str

DEFAULT = 'default'

The type of *telegram.MenuButtonDefault*.

Type
str

WEB_APP = 'web_app'

The type of *telegram.MenuButtonWebApp*.

Type
str

class telegram.constants.**MessageAttachmentType**(*value*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.Message* that can be seen as attachment. The enum members of this enumeration are instances of str and can be treated as such.

New in version 20.0.

ANIMATION = 'animation'

Messages with *telegram.Message.animation*.

Type
str

AUDIO = 'audio'

Messages with *telegram.Message.audio*.

Type
str

CONTACT = 'contact'

Messages with *telegram.Message.contact*.

Type
str

DICE = 'dice'

Messages with *telegram.Message.dice*.

Type
str

DOCUMENT = 'document'

Messages with *telegram.Message.document*.

Type
str

GAME = 'game'

Messages with *telegram.Message.game*.

Type
str

INVOICE = 'invoice'

Messages with *telegram.Message.invoice*.

Type
str

LOCATION = 'location'

Messages with *telegram.Message.location*.

Type
str

PASSPORT_DATA = 'passport_data'

Messages with *telegram.Message.passport_data*.

Type
str

PHOTO = 'photo'

Messages with *telegram.Message.photo*.

Type
str

POLL = 'poll'

Messages with *telegram.Message.poll*.

Type
str

STICKER = 'sticker'

Messages with *telegram.Message.sticker*.

Type
str

SUCCESSFUL_PAYMENT = 'successful_payment'

Messages with *telegram.Message.successful_payment*.

Type
str

VENUE = 'venue'

Messages with *telegram.Message.venue*.

Type
str

VIDEO = 'video'

Messages with *telegram.Message.video*.

Type
str

VIDEO_NOTE = 'video_note'

Messages with *telegram.Message.video_note*.

Type
str

VOICE = 'voice'

Messages with *telegram.Message.voice*.

Type
str

class telegram.constants.**MessageEntityType**(*value*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.MessageEntity*. The enum members of this enumeration are instances of str and can be treated as such.

New in version 20.0.

BOLD = 'bold'

Message entities representing bold text.

Type
str

BOT_COMMAND = 'bot_command'

Message entities representing a bot command.

Type
str

CASHTAG = 'cashtag'

Message entities representing a cashtag.

Type
str

CODE = 'code'

Message entities representing monowidth string.

Type
str

CUSTOM_EMOJI = 'custom_emoji'

Message entities representing inline custom emoji stickers.

New in version 20.0.

Type
str

EMAIL = 'email'

Message entities representing a email.

Type
str

HASHTAG = 'hashtag'

Message entities representing a hashtag.

Type
str

ITALIC = 'italic'

Message entities representing italic text.

Type
str

MENTION = 'mention'

Message entities representing a mention.

Type
str

PHONE_NUMBER = 'phone_number'

Message entities representing a phone number.

Type
str

PRE = 'pre'

Message entities representing monowidth block.

Type
str

SPOILER = 'spoiler'

Message entities representing spoiler text.

Type
str

STRIKETHROUGH = 'strikethrough'

Message entities representing strikethrough text.

Type
str

TEXT_LINK = 'text_link'

Message entities representing clickable text URLs.

Type
str

TEXT_MENTION = 'text_mention'

Message entities representing text mention for users without usernames.

Type
str

UNDERLINE = 'underline'

Message entities representing underline text.

Type
str

URL = 'url'

Message entities representing a url.

Type
str

class telegram.constants.**MessageLimit**(*value*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Message*/ *telegram.Bot.send_message()* & friends. The enum members of this enumeration are instances of int and can be treated as such.

New in version 20.0.

CAPTION_LENGTH = 1024

Maximum number of characters for a message caption.

Type
int

MESSAGE_ENTITIES = 100

Maximum number of entities that can be displayed in a message. Further entities will simply be ignored by Telegram.

Note: This value is undocumented and might be changed by Telegram.

Type
int

TEXT_LENGTH = 4096

Maximum number of characters for a text message.

Type
int

class telegram.constants.**MessageType**(*value*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.Message* that can be seen as attachment. The enum members of this enumeration are instances of str and can be treated as such.

New in version 20.0.

ANIMATION = 'animation'

Messages with *telegram.Message.animation*.

Type
str

AUDIO = 'audio'

Messages with *telegram.Message.audio*.

Type
str

CHANNEL_CHAT_CREATED = 'channel_chat_created'

Messages with *telegram.Message.channel_chat_created*.

Type
str

CONTACT = 'contact'

Messages with *telegram.Message.contact*.

Type
str

DELETE_CHAT_PHOTO = 'delete_chat_photo'

Messages with *telegram.Message.delete_chat_photo*.

Type
str

DICE = 'dice'

Messages with *telegram.Message.dice*.

Type
str

DOCUMENT = 'document'

Messages with *telegram.Message.document*.

Type
str

GAME = 'game'

Messages with *telegram.Message.game*.

Type
str

GROUP_CHAT_CREATED = 'group_chat_created'

Messages with *telegram.Message.group_chat_created*.

Type
str

INVOICE = 'invoice'

Messages with *telegram.Message.invoice*.

Type
str

LEFT_CHAT_MEMBER = 'left_chat_member'

Messages with *telegram.Message.left_chat_member*.

Type
str

LOCATION = 'location'

Messages with *telegram.Message.location*.

Type
str

MESSAGE_AUTO_DELETE_TIMER_CHANGED = 'message_auto_delete_timer_changed'

Messages with *telegram.Message.message_auto_delete_timer_changed*.

Type
str

MIGRATE_FROM_CHAT_ID = 'migrate_from_chat_id'

Messages with *telegram.Message.migrate_from_chat_id*.

Type
str

MIGRATE_TO_CHAT_ID = 'migrate_to_chat_id'

Messages with *telegram.Message.migrate_to_chat_id*.

Type
str

NEW_CHAT_MEMBERS = 'new_chat_members'

Messages with *telegram.Message.new_chat_members*.

Type
str

NEW_CHAT_PHOTO = 'new_chat_photo'

Messages with *telegram.Message.new_chat_photo*.

Type
str

NEW_CHAT_TITLE = 'new_chat_title'

Messages with *telegram.Message.new_chat_title*.

Type
str

PASSPORT_DATA = 'passport_data'
Messages with *telegram.Message.passport_data*.

Type
str

PHOTO = 'photo'
Messages with *telegram.Message.photo*.

Type
str

PINNED_MESSAGE = 'pinned_message'
Messages with *telegram.Message.pinned_message*.

Type
str

POLL = 'poll'
Messages with *telegram.Message.poll*.

Type
str

PROXIMITY_ALERT_TRIGGERED = 'proximity_alert_triggered'
Messages with *telegram.Message.proximity_alert_triggered*.

Type
str

STICKER = 'sticker'
Messages with *telegram.Message.sticker*.

Type
str

SUCCESSFUL_PAYMENT = 'successful_payment'
Messages with *telegram.Message.successful_payment*.

Type
str

SUPERGROUP_CHAT_CREATED = 'supergroup_chat_created'
Messages with *telegram.Message.supergroup_chat_created*.

Type
str

TEXT = 'text'
Messages with *telegram.Message.text*.

Type
str

VENUE = 'venue'
Messages with *telegram.Message.venue*.

Type
str

VIDEO = 'video'
Messages with *telegram.Message.video*.

Type
str

VIDEO_CHAT_ENDED = 'video_chat_ended'

Messages with *telegram.Message.video_chat_ended*.

Type
str

VIDEO_CHAT_PARTICIPANTS_INVITED = 'video_chat_participants_invited'

Messages with *telegram.Message.video_chat_participants_invited*.

Type
str

VIDEO_CHAT_SCHEDULED = 'video_chat_scheduled'

Messages with *telegram.Message.video_chat_scheduled*.

Type
str

VIDEO_CHAT_STARTED = 'video_chat_started'

Messages with *telegram.Message.video_chat_started*.

Type
str

VIDEO_NOTE = 'video_note'

Messages with *telegram.Message.video_note*.

Type
str

VOICE = 'voice'

Messages with *telegram.Message.voice*.

Type
str

class telegram.constants.**ParseMode**(*value*)

Bases: str, enum.Enum

This enum contains the available parse modes. The enum members of this enumeration are instances of str and can be treated as such.

New in version 20.0.

HTML = 'HTML'

HTML parse mode.

Type
str

MARKDOWN = 'Markdown'

Markdown parse mode.

Note: *MARKDOWN* is a legacy mode, retained by Telegram for backward compatibility. You should use *MARKDOWN_V2* instead.

Type
str

MARKDOWN_V2 = 'MarkdownV2'

Markdown parse mode version 2.

Type
str

class telegram.constants.**PollLimit**(*value*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Poll*/ *telegram.Bot.send_poll()*. The enum members of this enumeration are instances of int and can be treated as such.

New in version 20.0.

OPTION_LENGTH = 100

Maximum number of characters for each option for the poll.

Type
str

OPTION_NUMBER = 10

Maximum number of available options for the poll.

Type
str

QUESTION_LENGTH = 300

Maximum number of characters of the polls question.

Type
str

class telegram.constants.**PollType**(*value*)

Bases: str, enum.Enum

This enum contains the available types for *telegram.Poll*/ *telegram.Bot.send_poll()*. The enum members of this enumeration are instances of str and can be treated as such.

New in version 20.0.

QUIZ = 'quiz'

quiz polls.

Type
str

REGULAR = 'regular'

regular polls.

Type
str

telegram.constants.**SUPPORTED_WEBHOOK_PORTS = [443, 80, 88, 8443]**

Ports supported by *telegram.Bot.set_webhook.url*.

Type
List[int]

class telegram.constants.**StickerType**(*value*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.Sticker*. The enum members of this enumeration are instances of str and can be treated as such.

New in version 20.0.

CUSTOM_EMOJI = 'custom_emoji'

Custom emoji sticker.

Type
str

MASK = 'mask'

Mask sticker.

Type
str

REGULAR = 'regular'

Regular sticker.

Type
str

class telegram.constants.**UpdateType**(*value*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.Update*. The enum members of this enumeration are instances of str and can be treated as such.

New in version 20.0.

CALLBACK_QUERY = 'callback_query'

Updates with *telegram.Update.callback_query*.

Type
str

CHANNEL_POST = 'channel_post'

Updates with *telegram.Update.channel_post*.

Type
str

CHAT_JOIN_REQUEST = 'chat_join_request'

Updates with *telegram.Update.chat_join_request*.

Type
str

CHAT_MEMBER = 'chat_member'

Updates with *telegram.Update.chat_member*.

Type
str

CHOSEN_INLINE_RESULT = 'chosen_inline_result'

Updates with *telegram.Update.chosen_inline_result*.

Type
str

EDITED_CHANNEL_POST = 'edited_channel_post'

Updates with *telegram.Update.edited_channel_post*.

Type
str

EDITED_MESSAGE = 'edited_message'

Updates with *telegram.Update.edited_message*.

Type
str

INLINE_QUERY = 'inline_query'

Updates with *telegram.Update.inline_query*.

Type
str

MESSAGE = 'message'

Updates with *telegram.Update.message*.

Type
str

MY_CHAT_MEMBER = 'my_chat_member'

Updates with *telegram.Update.my_chat_member*.

Type
str

POLL = 'poll'

Updates with *telegram.Update.poll*.

Type
str

POLL_ANSWER = 'poll_answer'

Updates with *telegram.Update.poll_answer*.

Type
str

PRE_CHECKOUT_QUERY = 'pre_checkout_query'

Updates with *telegram.Update.pre_checkout_query*.

Type
str

SHIPPING_QUERY = 'shipping_query'

Updates with *telegram.Update.shipping_query*.

Type
str

class telegram.constants.**WebhookLimit**(*value*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Bot.set_webhook.secret_token*. The enum members of this enumeration are instances of int and can be treated as such.

New in version 20.0.

MAX_SECRET_TOKEN_LENGTH = 256

Maximum length of the secret token.

Type
int

MIN_SECRET_TOKEN_LENGTH = 1

Minimum length of the secret token.

Type
int

10.3.2 telegram.error Module

This module contains classes that represent Telegram errors.

Changed in version 20.0: Replaced `Unauthorized` by *Forbidden*.

exception telegram.error.**BadRequest**(*message*)

Bases: *telegram.error.NetworkError*

Raised when Telegram could not process the request correctly.

exception telegram.error.**ChatMigrated**(*new_chat_id*)

Bases: *telegram.error.TelegramError*

Raised when the requested group chat migrated to supergroup and has a new chat id.

Parameters

new_chat_id (int) – The new chat id of the group.

new_chat_id

The new chat id of the group.

Type

int

exception telegram.error.**Conflict**(*message*)

Bases: *telegram.error.TelegramError*

Raised when a long poll or webhook conflicts with another one.

exception telegram.error.**Forbidden**(*message*)

Bases: *telegram.error.TelegramError*

Raised when the bot has not enough rights to perform the requested action.

Changed in version 20.0: This class was previously named `Unauthorized`.

exception telegram.error.**InvalidToken**(*message=None*)

Bases: *telegram.error.TelegramError*

Raised when the token is invalid.

Parameters

message (str, optional) – Any additional information about the exception.

New in version 20.0.

exception telegram.error.**NetworkError**(*message*)

Bases: *telegram.error.TelegramError*

Base class for exceptions due to networking errors.

exception telegram.error.**PassportDecryptionError**(*message*)

Bases: *telegram.error.TelegramError*

Something went wrong with decryption.

Changed in version 20.0: This class was previously named `TelegramDecryptionError` and was available via `telegram.TelegramDecryptionError`.

exception telegram.error.**RetryAfter**(*retry_after*)

Bases: *telegram.error.TelegramError*

Raised when flood limits where exceeded.

Changed in version 20.0: *retry_after* is now an integer to comply with the Bot API.

Parameters

retry_after (int) – Time in seconds, after which the bot can retry the request.

retry_after

Time in seconds, after which the bot can retry the request.

Type
int

exception telegram.error.**TelegramError**(*message*)

Bases: Exception

Base class for Telegram errors.

exception telegram.error.**TimedOut**(*message=None*)

Bases: *telegram.error.NetworkError*

Raised when a request took too long to finish.

Parameters
message (str, optional) – Any additional information about the exception.

New in version 20.0.

10.3.3 telegram.helpers Module

This module contains convenience helper functions.

Changed in version 20.0: Previously, the contents of this module were available through the (no longer existing) module `telegram.utils.helpers`.

telegram.helpers.**create_deep_linked_url**(*bot_username*, *payload=None*, *group=False*)

Creates a deep-linked URL for this *bot_username* with the specified *payload*. See https://core.telegram.org/bots#deep-linking to learn more.

The *payload* may consist of the following characters: `A-Z, a-z, 0-9, _, -`

Note: Works well in conjunction with `CommandHandler("start", callback, filters=filters.Regex('payload'))`

Examples

```
create_deep_linked_url(bot.get_me().username, "some-params")
```

See also:

Deeplinking Example

Parameters

- ***bot_username*** (str) – The username to link to
- ***payload*** (str, optional) – Parameters to encode in the created URL
- ***group*** (bool, optional) – If True the user is prompted to select a group to add the bot to. If False, opens a one-on-one conversation with the bot. Defaults to False.

Returns
An URL to start the bot with specific parameters

Return type
str

telegram.helpers.**effective_message_type**(*entity*)

Extracts the type of message as a string identifier from a *telegram.Message* or a *telegram.Update*.

Parameters

entity (*telegram.Update* | *telegram.Message*) – The `update` or `message` to extract from.

Returns

One of *telegram.constants.MessageType* if the entity
contains a message that matches one of those types. `None` otherwise.

Return type
`str` | `None`

telegram.helpers.**escape_markdown**(*text*, *version=1*, *entity_type=None*)

Helper function to escape telegram markup symbols.

Parameters

- ***text*** (`str`) – The text.
- ***version*** (`int` | `str`) – Use to specify the version of telegrams Markdown. Either `1` or `2`. Defaults to `1`.
- ***entity_type*** (`str`, optional) – For the entity types *'pre'*, *'code'* and the link part of *'text_link'*, only certain characters need to be escaped in *'MarkdownV2'*. See the official API documentation for details. Only valid in combination with `version=2`, will be ignored else.

telegram.helpers.**mention_html**(*user_id*, *name*)

Parameters

- ***user_id*** (`int`) – The user's id which you want to mention.
- ***name*** (`str`) – The name the mention is showing.

Returns
The inline mention for the user as HTML.

Return type
`str`

telegram.helpers.**mention_markdown**(*user_id*, *name*, *version=1*)

Parameters

- ***user_id*** (`int`) – The user's id which you want to mention.
- ***name*** (`str`) – The name the mention is showing.
- ***version*** (`int` | `str`) – Use to specify the version of Telegram's Markdown. Either `1` or `2`. Defaults to `1`.

Returns
The inline mention for the user as Markdown.

Return type
`str`

10.3.4 telegram.request Module

New in version 20.0.

telegram.request.BaseRequest

class telegram.request.**BaseRequest**

Bases: AbstractAsyncContextManager, ABC

Abstract interface class that allows python-telegram-bot to make requests to the Bot API. Can be implemented via different asyncio HTTP libraries. An implementation of this class must implement all abstract methods and properties.

Instances of this class can be used as asyncio context managers, where

```
async with request_object:
    # code
```

is roughly equivalent to

```
try:
    await request_object.initialize()
    # code
finally:
    await request_object.shutdown()
```

Tip: JSON encoding and decoding is done with the standard library's `json` by default. To use a custom library for this, you can override *parse_json_payload()* and implement custom logic to encode the keys of *telegram.request.RequestData.parameters*.

New in version 20.0.

DEFAULT_NONE = None

A special object that indicates that an argument of a function was not explicitly passed. Used for the timeout parameters of *post()* and *do_request()*.

Example

When calling `request.post(url)`, `request` should use the default timeouts set on initialization. When calling `request.post(url, connect_timeout=5, read_timeout=None)`, `request` should use `5` for the connect timeout and `None` for the read timeout.

Use `if parameter is (not) BaseRequest.DEFAULT_NONE:` to check if the parameter was set.

Type
object

USER_AGENT = 'python-telegram-bot v20.0a3 (https://python-telegram-bot.org)'

A description that can be used as user agent for requests made to the Bot API.

Type
str

abstract async do_request(*url*, *method*, *request_data=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*)

Makes a request to the Bot API. Must be implemented by a subclass.

Warning: This method will be called by *post()* and *retrieve()*. It should *not* be called manually.

Parameters

- ***url*** (str) – The URL to request.
- ***method*** (str) – HTTP method (i.e. 'POST', 'GET', etc.).
- ***request_data*** (*telegram.request.RequestData*, optional) – An object containing information about parameters and files to upload for the request.
- ***read_timeout*** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a response from Telegram's server instead of the time specified during creating of this object. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a write operation to complete (in terms of a network socket; i.e. POSTing a request or uploading a file) instead of the time specified during creating of this object. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a connection attempt to a server to succeed instead of the time specified during creating of this object. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a connection to become available instead of the time specified during creating of this object. Defaults to *DEFAULT_NONE*.

Returns
The HTTP return code & the payload part of the server response.

Return type
Tuple[int, bytes]

abstract async initialize()

Initialize resources used by this class. Must be implemented by a subclass.

static parse_json_payload(*payload*)

Parse the JSON returned from Telegram.

Tip: By default, this method uses the standard library's json.loads() and errors="replace" in bytes.decode(). You can override it to customize either of these behaviors.

Parameters
payload (bytes) – The UTF-8 encoded JSON payload as returned by Telegram.

Returns
A JSON parsed as Python dict with results.

Return type
dict

Raises
TelegramError – If loading the JSON data failed

async post(*url*, *request_data=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*)

Makes a request to the Bot API handles the return code and parses the answer.

Warning: This method will be called by the methods of *telegram.Bot* and should *not* be called manually.

Parameters

- ***url*** (str) – The URL to request.
- ***request_data*** (*telegram.request.RequestData*, optional) – An object containing information about parameters and files to upload for the request.
- ***read_timeout*** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a response from Telegram's server instead of the time specified during creating of this object. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a write operation to complete (in terms of a network socket; i.e. POSTing a request or uploading a file) instead of the time specified during creating of this object. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a connection attempt to a server to succeed instead of the time specified during creating of this object. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a connection to become available instead of the time specified during creating of this object. Defaults to *DEFAULT_NONE*.

Returns
The JSON response of the Bot API.

Return type
Dict[str, ...]

async retrieve(*url*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*)

Retrieve the contents of a file by its URL.

Warning: This method will be called by the methods of *telegram.Bot* and should *not* be called manually.

Parameters

- ***url*** (str) – The web location we want to retrieve.
- ***read_timeout*** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a response from Telegram's server instead of the time specified during creating of this object. Defaults to *DEFAULT_NONE*.
- ***write_timeout*** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a write operation to complete (in terms of a network socket; i.e. POSTing a request or uploading a file) instead of the time specified during creating of this object. Defaults to *DEFAULT_NONE*.
- ***connect_timeout*** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a connection attempt to a server to succeed instead of the time specified during creating of this object. Defaults to *DEFAULT_NONE*.
- ***pool_timeout*** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a connection to become available instead of the time specified during creating of this object. Defaults to *DEFAULT_NONE*.

Returns
The files contents.

Return type
bytes

abstract async shutdown()
Stop & clear resources used by this class. Must be implemented by a subclass.

telegram.request.RequestData

class telegram.request.**RequestData**(*parameters=None*)
Bases: object

Instances of this class collect the data needed for one request to the Bot API, including all parameters and files to be sent along with the request.

New in version 20.0.

> **Warning:** How exactly instances of this will are created should be considered an implementation detail and not part of PTBs public API. Users should exclusively rely on the documented attributes, properties and methods.

contains_files
Whether this object contains files to be uploaded via multipart/form-data.

Type
bool

property json_parameters
Gives the parameters as mapping of parameter name to the respective JSON encoded value.

> **Tip:** By default, this property uses the standard library's json.dumps(). To use a custom library for JSON encoding, you can directly encode the keys of *parameters* - note that string valued keys should not be JSON encoded.

property json_payload
The *parameters* as UTF-8 encoded JSON payload.

> **Tip:** By default, this property uses the standard library's json.dumps(). To use a custom library for JSON encoding, you can directly encode the keys of *parameters* - note that string valued keys should not be JSON encoded.

property multipart_data
Gives the files contained in this object as mapping of part name to encoded content.

property parameters
Gives the parameters as mapping of parameter name to the parameter value, which can be a single object of type int, float, str or bool or any (possibly nested) composition of lists, tuples and dictionaries, where each entry, key and value is of one of the mentioned types.

parametrized_url(*url*, *encode_kwargs=None*)
Shortcut for attaching the return value of *url_encoded_parameters()* to the *url*.

Parameters
- ***url*** (str) – The URL the parameters will be attached to.

- ***encode_kwargs*** (Dict[str, any], optional) – Additional keyword arguments to pass along to urllib.parse.urlencode().

url_encoded_parameters(*encode_kwargs=None*)

Encodes the parameters with urllib.parse.urlencode().

Parameters

encode_kwargs (Dict[str, any], optional) – Additional keyword arguments to pass along to urllib.parse.urlencode().

telegram.request.HTTPXRequest

class telegram.request.**HTTPXRequest**(*connection_pool_size=1*, *proxy_url=None*, *read_timeout=5.0*, *write_timeout=5.0*, *connect_timeout=5.0*, *pool_timeout=1.0*)

Bases: *telegram.request.BaseRequest*

Implementation of *BaseRequest* using the library httpx.

New in version 20.0.

Parameters

- ***connection_pool_size*** (int, optional) – Number of connections to keep in the connection pool. Defaults to 1.

  Note: Independent of the value, one additional connection will be reserved for *telegram.Bot.get_updates()*.

- ***proxy_url*** (str, optional) – The URL to the proxy server. For example 'http://127.0.0.1:3128' or 'socks5://127.0.0.1:3128'. Defaults to None.

  Note:

  – The proxy URL can also be set via the environment variables HTTPS_PROXY or ALL_PROXY. See the docs of httpx for more info.

  – For Socks5 support, additional dependencies are required. Make sure to install PTB via **pip install python-telegram-bot[socks]** in this case.

  – Socks5 proxies can not be set via environment variables.

- ***read_timeout*** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a response from Telegram's server. This value is used unless a different value is passed to *do_request()*. Defaults to 5.
- ***write_timeout*** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a write operation to complete (in terms of a network socket; i.e. POSTing a request or uploading a file). This value is used unless a different value is passed to *do_request()*. Defaults to 5.
- ***connect_timeout*** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a connection attempt to a server to succeed. This value is used unless a different value is passed to *do_request()*. Defaults to 5.
- ***pool_timeout*** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a connection to become available. This value is used unless a different value is passed to *do_request()*. Defaults to 1.

> **Warning:** With a finite pool timeout, you must expect *telegram.error.TimedOut* exceptions to be thrown when more requests are made simultaneously than there are connections in the connection pool!

async do_request(*url*, *method*, *request_data=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*)

See *BaseRequest.do_request()*.

async initialize()

See *BaseRequest.initialize()*.

async shutdown()

See *BaseRequest.shutdown()*.

10.3.5 telegram.warnings Module

This module contains classes used for warnings issued by this library.

New in version 20.0.

exception telegram.warnings.**PTBDeprecationWarning**

Bases: *telegram.warnings.PTBUserWarning*, DeprecationWarning

Custom warning class for deprecations in this library.

Changed in version 20.0: Renamed TelegramDeprecationWarning to PTBDeprecationWarning.

exception telegram.warnings.**PTBRuntimeWarning**

Bases: *telegram.warnings.PTBUserWarning*, RuntimeWarning

Custom runtime warning class used for warnings in this library.

New in version 20.0.

exception telegram.warnings.**PTBUserWarning**

Bases: UserWarning

Custom user warning class used for warnings in this library.

New in version 20.0.

10.4 Examples

In this section we display small examples to show what a bot written with `python-telegram-bot` looks like. Some bots focus on one specific aspect of the Telegram Bot API while others focus on one of the mechanics of this library. Except for the *rawapibot.py* example, they all use the high-level framework this library provides with the *telegram.ext* submodule.

All examples are licensed under the CC0 License and are therefore fully dedicated to the public domain. You can use them as the base for your own bots without worrying about copyrights.

Do note that we ignore one pythonic convention. Best practice would dictate, in many handler callbacks function signatures, to replace the argument `context` with an underscore, since `context` is an unused local variable in those callbacks. However, since these are examples and not having a name for that argument confuses beginners, we decided to have it present.

10.4.1 echobot.py

This is probably the base for most of the bots made with `python-telegram-bot`. It simply replies to each text message with a message that contains the same text.

10.4.2 timerbot.py

This bot uses the *telegram.ext.JobQueue* class to send timed messages. The user sets a timer by using `/set` command with a specific time, for example `/set 30`. The bot then sets up a job to send a message to that user after 30 seconds. The user can also cancel the timer by sending `/unset`. To learn more about the `JobQueue`, read this wiki article.

10.4.3 conversationbot.py

A common task for a bot is to ask information from the user. In v5.0 of this library, we introduced the *telegram.ext.ConversationHandler* for that exact purpose. This example uses it to retrieve user-information in a conversation-like style. To get a better understanding, take a look at the *state diagram*.

10.4.4 conversationbot2.py

A more complex example of a bot that uses the `ConversationHandler`. It is also more confusing. Good thing there is a *fancy state diagram*. for this one, too!

10.4.5 nestedconversationbot.py

A even more complex example of a bot that uses the nested `ConversationHandler`s. While it's certainly not that complex that you couldn't built it without nested `ConversationHanldler`s, it gives a good impression on how to work with them. Of course, there is a *fancy state diagram* for this example, too!

10.4.6 persistentconversationbot.py

A basic example of a bot store conversation state and user_data over multiple restarts.

10.4.7 inlinekeyboard.py

This example sheds some light on inline keyboards, callback queries and message editing. A wiki site explaining this examples lives here.

10.4.8 inlinekeyboard2.py

A more complex example about inline keyboards, callback queries and message editing. This example showcases how an interactive menu could be build using inline keyboards.

10.4.9 deeplinking.py

A basic example on how to use deeplinking with inline keyboards.

10.4.10 inlinebot.py

A basic example of an inline bot. Don't forget to enable inline mode with @BotFather.

10.4.11 pollbot.py

This example sheds some light on polls, poll answers and the corresponding handlers.

10.4.12 passportbot.py

A basic example of a bot that can accept passports. Use in combination with the *HTML page*. Don't forget to enable and configure payments with @BotFather. Check out this guide on Telegram passports in PTB.

10.4.13 paymentbot.py

A basic example of a bot that can accept payments. Don't forget to enable and configure payments with @BotFather.

10.4.14 errorhandlerbot.py

A basic example on how to set up a custom error handler.

10.4.15 chatmemberbot.py

A basic example on how `(my_)chat_member` updates can be used.

10.4.16 webappbot.py

A basic example of how Telegram WebApps can be used. Use in combination with the *HTML page*. For your convenience, this file is hosted by the PTB team such that you don't need to host it yourself. Uses the iro.js JavaScript library to showcase a user interface that is hard to achieve with native Telegram functionality.

10.4.17 contexttypesbot.py

This example showcases how `telegram.ext.ContextTypes` can be used to customize the `context` argument of handler and job callbacks.

10.4.18 customwebhookbot.py

This example showcases how a custom webhook setup can be used in combination with `telegram.ext.Application`.

10.4.19 arbitrarycallbackdatabot.py

This example showcases how PTBs "arbitrary callback data" feature can be used.

10.4.20 Pure API

The *rawapibot.py* example example uses only the pure, "bare-metal" API wrapper.

arbitrarycallbackdatabot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument, wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""This example showcases how PTBs "arbitrary callback data" feature can be used.

For detailed info on arbitrary callback data, see the wiki page at
https://github.com/python-telegram-bot/python-telegram-bot/wiki/Arbitrary-callback_data
"""
import logging
from typing import List, Tuple, cast

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import InlineKeyboardButton, InlineKeyboardMarkup, Update
from telegram.ext import (
    Application,
    CallbackQueryHandler,
    CommandHandler,
    ContextTypes,
    InvalidCallbackData,
    PicklePersistence,
)

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)


async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
```

(continues on next page)

(continued from previous page)

```
    """Sends a message with 5 inline buttons attached."""
    number_list: List[int] = []
    await update.message.reply_text("Please choose:", reply_markup=build_
↪keyboard(number_list))


async def help_command(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Displays info on how to use the bot."""
    await update.message.reply_text(
        "Use /start to test this bot. Use /clear to clear the stored data so that you␣
↪can see "
        "what happens, if the button data is not available. "
    )


async def clear(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Clears the callback data cache"""
    context.bot.callback_data_cache.clear_callback_data()
    context.bot.callback_data_cache.clear_callback_queries()
    await update.effective_message.reply_text("All clear!")


def build_keyboard(current_list: List[int]) -> InlineKeyboardMarkup:
    """Helper function to build the next inline keyboard."""
    return InlineKeyboardMarkup.from_column(
        [InlineKeyboardButton(str(i), callback_data=(i, current_list)) for i in␣
↪range(1, 6)]
    )


async def list_button(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Parses the CallbackQuery and updates the message text."""
    query = update.callback_query
    await query.answer()
    # Get the data from the callback_data.
    # If you're using a type checker like MyPy, you'll have to use typing.cast
    # to make the checker get the expected type of the callback_data
    number, number_list = cast(Tuple[int, List[int]], query.data)
    # append the number to the list
    number_list.append(number)

    await query.edit_message_text(
        text=f"So far you've selected {number_list}. Choose the next item:",
        reply_markup=build_keyboard(number_list),
    )

    # we can delete the data stored for the query, because we've replaced the buttons
    context.drop_callback_data(query)


async def handle_invalid_button(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
↪ None:
    """Informs the user that the button is no longer available."""
    await update.callback_query.answer()
    await update.effective_message.edit_text(
        "Sorry, I could not process this button click  Please send /start to get a␣
```

(continues on next page)

(continued from previous page)

```
    ↪new keyboard."
    )


def main() -> None:
    """Run the bot."""
    # We use persistence to demonstrate how buttons can still work after the bot was␣
    ↪restarted
    persistence = PicklePersistence(filepath="arbitrarycallbackdatabot")
    # Create the Application and pass it your bot's token.
    application = (
        Application.builder()
        .token("TOKEN")
        .persistence(persistence)
        .arbitrary_callback_data(True)
        .build()
    )

    application.add_handler(CommandHandler("start", start))
    application.add_handler(CommandHandler("help", help_command))
    application.add_handler(CommandHandler("clear", clear))
    application.add_handler(
        CallbackQueryHandler(handle_invalid_button, pattern=InvalidCallbackData)
    )
    application.add_handler(CallbackQueryHandler(list_button))

    # Run the bot until the user presses Ctrl-C
    application.run_polling()


if __name__ == "__main__":
    main()
```

chatmemberbot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument, wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""
Simple Bot to handle '(my_)chat_member' updates.
Greets new users & keeps track of which chats the bot is in.

Usage:
Press Ctrl-C on the command line or send a signal to the process to stop the
bot.
"""

import logging
from typing import Optional, Tuple

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
```

(continues on next page)

(continued from previous page)

```
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import Chat, ChatMember, ChatMemberUpdated, Update
from telegram.constants import ParseMode
from telegram.ext import Application, ChatMemberHandler, CommandHandler, ContextTypes

# Enable logging

logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)

logger = logging.getLogger(__name__)


def extract_status_change(chat_member_update: ChatMemberUpdated) -> Optional[Tuple[bool, bool]]:
    """Takes a ChatMemberUpdated instance and extracts whether the 'old_chat_member' was a member
    of the chat and whether the 'new_chat_member' is a member of the chat. Returns None, if
    the status didn't change.
    """
    status_change = chat_member_update.difference().get("status")
    old_is_member, new_is_member = chat_member_update.difference().get("is_member", (None, None))

    if status_change is None:
        return None

    old_status, new_status = status_change
    was_member = old_status in [
        ChatMember.MEMBER,
        ChatMember.OWNER,
        ChatMember.ADMINISTRATOR,
    ] or (old_status == ChatMember.RESTRICTED and old_is_member is True)
    is_member = new_status in [
        ChatMember.MEMBER,
        ChatMember.OWNER,
        ChatMember.ADMINISTRATOR,
    ] or (new_status == ChatMember.RESTRICTED and new_is_member is True)

    return was_member, is_member


async def track_chats(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Tracks the chats the bot is in."""
    result = extract_status_change(update.my_chat_member)
```

(continues on next page)

(continued from previous page)

```
    if result is None:
        return
    was_member, is_member = result

    # Let's check who is responsible for the change
    cause_name = update.effective_user.full_name

    # Handle chat types differently:
    chat = update.effective_chat
    if chat.type == Chat.PRIVATE:
        if not was_member and is_member:
            logger.info("%s started the bot", cause_name)
            context.bot_data.setdefault("user_ids", set()).add(chat.id)
        elif was_member and not is_member:
            logger.info("%s blocked the bot", cause_name)
            context.bot_data.setdefault("user_ids", set()).discard(chat.id)
    elif chat.type in [Chat.GROUP, Chat.SUPERGROUP]:
        if not was_member and is_member:
            logger.info("%s added the bot to the group %s", cause_name, chat.title)
            context.bot_data.setdefault("group_ids", set()).add(chat.id)
        elif was_member and not is_member:
            logger.info("%s removed the bot from the group %s", cause_name, chat.
↪title)
            context.bot_data.setdefault("group_ids", set()).discard(chat.id)
    else:
        if not was_member and is_member:
            logger.info("%s added the bot to the channel %s", cause_name, chat.title)
            context.bot_data.setdefault("channel_ids", set()).add(chat.id)
        elif was_member and not is_member:
            logger.info("%s removed the bot from the channel %s", cause_name, chat.
↪title)
            context.bot_data.setdefault("channel_ids", set()).discard(chat.id)


async def show_chats(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Shows which chats the bot is in"""
    user_ids = ", ".join(str(uid) for uid in context.bot_data.setdefault("user_ids",␣
↪set()))
    group_ids = ", ".join(str(gid) for gid in context.bot_data.setdefault("group_ids",
↪ set()))
    channel_ids = ", ".join(str(cid) for cid in context.bot_data.setdefault("channel_
↪ids", set()))
    text = (
        f"@{context.bot.username} is currently in a conversation with the user IDs
↪{user_ids}."
        f" Moreover it is a member of the groups with IDs {group_ids} "
        f"and administrator in the channels with IDs {channel_ids}."
    )
    await update.effective_message.reply_text(text)


async def greet_chat_members(update: Update, context: ContextTypes.DEFAULT_TYPE) ->␣
↪None:
    """Greets new users in chats and announces when someone leaves"""
    result = extract_status_change(update.chat_member)
    if result is None:
```

(continues on next page)

(continued from previous page)

```
        return

    was_member, is_member = result
    cause_name = update.chat_member.from_user.mention_html()
    member_name = update.chat_member.new_chat_member.user.mention_html()

    if not was_member and is_member:
        await update.effective_chat.send_message(
            f"{member_name} was added by {cause_name}. Welcome!",
            parse_mode=ParseMode.HTML,
        )
    elif was_member and not is_member:
        await update.effective_chat.send_message(
            f"{member_name} is no longer with us. Thanks a lot, {cause_name} ...",
            parse_mode=ParseMode.HTML,
        )


def main() -> None:
    """Start the bot."""
    # Create the Application and pass it your bot's token.
    application = Application.builder().token("TOKEN").build()

    # Keep track of which chats the bot is in
    application.add_handler(ChatMemberHandler(track_chats, ChatMemberHandler.MY_CHAT_
↪MEMBER))
    application.add_handler(CommandHandler("show_chats", show_chats))

    # Handle members joining/leaving chats.
    application.add_handler(ChatMemberHandler(greet_chat_members, ChatMemberHandler.
↪CHAT_MEMBER))

    # Run the bot until the user presses Ctrl-C
    # We pass 'allowed_updates' handle *all* updates including `chat_member` updates
    # To reset this, simply pass `allowed_updates=[]`
    application.run_polling(allowed_updates=Update.ALL_TYPES)


if __name__ == "__main__":
    main()
```

contexttypesbot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument, wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""
Simple Bot to showcase `telegram.ext.ContextTypes`.

Usage:
Press Ctrl-C on the command line or send a signal to the process to stop the
bot.
"""

```

(continues on next page)

(continued from previous page)

```
import logging
from collections import defaultdict
from typing import DefaultDict, Optional, Set

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import InlineKeyboardButton, InlineKeyboardMarkup, Update
from telegram.constants import ParseMode
from telegram.ext import (
    Application,
    CallbackContext,
    CallbackQueryHandler,
    CommandHandler,
    ContextTypes,
    ExtBot,
    TypeHandler,
)

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)


class ChatData:
    """Custom class for chat_data. Here we store data per message."""

    def __init__(self) -> None:
        self.clicks_per_message: DefaultDict[int, int] = defaultdict(int)


# The [ExtBot, dict, ChatData, dict] is for type checkers like mypy
class CustomContext(CallbackContext[ExtBot, dict, ChatData, dict]):
    """Custom class for context."""

    def __init__(self, application: Application, chat_id: int = None, user_id: int = None):
        super().__init__(application=application, chat_id=chat_id, user_id=user_id)
        self._message_id: Optional[int] = None

    @property
    def bot_user_ids(self) -> Set[int]:
        """Custom shortcut to access a value stored in the bot_data dict"""
```

(continues on next page)

(continued from previous page)

```
        return self.bot_data.setdefault("user_ids", set())

    @property
    def message_clicks(self) -> Optional[int]:
        """Access the number of clicks for the message this context object was built for."""
        if self._message_id:
            return self.chat_data.clicks_per_message[self._message_id]
        return None

    @message_clicks.setter
    def message_clicks(self, value: int) -> None:
        """Allow to change the count"""
        if not self._message_id:
            raise RuntimeError("There is no message associated with this context object.")
        self.chat_data.clicks_per_message[self._message_id] = value

    @classmethod
    def from_update(cls, update: object, application: "Application") -> "CustomContext":
        """Override from_update to set _message_id."""
        # Make sure to call super()
        context = super().from_update(update, application)

        if context.chat_data and isinstance(update, Update) and update.effective_message:
            # pylint: disable=protected-access
            context._message_id = update.effective_message.message_id

        # Remember to return the object
        return context


async def start(update: Update, context: CustomContext) -> None:
    """Display a message with a button."""
    await update.message.reply_html(
        "This button was clicked <i>0</i> times.",
        reply_markup=InlineKeyboardMarkup.from_button(
            InlineKeyboardButton(text="Click me!", callback_data="button")
        ),
    )


async def count_click(update: Update, context: CustomContext) -> None:
    """Update the click count for the message."""
    context.message_clicks += 1
    await update.callback_query.answer()
    await update.effective_message.edit_text(
        f"This button was clicked <i>{context.message_clicks}</i> times.",
        reply_markup=InlineKeyboardMarkup.from_button(
            InlineKeyboardButton(text="Click me!", callback_data="button")
        ),
        parse_mode=ParseMode.HTML,
    )

```

(continues on next page)

(continued from previous page)

```

async def print_users(update: Update, context: CustomContext) -> None:
    """Show which users have been using this bot."""
    await update.message.reply_text(
        "The following user IDs have used this bot: "
        f'{", ".join(map(str, context.bot_user_ids))}'
    )


async def track_users(update: Update, context: CustomContext) -> None:
    """Store the user id of the incoming update, if any."""
    if update.effective_user:
        context.bot_user_ids.add(update.effective_user.id)


def main() -> None:
    """Run the bot."""
    context_types = ContextTypes(context=CustomContext, chat_data=ChatData)
    application = Application.builder().token("TOKEN").context_types(context_types).
↪build()

    # run track_users in its own group to not interfere with the user handlers
    application.add_handler(TypeHandler(Update, track_users), group=-1)
    application.add_handler(CommandHandler("start", start))
    application.add_handler(CallbackQueryHandler(count_click))
    application.add_handler(CommandHandler("print_users", print_users))

    application.run_polling()


if __name__ == "__main__":
    main()
```

conversationbot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument, wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""
First, a few callback functions are defined. Then, those functions are passed to
the Application and registered at their respective places.
Then, the bot is started and runs until we press Ctrl-C on the command line.

Usage:
Example of a bot-user conversation using ConversationHandler.
Send /start to initiate the conversation.
Press Ctrl-C on the command line or send a signal to the process to stop the
bot.
"""

import logging

from telegram import __version__ as TG_VER

```

(continues on next page)

(continued from previous page)

```
try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import ReplyKeyboardMarkup, ReplyKeyboardRemove, Update
from telegram.ext import (
    Application,
    CommandHandler,
    ContextTypes,
    ConversationHandler,
    MessageHandler,
    filters,
)

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)

GENDER, PHOTO, LOCATION, BIO = range(4)


async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Starts the conversation and asks the user about their gender."""
    reply_keyboard = [["Boy", "Girl", "Other"]]

    await update.message.reply_text(
        "Hi! My name is Professor Bot. I will hold a conversation with you. "
        "Send /cancel to stop talking to me.\n\n"
        "Are you a boy or a girl?",
        reply_markup=ReplyKeyboardMarkup(
            reply_keyboard, one_time_keyboard=True, input_field_placeholder="Boy or Girl?"
        ),
    )

    return GENDER


async def gender(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Stores the selected gender and asks for a photo."""
    user = update.message.from_user
    logger.info("Gender of %s: %s", user.first_name, update.message.text)
    await update.message.reply_text(
        "I see! Please send me a photo of yourself, "
        "so I know what you look like, or send /skip if you don't want to.",
        reply_markup=ReplyKeyboardRemove(),
```

(continues on next page)

(continued from previous page)

```
    )

    return PHOTO


async def photo(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Stores the photo and asks for a location."""
    user = update.message.from_user
    photo_file = await update.message.photo[-1].get_file()
    await photo_file.download("user_photo.jpg")
    logger.info("Photo of %s: %s", user.first_name, "user_photo.jpg")
    await update.message.reply_text(
        "Gorgeous! Now, send me your location please, or send /skip if you don't want to."
    )

    return LOCATION


async def skip_photo(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Skips the photo and asks for a location."""
    user = update.message.from_user
    logger.info("User %s did not send a photo.", user.first_name)
    await update.message.reply_text(
        "I bet you look great! Now, send me your location please, or send /skip."
    )

    return LOCATION


async def location(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Stores the location and asks for some info about the user."""
    user = update.message.from_user
    user_location = update.message.location
    logger.info(
        "Location of %s: %f / %f", user.first_name, user_location.latitude, user_location.longitude
    )
    await update.message.reply_text(
        "Maybe I can visit you sometime! At last, tell me something about yourself."
    )

    return BIO


async def skip_location(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Skips the location and asks for info about the user."""
    user = update.message.from_user
    logger.info("User %s did not send a location.", user.first_name)
    await update.message.reply_text(
        "You seem a bit paranoid! At last, tell me something about yourself."
    )

    return BIO


```

(continues on next page)

(continued from previous page)

```
async def bio(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Stores the info about the user and ends the conversation."""
    user = update.message.from_user
    logger.info("Bio of %s: %s", user.first_name, update.message.text)
    await update.message.reply_text("Thank you! I hope we can talk again some day.")

    return ConversationHandler.END


async def cancel(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Cancels and ends the conversation."""
    user = update.message.from_user
    logger.info("User %s canceled the conversation.", user.first_name)
    await update.message.reply_text(
        "Bye! I hope we can talk again some day.", reply_markup=ReplyKeyboardRemove()
    )

    return ConversationHandler.END


def main() -> None:
    """Run the bot."""
    # Create the Application and pass it your bot's token.
    application = Application.builder().token("TOKEN").build()

    # Add conversation handler with the states GENDER, PHOTO, LOCATION and BIO
    conv_handler = ConversationHandler(
        entry_points=[CommandHandler("start", start)],
        states={
            GENDER: [MessageHandler(filters.Regex("^(Boy|Girl|Other)$"), gender)],
            PHOTO: [MessageHandler(filters.PHOTO, photo), CommandHandler("skip", skip_
↪photo)],
            LOCATION: [
                MessageHandler(filters.LOCATION, location),
                CommandHandler("skip", skip_location),
            ],
            BIO: [MessageHandler(filters.TEXT & ~filters.COMMAND, bio)],
        },
        fallbacks=[CommandHandler("cancel", cancel)],
    )

    application.add_handler(conv_handler)

    # Run the bot until the user presses Ctrl-C
    application.run_polling()


if __name__ == "__main__":
    main()
```

State Diagram

conversationbot2.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument, wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""
First, a few callback functions are defined. Then, those functions are passed to
the Application and registered at their respective places.
Then, the bot is started and runs until we press Ctrl-C on the command line.

Usage:
Example of a bot-user conversation using ConversationHandler.
Send /start to initiate the conversation.
Press Ctrl-C on the command line or send a signal to the process to stop the
bot.
"""

import logging
from typing import Dict

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To␣
↪view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import ReplyKeyboardMarkup, ReplyKeyboardRemove, Update
from telegram.ext import (
    Application,
    CommandHandler,
    ContextTypes,
    ConversationHandler,
    MessageHandler,
    filters,
)

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)

CHOOSING, TYPING_REPLY, TYPING_CHOICE = range(3)

reply_keyboard = [
    ["Age", "Favourite colour"],
```

(continues on next page)

(continued from previous page)

```
    ["Number of siblings", "Something else..."],
    ["Done"],
]
markup = ReplyKeyboardMarkup(reply_keyboard, one_time_keyboard=True)


def facts_to_str(user_data: Dict[str, str]) -> str:
    """Helper function for formatting the gathered user info."""
    facts = [f"{key} - {value}" for key, value in user_data.items()]
    return "\n".join(facts).join(["\n", "\n"])


async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Start the conversation and ask user for input."""
    await update.message.reply_text(
        "Hi! My name is Doctor Botter. I will hold a more complex conversation with you. "
        "Why don't you tell me something about yourself?",
        reply_markup=markup,
    )

    return CHOOSING


async def regular_choice(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Ask the user for info about the selected predefined choice."""
    text = update.message.text
    context.user_data["choice"] = text
    await update.message.reply_text(f"Your {text.lower()}? Yes, I would love to hear about that!")

    return TYPING_REPLY


async def custom_choice(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Ask the user for a description of a custom category."""
    await update.message.reply_text(
        'Alright, please send me the category first, for example "Most impressive skill"'
    )

    return TYPING_CHOICE


async def received_information(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Store info provided by user and ask for the next category."""
    user_data = context.user_data
    text = update.message.text
    category = user_data["choice"]
    user_data[category] = text
    del user_data["choice"]

    await update.message.reply_text(
        "Neat! Just so you know, this is what you already told me:"
        f"{facts_to_str(user_data)}You can tell me more, or change your opinion"
```

(continues on next page)

(continued from previous page)

```
        " on something.",
        reply_markup=markup,
    )

    return CHOOSING


async def done(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Display the gathered info and end the conversation."""
    user_data = context.user_data
    if "choice" in user_data:
        del user_data["choice"]

    await update.message.reply_text(
        f"I learned these facts about you: {facts_to_str(user_data)}Until next time!",
        reply_markup=ReplyKeyboardRemove(),
    )

    user_data.clear()
    return ConversationHandler.END


def main() -> None:
    """Run the bot."""
    # Create the Application and pass it your bot's token.
    application = Application.builder().token("TOKEN").build()

    # Add conversation handler with the states CHOOSING, TYPING_CHOICE and TYPING_
↪REPLY
    conv_handler = ConversationHandler(
        entry_points=[CommandHandler("start", start)],
        states={
            CHOOSING: [
                MessageHandler(
                    filters.Regex("^(Age|Favourite colour|Number of siblings)$"),␣
↪regular_choice
                ),
                MessageHandler(filters.Regex("^Something else...$"), custom_choice),
            ],
            TYPING_CHOICE: [
                MessageHandler(
                    filters.TEXT & ~(filters.COMMAND | filters.Regex("^Done$")),␣
↪regular_choice
                )
            ],
            TYPING_REPLY: [
                MessageHandler(
                    filters.TEXT & ~(filters.COMMAND | filters.Regex("^Done$")),
                    received_information,
                )
            ],
        },
        fallbacks=[MessageHandler(filters.Regex("^Done$"), done)],
    )

    application.add_handler(conv_handler)
```

(continues on next page)

(continued from previous page)

```
    # Run the bot until the user presses Ctrl-C
    application.run_polling()


if __name__ == "__main__":
    main()
```

State Diagram

customwebhookbot.py

```
#!/usr/bin/env python
# This program is dedicated to the public domain under the CC0 license.
# pylint: disable=import-error,wrong-import-position
"""
Simple example of a bot that uses a custom webhook setup and handles custom updates.
For the custom webhook setup, the libraries `starlette` and `uvicorn` are used. Please install
them as `pip install starlette~=0.20.0 uvicorn~=0.17.0`.
Note that any other `asyncio` based web server framework can be used for a custom webhook setup
just as well.

Usage:
Set bot token, url, admin chat_id and port at the start of the `main` function.
You may also need to change the `listen` value in the uvicorn configuration to match your setup.
Press Ctrl-C on the command line or send a signal to the process to stop the bot.
"""
import asyncio
import html
import logging
from dataclasses import dataclass
from http import HTTPStatus

import uvicorn
from starlette.applications import Starlette
from starlette.requests import Request
from starlette.responses import PlainTextResponse, Response
from starlette.routing import Route

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To view the "
        f"{TG_VER} version of this example, "
```

(continues on next page)

(continued from previous page)

```
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )

from telegram import Update
from telegram.constants import ParseMode
from telegram.ext import (
    Application,
    CallbackContext,
    CommandHandler,
    ContextTypes,
    ExtBot,
    TypeHandler,
)

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)


@dataclass
class WebhookUpdate:
    """Simple dataclass to wrap a custom update type"""

    user_id: int
    payload: str


class CustomContext(CallbackContext[ExtBot, dict, dict, dict]):
    """
    Custom CallbackContext class that makes `user_data` available for updates of type
    `WebhookUpdate`.
    """

    @classmethod
    def from_update(
        cls,
        update: object,
        application: "Application",
    ) -> "CustomContext":
        if isinstance(update, WebhookUpdate):
            return cls(application=application, user_id=update.user_id)
        return super().from_update(update, application)


async def start(update: Update, context: CustomContext) -> None:
    """Display a message with instructions on how to use this bot."""
    url = context.bot_data["url"]
    payload_url = html.escape(f"{url}/submitpayload?user_id=<your user id>&payload=
↪<payload>")
    text = (
        f"To check if the bot is still running, call <code>{url}/healthcheck</code>.\
↪n\n"
        f"To post a custom update, call <code>{payload_url}</code>."
    )
```

(continues on next page)

(continued from previous page)

```
    await update.message.reply_html(text=text)


async def webhook_update(update: WebhookUpdate, context: CustomContext) -> None:
    """Callback that handles the custom updates."""
    chat_member = await context.bot.get_chat_member(chat_id=update.user_id, user_
↪id=update.user_id)
    payloads = context.user_data.setdefault("payloads", [])
    payloads.append(update.payload)
    combined_payloads = "</code>\n• <code>".join(payloads)
    text = (
        f"The user {chat_member.user.mention_html()} has sent a new payload. "
        f"So far they have sent the following payloads: \n\n• <code>{combined_
↪payloads}</code>"
    )
    await context.bot.send_message(
        chat_id=context.bot_data["admin_chat_id"], text=text, parse_mode=ParseMode.
↪HTML
    )


async def main() -> None:
    """Set up the application and a custom webserver."""
    url = "https://domain.tld"
    admin_chat_id = 123456
    port = 8000

    context_types = ContextTypes(context=CustomContext)
    # Here we set updater to None because we want our custom webhook server to handle␣
↪the updates
    # and hence we don't need an Updater instance
    application = (
        Application.builder().token("TOKEN").updater(None).context_types(context_
↪types).build()
    )
    # save the values in `bot_data` such that we may easily access them in the␣
↪callbacks
    application.bot_data["url"] = url
    application.bot_data["admin_chat_id"] = admin_chat_id

    # register handlers
    application.add_handler(CommandHandler("start", start))
    application.add_handler(TypeHandler(type=WebhookUpdate, callback=webhook_update))

    # Pass webhook settings to telegram
    await application.bot.set_webhook(url=f"{url}/telegram")

    # Set up webserver
    async def telegram(request: Request) -> Response:
        """Handle incoming Telegram updates by putting them into the `update_queue`"""
        await application.update_queue.put(
            Update.de_json(data=await request.json(), bot=application.bot)
        )
        return Response()

    async def custom_updates(request: Request) -> PlainTextResponse:
```

(continues on next page)

(continued from previous page)

```
        """
        Handle incoming webhook updates by also putting them into the `update_queue` if
        the required parameters were passed correctly.
        """
        try:
            user_id = int(request.query_params["user_id"])
            payload = request.query_params["payload"]
        except KeyError:
            return PlainTextResponse(
                status_code=HTTPStatus.BAD_REQUEST,
                content="Please pass both `user_id` and `payload` as query parameters.",
            )
        except ValueError:
            return PlainTextResponse(
                status_code=HTTPStatus.BAD_REQUEST,
                content="The `user_id` must be a string!",
            )

        await application.update_queue.put(WebhookUpdate(user_id=user_id, payload=payload))
        return PlainTextResponse("Thank you for the submission! It's being forwarded.")

    async def health(_: Request) -> PlainTextResponse:
        """For the health endpoint, reply with a simple plain text message."""
        return PlainTextResponse(content="The bot is still running fine :)")

    starlette_app = Starlette(
        routes=[
            Route("/telegram", telegram, methods=["POST"]),
            Route("/healthcheck", health, methods=["GET"]),
            Route("/submitpayload", custom_updates, methods=["POST", "GET"]),
        ]
    )
    webserver = uvicorn.Server(
        config=uvicorn.Config(
            app=starlette_app,
            port=port,
            use_colors=False,
            host="127.0.0.1",
        )
    )

    # Run application and webserver together
    async with application:
        await application.start()
        await webserver.serve()
        await application.stop()


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

deeplinking.py

```python
#!/usr/bin/env python
# pylint: disable=unused-argument, wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""Bot that explains Telegram's "Deep Linking Parameters" functionality.

This program is dedicated to the public domain under the CC0 license.

This Bot uses the Application class to handle the bot.

First, a few handler functions are defined. Then, those functions are passed to
the Application and registered at their respective places.
Then, the bot is started and runs until we press Ctrl-C on the command line.

Usage:
Deep Linking example. Send /start to get the link.
Press Ctrl-C on the command line or send a signal to the process to stop the
bot.
"""

import logging

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To␣
↪view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import InlineKeyboardButton, InlineKeyboardMarkup, Update, helpers
from telegram.constants import ParseMode
from telegram.ext import Application, CallbackQueryHandler, CommandHandler,␣
↪ContextTypes, filters

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)

logger = logging.getLogger(__name__)

# Define constants that will allow us to reuse the deep-linking parameters.
CHECK_THIS_OUT = "check-this-out"
USING_ENTITIES = "using-entities-here"
USING_KEYBOARD = "using-keyboard-here"
SO_COOL = "so-cool"

# Callback data to pass in 3rd level deep-linking
```

(continues on next page)

(continued from previous page)

```
KEYBOARD_CALLBACKDATA = "keyboard-callback-data"


async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Send a deep-linked URL when the command /start is issued."""
    bot = context.bot
    url = helpers.create_deep_linked_url(bot.username, CHECK_THIS_OUT, group=True)
    text = "Feel free to tell your friends about it:\n\n" + url
    await update.message.reply_text(text)


async def deep_linked_level_1(update: Update, context: ContextTypes.DEFAULT_TYPE) ->␣
↪None:
    """Reached through the CHECK_THIS_OUT payload"""
    bot = context.bot
    url = helpers.create_deep_linked_url(bot.username, SO_COOL)
    text = (
        "Awesome, you just accessed hidden functionality! "
        "Now let's get back to the private chat."
    )
    keyboard = InlineKeyboardMarkup.from_button(
        InlineKeyboardButton(text="Continue here!", url=url)
    )
    await update.message.reply_text(text, reply_markup=keyboard)


async def deep_linked_level_2(update: Update, context: ContextTypes.DEFAULT_TYPE) ->␣
↪None:
    """Reached through the SO_COOL payload"""
    bot = context.bot
    url = helpers.create_deep_linked_url(bot.username, USING_ENTITIES)
    text = f'You can also mask the deep-linked URLs as links: <a href="{url}"> CLICK␣
↪HERE</a>.'
    await update.message.reply_text(text, parse_mode=ParseMode.HTML, disable_web_page_
↪preview=True)


async def deep_linked_level_3(update: Update, context: ContextTypes.DEFAULT_TYPE) ->␣
↪None:
    """Reached through the USING_ENTITIES payload"""
    await update.message.reply_text(
        "It is also possible to make deep-linking using InlineKeyboardButtons.",
        reply_markup=InlineKeyboardMarkup(
            [[InlineKeyboardButton(text="Like this!", callback_data=KEYBOARD_
↪CALLBACKDATA)]]
        ),
    )


async def deep_link_level_3_callback(update: Update, context: ContextTypes.DEFAULT_
↪TYPE) -> None:
    """Answers CallbackQuery with deeplinking url."""
    bot = context.bot
    url = helpers.create_deep_linked_url(bot.username, USING_KEYBOARD)
    await update.callback_query.answer(url=url)


```

(continues on next page)

(continued from previous page)

```
103
104 async def deep_linked_level_4(update: Update, context: ContextTypes.DEFAULT_TYPE) ->␣
    ↪None:
105     """Reached through the USING_KEYBOARD payload"""
106     payload = context.args
107     await update.message.reply_text(
108         f"Congratulations! This is as deep as it gets \n\nThe payload was: {payload}"
109     )
110
111
112 def main() -> None:
113     """Start the bot."""
114     # Create the Application and pass it your bot's token.
115     application = Application.builder().token("TOKEN").build()
116
117     # More info on what deep linking actually is (read this first if it's unclear to␣
    ↪you):
118     # https://core.telegram.org/bots#deep-linking
119
120     # Register a deep-linking handler
121     application.add_handler(
122         CommandHandler("start", deep_linked_level_1, filters.Regex(CHECK_THIS_OUT))
123     )
124
125     # This one works with a textual link instead of an URL
126     application.add_handler(CommandHandler("start", deep_linked_level_2, filters.
    ↪Regex(SO_COOL)))
127
128     # We can also pass on the deep-linking payload
129     application.add_handler(
130         CommandHandler("start", deep_linked_level_3, filters.Regex(USING_ENTITIES))
131     )
132
133     # Possible with inline keyboard buttons as well
134     application.add_handler(
135         CommandHandler("start", deep_linked_level_4, filters.Regex(USING_KEYBOARD))
136     )
137
138     # register callback handler for inline keyboard button
139     application.add_handler(
140         CallbackQueryHandler(deep_link_level_3_callback, pattern=KEYBOARD_
    ↪CALLBACKDATA)
141     )
142
143     # Make sure the deep-linking handlers occur *before* the normal /start handler.
144     application.add_handler(CommandHandler("start", start))
145
146     # Run the bot until the user presses Ctrl-C
147     application.run_polling()
148
149
150 if __name__ == "__main__":
151     main()
```

echobot.py

```python
#!/usr/bin/env python
# pylint: disable=unused-argument, wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""
Simple Bot to reply to Telegram messages.

First, a few handler functions are defined. Then, those functions are passed to
the Application and registered at their respective places.
Then, the bot is started and runs until we press Ctrl-C on the command line.

Usage:
Basic Echobot example, repeats messages.
Press Ctrl-C on the command line or send a signal to the process to stop the
bot.
"""

import logging

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import ForceReply, Update
from telegram.ext import Application, CommandHandler, ContextTypes, MessageHandler, filters

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)


# Define a few command handlers. These usually take the two arguments update and
# context.
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Send a message when the command /start is issued."""
    user = update.effective_user
    await update.message.reply_html(
        rf"Hi {user.mention_html()}!",
        reply_markup=ForceReply(selective=True),
    )


```

(continues on next page)

(continued from previous page)

```python
async def help_command(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Send a message when the command /help is issued."""
    await update.message.reply_text("Help!")


async def echo(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Echo the user message."""
    await update.message.reply_text(update.message.text)


def main() -> None:
    """Start the bot."""
    # Create the Application and pass it your bot's token.
    application = Application.builder().token("TOKEN").build()

    # on different commands - answer in Telegram
    application.add_handler(CommandHandler("start", start))
    application.add_handler(CommandHandler("help", help_command))

    # on non command i.e message - echo the message on Telegram
    application.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, echo))

    # Run the bot until the user presses Ctrl-C
    application.run_polling()


if __name__ == "__main__":
    main()
```

errorhandlerbot.py

```python
#!/usr/bin/env python
# pylint: disable=unused-argument, wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""This is a very simple example on how one could implement a custom error handler."""
import html
import json
import logging
import traceback

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To␣
↪view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
```

(continues on next page)

(continued from previous page)

```
from telegram import Update
from telegram.constants import ParseMode
from telegram.ext import Application, CommandHandler, ContextTypes

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)

# This can be your own ID, or one for a developer group/channel.
# You can use the /start command of this bot to see your chat id.
DEVELOPER_CHAT_ID = 123456789


async def error_handler(update: object, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Log the error and send a telegram message to notify the developer."""
    # Log the error before we do anything else, so we can see it even if something␣
↪breaks.
    logger.error(msg="Exception while handling an update:", exc_info=context.error)

    # traceback.format_exception returns the usual python message about an exception,␣
↪but as a
    # list of strings rather than a single string, so we have to join them together.
    tb_list = traceback.format_exception(None, context.error, context.error.__
↪traceback__)
    tb_string = "".join(tb_list)

    # Build the message with some markup and additional information about what␣
↪happened.
    # You might need to add some logic to deal with messages longer than the 4096␣
↪character limit.
    update_str = update.to_dict() if isinstance(update, Update) else str(update)
    message = (
        f"An exception was raised while handling an update\n"
        f"<pre>update = {html.escape(json.dumps(update_str, indent=2, ensure_
↪ascii=False))}"
        "</pre>\n\n"
        f"<pre>context.chat_data = {html.escape(str(context.chat_data))}</pre>\n\n"
        f"<pre>context.user_data = {html.escape(str(context.user_data))}</pre>\n\n"
        f"<pre>{html.escape(tb_string)}</pre>"
    )

    # Finally, send the message
    await context.bot.send_message(
        chat_id=DEVELOPER_CHAT_ID, text=message, parse_mode=ParseMode.HTML
    )


async def bad_command(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Raise an error to trigger the error handler."""
    await context.bot.wrong_method_name()  # type: ignore[attr-defined]


async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Displays info on how to trigger an error."""
```

(continues on next page)

(continued from previous page)

```
    await update.effective_message.reply_html(
        "Use /bad_command to cause an error.\n"
        f"Your chat id is <code>{update.effective_chat.id}</code>."
    )


def main() -> None:
    """Run the bot."""
    # Create the Application and pass it your bot's token.
    application = Application.builder().token("TOKEN").build()

    # Register the commands...
    application.add_handler(CommandHandler("start", start))
    application.add_handler(CommandHandler("bad_command", bad_command))

    # ...and the error handler
    application.add_error_handler(error_handler)

    # Run the bot until the user presses Ctrl-C
    application.run_polling()


if __name__ == "__main__":
    main()
```

inlinebot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument, wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""
Don't forget to enable inline mode with @BotFather

First, a few handler functions are defined. Then, those functions are passed to
the Application and registered at their respective places.
Then, the bot is started and runs until we press Ctrl-C on the command line.

Usage:
Basic inline bot example. Applies different text transformations.
Press Ctrl-C on the command line or send a signal to the process to stop the
bot.
"""
import logging
from html import escape
from uuid import uuid4

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
```

(continues on next page)

(continued from previous page)

```
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import InlineQueryResultArticle, InputTextMessageContent, Update
from telegram.constants import ParseMode
from telegram.ext import Application, CommandHandler, ContextTypes, InlineQueryHandler

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)


# Define a few command handlers. These usually take the two arguments update and
# context.
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Send a message when the command /start is issued."""
    await update.message.reply_text("Hi!")


async def help_command(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Send a message when the command /help is issued."""
    await update.message.reply_text("Help!")


async def inline_query(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Handle the inline query. This is run when you type: @botusername <query>"""
    query = update.inline_query.query

    if query == "":
        return

    results = [
        InlineQueryResultArticle(
            id=str(uuid4()),
            title="Caps",
            input_message_content=InputTextMessageContent(query.upper()),
        ),
        InlineQueryResultArticle(
            id=str(uuid4()),
            title="Bold",
            input_message_content=InputTextMessageContent(
                f"<b>{escape(query)}</b>", parse_mode=ParseMode.HTML
            ),
        ),
        InlineQueryResultArticle(
            id=str(uuid4()),
            title="Italic",
            input_message_content=InputTextMessageContent(
                f"<i>{escape(query)}</i>", parse_mode=ParseMode.HTML
            ),
        ),
```

(continues on next page)

(continued from previous page)

```
    ]

    await update.inline_query.answer(results)


def main() -> None:
    """Run the bot."""
    # Create the Application and pass it your bot's token.
    application = Application.builder().token("TOKEN").build()

    # on different commands - answer in Telegram
    application.add_handler(CommandHandler("start", start))
    application.add_handler(CommandHandler("help", help_command))

    # on non command i.e message - echo the message on Telegram
    application.add_handler(InlineQueryHandler(inline_query))

    # Run the bot until the user presses Ctrl-C
    application.run_polling()


if __name__ == "__main__":
    main()
```

inlinekeyboard.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument, wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""
Basic example for a bot that uses inline keyboards. For an in-depth explanation, check out
 https://github.com/python-telegram-bot/python-telegram-bot/wiki/InlineKeyboard-Example.
"""
import logging

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import InlineKeyboardButton, InlineKeyboardMarkup, Update
from telegram.ext import Application, CallbackQueryHandler, CommandHandler, ContextTypes
```

(continues on next page)

(continued from previous page)

```

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)


async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Sends a message with three inline buttons attached."""
    keyboard = [
        [
            InlineKeyboardButton("Option 1", callback_data="1"),
            InlineKeyboardButton("Option 2", callback_data="2"),
        ],
        [InlineKeyboardButton("Option 3", callback_data="3")],
    ]

    reply_markup = InlineKeyboardMarkup(keyboard)

    await update.message.reply_text("Please choose:", reply_markup=reply_markup)


async def button(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Parses the CallbackQuery and updates the message text."""
    query = update.callback_query

    # CallbackQueries need to be answered, even if no notification to the user is␣
↪needed
    # Some clients may have trouble otherwise. See https://core.telegram.org/bots/api
↪#callbackquery
    await query.answer()

    await query.edit_message_text(text=f"Selected option: {query.data}")


async def help_command(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Displays info on how to use the bot."""
    await update.message.reply_text("Use /start to test this bot.")


def main() -> None:
    """Run the bot."""
    # Create the Application and pass it your bot's token.
    application = Application.builder().token("TOKEN").build()

    application.add_handler(CommandHandler("start", start))
    application.add_handler(CallbackQueryHandler(button))
    application.add_handler(CommandHandler("help", help_command))

    # Run the bot until the user presses Ctrl-C
    application.run_polling()


if __name__ == "__main__":
    main()
```

inlinekeyboard2.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument, wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""Simple inline keyboard bot with multiple CallbackQueryHandlers.

This Bot uses the Application class to handle the bot.
First, a few callback functions are defined as callback query handler. Then, those functions are
passed to the Application and registered at their respective places.
Then, the bot is started and runs until we press Ctrl-C on the command line.
Usage:
Example of a bot that uses inline keyboard that has multiple CallbackQueryHandlers arranged in a
ConversationHandler.
Send /start to initiate the conversation.
Press Ctrl-C on the command line to stop the bot.
"""
import logging

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import InlineKeyboardButton, InlineKeyboardMarkup, Update
from telegram.ext import (
    Application,
    CallbackQueryHandler,
    CommandHandler,
    ContextTypes,
    ConversationHandler,
)

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)

# Stages
START_ROUTES, END_ROUTES = range(2)
# Callback data
ONE, TWO, THREE, FOUR = range(4)


```

(continues on next page)

(continued from previous page)

```
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Send message on `/start`."""
    # Get user that sent /start and log his name
    user = update.message.from_user
    logger.info("User %s started the conversation.", user.first_name)
    # Build InlineKeyboard where each button has a displayed text
    # and a string as callback_data
    # The keyboard is a list of button rows, where each row is in turn
    # a list (hence `[[...]]`).
    keyboard = [
        [
            InlineKeyboardButton("1", callback_data=str(ONE)),
            InlineKeyboardButton("2", callback_data=str(TWO)),
        ]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    # Send message with text and appended InlineKeyboard
    await update.message.reply_text("Start handler, Choose a route", reply_
↪markup=reply_markup)
    # Tell ConversationHandler that we're in state `FIRST` now
    return START_ROUTES


async def start_over(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Prompt same text & keyboard as `start` does but not as new message"""
    # Get CallbackQuery from Update
    query = update.callback_query
    # CallbackQueries need to be answered, even if no notification to the user is␣
↪needed
    # Some clients may have trouble otherwise. See https://core.telegram.org/bots/api
↪#callbackquery
    await query.answer()
    keyboard = [
        [
            InlineKeyboardButton("1", callback_data=str(ONE)),
            InlineKeyboardButton("2", callback_data=str(TWO)),
        ]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    # Instead of sending a new message, edit the message that
    # originated the CallbackQuery. This gives the feeling of an
    # interactive menu.
    await query.edit_message_text(text="Start handler, Choose a route", reply_
↪markup=reply_markup)
    return START_ROUTES


async def one(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Show new choice of buttons"""
    query = update.callback_query
    await query.answer()
    keyboard = [
        [
            InlineKeyboardButton("3", callback_data=str(THREE)),
            InlineKeyboardButton("4", callback_data=str(FOUR)),
        ]
```

(continues on next page)

(continued from previous page)

```
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await query.edit_message_text(
        text="First CallbackQueryHandler, Choose a route", reply_markup=reply_markup
    )
    return START_ROUTES


async def two(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Show new choice of buttons"""
    query = update.callback_query
    await query.answer()
    keyboard = [
        [
            InlineKeyboardButton("1", callback_data=str(ONE)),
            InlineKeyboardButton("3", callback_data=str(THREE)),
        ]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await query.edit_message_text(
        text="Second CallbackQueryHandler, Choose a route", reply_markup=reply_markup
    )
    return START_ROUTES


async def three(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Show new choice of buttons. This is the end point of the conversation."""
    query = update.callback_query
    await query.answer()
    keyboard = [
        [
            InlineKeyboardButton("Yes, let's do it again!", callback_data=str(ONE)),
            InlineKeyboardButton("Nah, I've had enough ...", callback_data=str(TWO)),
        ]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await query.edit_message_text(
        text="Third CallbackQueryHandler. Do want to start over?", reply_markup=reply_
↪markup
    )
    # Transfer to conversation state `SECOND`
    return END_ROUTES


async def four(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Show new choice of buttons"""
    query = update.callback_query
    await query.answer()
    keyboard = [
        [
            InlineKeyboardButton("2", callback_data=str(TWO)),
            InlineKeyboardButton("3", callback_data=str(THREE)),
        ]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await query.edit_message_text(
```

(continues on next page)

(continued from previous page)

```
        text="Fourth CallbackQueryHandler, Choose a route", reply_markup=reply_markup
    )
    return START_ROUTES


async def end(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Returns `ConversationHandler.END`, which tells the
    ConversationHandler that the conversation is over.
    """
    query = update.callback_query
    await query.answer()
    await query.edit_message_text(text="See you next time!")
    return ConversationHandler.END


def main() -> None:
    """Run the bot."""
    # Create the Application and pass it your bot's token.
    application = Application.builder().token("TOKEN").build()

    # Setup conversation handler with the states FIRST and SECOND
    # Use the pattern parameter to pass CallbackQueries with specific
    # data pattern to the corresponding handlers.
    # ^ means "start of line/string"
    # $ means "end of line/string"
    # So ^ABC$ will only allow 'ABC'
    conv_handler = ConversationHandler(
        entry_points=[CommandHandler("start", start)],
        states={
            START_ROUTES: [
                CallbackQueryHandler(one, pattern="^" + str(ONE) + "$"),
                CallbackQueryHandler(two, pattern="^" + str(TWO) + "$"),
                CallbackQueryHandler(three, pattern="^" + str(THREE) + "$"),
                CallbackQueryHandler(four, pattern="^" + str(FOUR) + "$"),
            ],
            END_ROUTES: [
                CallbackQueryHandler(start_over, pattern="^" + str(ONE) + "$"),
                CallbackQueryHandler(end, pattern="^" + str(TWO) + "$"),
            ],
        },
        fallbacks=[CommandHandler("start", start)],
    )

    # Add ConversationHandler to application that will be used for handling updates
    application.add_handler(conv_handler)

    # Run the bot until the user presses Ctrl-C
    application.run_polling()


if __name__ == "__main__":
    main()
```

nestedconversationbot.py

```python
#!/usr/bin/env python
# pylint: disable=unused-argument, wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""
First, a few callback functions are defined. Then, those functions are passed to
the Application and registered at their respective places.
Then, the bot is started and runs until we press Ctrl-C on the command line.

Usage:
Example of a bot-user conversation using nested ConversationHandlers.
Send /start to initiate the conversation.
Press Ctrl-C on the command line or send a signal to the process to stop the
bot.
"""

import logging
from typing import Any, Dict, Tuple

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To␣
↪view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import InlineKeyboardButton, InlineKeyboardMarkup, Update
from telegram.ext import (
    Application,
    CallbackQueryHandler,
    CommandHandler,
    ContextTypes,
    ConversationHandler,
    MessageHandler,
    filters,
)

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)

# State definitions for top level conversation
SELECTING_ACTION, ADDING_MEMBER, ADDING_SELF, DESCRIBING_SELF = map(chr, range(4))
# State definitions for second level conversation
SELECTING_LEVEL, SELECTING_GENDER = map(chr, range(4, 6))
# State definitions for descriptions conversation
```

(continues on next page)

(continued from previous page)

```
SELECTING_FEATURE, TYPING = map(chr, range(6, 8))
# Meta states
STOPPING, SHOWING = map(chr, range(8, 10))
# Shortcut for ConversationHandler.END
END = ConversationHandler.END

# Different constants for this example
(
    PARENTS,
    CHILDREN,
    SELF,
    GENDER,
    MALE,
    FEMALE,
    AGE,
    NAME,
    START_OVER,
    FEATURES,
    CURRENT_FEATURE,
    CURRENT_LEVEL,
) = map(chr, range(10, 22))


# Helper
def _name_switcher(level: str) -> Tuple[str, str]:
    if level == PARENTS:
        return "Father", "Mother"
    return "Brother", "Sister"


# Top level conversation callbacks
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
    """Select an action: Adding parent/child or show data."""
    text = (
        "You may choose to add a family member, yourself, show the gathered data, or␣
↪end the "
        "conversation. To abort, simply type /stop."
    )

    buttons = [
        [
            InlineKeyboardButton(text="Add family member", callback_data=str(ADDING_
↪MEMBER)),
            InlineKeyboardButton(text="Add yourself", callback_data=str(ADDING_SELF)),
        ],
        [
            InlineKeyboardButton(text="Show data", callback_data=str(SHOWING)),
            InlineKeyboardButton(text="Done", callback_data=str(END)),
        ],
    ]
    keyboard = InlineKeyboardMarkup(buttons)

    # If we're starting over we don't need to send a new message
    if context.user_data.get(START_OVER):
        await update.callback_query.answer()
        await update.callback_query.edit_message_text(text=text, reply_
```

(continues on next page)

(continued from previous page)

```
↪markup=keyboard)
    else:
        await update.message.reply_text(
            "Hi, I'm Family Bot and I'm here to help you gather information about␣
↪your family."
        )
        await update.message.reply_text(text=text, reply_markup=keyboard)

    context.user_data[START_OVER] = False
    return SELECTING_ACTION


async def adding_self(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
    """Add information about yourself."""
    context.user_data[CURRENT_LEVEL] = SELF
    text = "Okay, please tell me about yourself."
    button = InlineKeyboardButton(text="Add info", callback_data=str(MALE))
    keyboard = InlineKeyboardMarkup.from_button(button)

    await update.callback_query.answer()
    await update.callback_query.edit_message_text(text=text, reply_markup=keyboard)

    return DESCRIBING_SELF


async def show_data(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
    """Pretty print gathered data."""

    def pretty_print(data: Dict[str, Any], level: str) -> str:
        people = data.get(level)
        if not people:
            return "\nNo information yet."

        return_str = ""
        if level == SELF:
            for person in data[level]:
                return_str += f"\nName: {person.get(NAME, '-')}, Age: {person.get(AGE,
↪ '-')}"
        else:
            male, female = _name_switcher(level)

            for person in data[level]:
                gender = female if person[GENDER] == FEMALE else male
                return_str += (
                    f"\n{gender}: Name: {person.get(NAME, '-')}, Age: {person.get(AGE,
↪ '-')}"
                )
        return return_str

    user_data = context.user_data
    text = f"Yourself:{pretty_print(user_data, SELF)}"
    text += f"\n\nParents:{pretty_print(user_data, PARENTS)}"
    text += f"\n\nChildren:{pretty_print(user_data, CHILDREN)}"

    buttons = [[InlineKeyboardButton(text="Back", callback_data=str(END))]]
    keyboard = InlineKeyboardMarkup(buttons)
```

(continues on next page)

(continued from previous page)

```
    await update.callback_query.answer()
    await update.callback_query.edit_message_text(text=text, reply_markup=keyboard)
    user_data[START_OVER] = True

    return SHOWING


async def stop(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """End Conversation by command."""
    await update.message.reply_text("Okay, bye.")

    return END


async def end(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """End conversation from InlineKeyboardButton."""
    await update.callback_query.answer()

    text = "See you around!"
    await update.callback_query.edit_message_text(text=text)

    return END


# Second level conversation callbacks
async def select_level(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
    """Choose to add a parent or a child."""
    text = "You may add a parent or a child. Also you can show the gathered data or␣
↪go back."
    buttons = [
        [
            InlineKeyboardButton(text="Add parent", callback_data=str(PARENTS)),
            InlineKeyboardButton(text="Add child", callback_data=str(CHILDREN)),
        ],
        [
            InlineKeyboardButton(text="Show data", callback_data=str(SHOWING)),
            InlineKeyboardButton(text="Back", callback_data=str(END)),
        ],
    ]
    keyboard = InlineKeyboardMarkup(buttons)

    await update.callback_query.answer()
    await update.callback_query.edit_message_text(text=text, reply_markup=keyboard)

    return SELECTING_LEVEL


async def select_gender(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
    """Choose to add mother or father."""
    level = update.callback_query.data
    context.user_data[CURRENT_LEVEL] = level

    text = "Please choose, whom to add."

    male, female = _name_switcher(level)
```

(continues on next page)

(continued from previous page)

```
    buttons = [
        [
            InlineKeyboardButton(text=f"Add {male}", callback_data=str(MALE)),
            InlineKeyboardButton(text=f"Add {female}", callback_data=str(FEMALE)),
        ],
        [
            InlineKeyboardButton(text="Show data", callback_data=str(SHOWING)),
            InlineKeyboardButton(text="Back", callback_data=str(END)),
        ],
    ]
    keyboard = InlineKeyboardMarkup(buttons)

    await update.callback_query.answer()
    await update.callback_query.edit_message_text(text=text, reply_markup=keyboard)

    return SELECTING_GENDER


async def end_second_level(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Return to top level conversation."""
    context.user_data[START_OVER] = True
    await start(update, context)

    return END


# Third level callbacks
async def select_feature(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
    """Select a feature to update for the person."""
    buttons = [
        [
            InlineKeyboardButton(text="Name", callback_data=str(NAME)),
            InlineKeyboardButton(text="Age", callback_data=str(AGE)),
            InlineKeyboardButton(text="Done", callback_data=str(END)),
        ]
    ]
    keyboard = InlineKeyboardMarkup(buttons)

    # If we collect features for a new person, clear the cache and save the gender
    if not context.user_data.get(START_OVER):
        context.user_data[FEATURES] = {GENDER: update.callback_query.data}
        text = "Please select a feature to update."

        await update.callback_query.answer()
        await update.callback_query.edit_message_text(text=text, reply_
↪markup=keyboard)
    # But after we do that, we need to send a new message
    else:
        text = "Got it! Please select a feature to update."
        await update.message.reply_text(text=text, reply_markup=keyboard)

    context.user_data[START_OVER] = False
    return SELECTING_FEATURE


```

(continues on next page)

(continued from previous page)

```
async def ask_for_input(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
    """Prompt user to input data for selected feature."""
    context.user_data[CURRENT_FEATURE] = update.callback_query.data
    text = "Okay, tell me."

    await update.callback_query.answer()
    await update.callback_query.edit_message_text(text=text)

    return TYPING


async def save_input(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
    """Save input for feature and return to feature selection."""
    user_data = context.user_data
    user_data[FEATURES][user_data[CURRENT_FEATURE]] = update.message.text

    user_data[START_OVER] = True

    return await select_feature(update, context)


async def end_describing(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """End gathering of features and return to parent conversation."""
    user_data = context.user_data
    level = user_data[CURRENT_LEVEL]
    if not user_data.get(level):
        user_data[level] = []
    user_data[level].append(user_data[FEATURES])

    # Print upper level menu
    if level == SELF:
        user_data[START_OVER] = True
        await start(update, context)
    else:
        await select_level(update, context)

    return END


async def stop_nested(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
    """Completely end conversation from within nested conversation."""
    await update.message.reply_text("Okay, bye.")

    return STOPPING


def main() -> None:
    """Run the bot."""
    # Create the Application and pass it your bot's token.
    application = Application.builder().token("TOKEN").build()

    # Set up third level ConversationHandler (collecting features)
    description_conv = ConversationHandler(
        entry_points=[
            CallbackQueryHandler(
                select_feature, pattern="^" + str(MALE) + "$|^" + str(FEMALE) + "$"
```

(continues on next page)

(continued from previous page)

```
            )
        ],
        states={
            SELECTING_FEATURE: [
                CallbackQueryHandler(ask_for_input, pattern="^(?!" + str(END) + ").*$
↪")
            ],
            TYPING: [MessageHandler(filters.TEXT & ~filters.COMMAND, save_input)],
        },
        fallbacks=[
            CallbackQueryHandler(end_describing, pattern="^" + str(END) + "$"),
            CommandHandler("stop", stop_nested),
        ],
        map_to_parent={
            # Return to second level menu
            END: SELECTING_LEVEL,
            # End conversation altogether
            STOPPING: STOPPING,
        },
    )

    # Set up second level ConversationHandler (adding a person)
    add_member_conv = ConversationHandler(
        entry_points=[CallbackQueryHandler(select_level, pattern="^" + str(ADDING_
↪MEMBER) + "$")],
        states={
            SELECTING_LEVEL: [
                CallbackQueryHandler(select_gender, pattern=f"^{PARENTS}$|^{CHILDREN}$
↪")
            ],
            SELECTING_GENDER: [description_conv],
        },
        fallbacks=[
            CallbackQueryHandler(show_data, pattern="^" + str(SHOWING) + "$"),
            CallbackQueryHandler(end_second_level, pattern="^" + str(END) + "$"),
            CommandHandler("stop", stop_nested),
        ],
        map_to_parent={
            # After showing data return to top level menu
            SHOWING: SHOWING,
            # Return to top level menu
            END: SELECTING_ACTION,
            # End conversation altogether
            STOPPING: END,
        },
    )

    # Set up top level ConversationHandler (selecting action)
    # Because the states of the third level conversation map to the ones of the␣
↪second level
    # conversation, we need to make sure the top level conversation can also handle␣
↪them
    selection_handlers = [
        add_member_conv,
        CallbackQueryHandler(show_data, pattern="^" + str(SHOWING) + "$"),
        CallbackQueryHandler(adding_self, pattern="^" + str(ADDING_SELF) + "$"),
```

(continues on next page)

(continued from previous page)

```
        CallbackQueryHandler(end, pattern="^" + str(END) + "$"),
    ]
    conv_handler = ConversationHandler(
        entry_points=[CommandHandler("start", start)],
        states={
            SHOWING: [CallbackQueryHandler(start, pattern="^" + str(END) + "$")],
            SELECTING_ACTION: selection_handlers,
            SELECTING_LEVEL: selection_handlers,
            DESCRIBING_SELF: [description_conv],
            STOPPING: [CommandHandler("start", start)],
        },
        fallbacks=[CommandHandler("stop", stop)],
    )

    application.add_handler(conv_handler)

    # Run the bot until the user presses Ctrl-C
    application.run_polling()


if __name__ == "__main__":
    main()
```

State Diagram

passportbot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument, wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""
Simple Bot to print/download all incoming passport data

See https://telegram.org/blog/passport for info about what telegram passport is.

See https://github.com/python-telegram-bot/python-telegram-bot/wiki/Telegram-Passport
 for how to use Telegram Passport properly with python-telegram-bot.

"""
import logging
from pathlib import Path

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To␣
↪view the "
```

(continues on next page)

(continued from previous page)

```
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import Update
from telegram.ext import Application, ContextTypes, MessageHandler, filters

# Enable logging

logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)

logger = logging.getLogger(__name__)


async def msg(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Downloads and prints the received passport data."""
    # Retrieve passport data
    passport_data = update.message.passport_data
    # If our nonce doesn't match what we think, this Update did not originate from us
    # Ideally you would randomize the nonce on the server
    if passport_data.decrypted_credentials.nonce != "thisisatest":
        return

    # Print the decrypted credential data
    # For all elements
    # Print their decrypted data
    # Files will be downloaded to current directory
    for data in passport_data.decrypted_data:  # This is where the data gets decrypted
        if data.type == "phone_number":
            print("Phone: ", data.phone_number)
        elif data.type == "email":
            print("Email: ", data.email)
        if data.type in (
            "personal_details",
            "passport",
            "driver_license",
            "identity_card",
            "internal_passport",
            "address",
        ):
            print(data.type, data.data)
        if data.type in (
            "utility_bill",
            "bank_statement",
            "rental_agreement",
            "passport_registration",
            "temporary_registration",
        ):
            print(data.type, len(data.files), "files")
            for file in data.files:
                actual_file = await file.get_file()
                print(actual_file)
                await actual_file.download()
        if (
            data.type in ("passport", "driver_license", "identity_card", "internal_
```

(continues on next page)

(continued from previous page)

```
↪passport")
            and data.front_side
        ):
            front_file = await data.front_side.get_file()
            print(data.type, front_file)
            await front_file.download()
        if data.type in ("driver_license" and "identity_card") and data.reverse_side:
            reverse_file = await data.reverse_side.get_file()
            print(data.type, reverse_file)
            await reverse_file.download()
        if (
            data.type in ("passport", "driver_license", "identity_card", "internal_
↪passport")
            and data.selfie
        ):
            selfie_file = await data.selfie.get_file()
            print(data.type, selfie_file)
            await selfie_file.download()
        if data.translation and data.type in (
            "passport",
            "driver_license",
            "identity_card",
            "internal_passport",
            "utility_bill",
            "bank_statement",
            "rental_agreement",
            "passport_registration",
            "temporary_registration",
        ):
            print(data.type, len(data.translation), "translation")
            for file in data.translation:
                actual_file = await file.get_file()
                print(actual_file)
                await actual_file.download()


def main() -> None:
    """Start the bot."""
    # Create the Application and pass it your token and private key
    private_key = Path("private.key")
    application = (
        Application.builder().token("TOKEN").private_key(private_key.read_bytes()).
↪build()
    )

    # On messages that include passport data call msg
    application.add_handler(MessageHandler(filters.PASSPORT_DATA, msg))

    # Run the bot until the user presses Ctrl-C
    application.run_polling()


if __name__ == "__main__":
    main()
```

HTML Page

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Telegram passport test!</title>
    <meta charset="utf-8">
    <meta content="IE=edge" http-equiv="X-UA-Compatible">
    <meta content="width=device-width, initial-scale=1" name="viewport">
</head>
<body>
<h1>Telegram passport test</h1>

<div id="telegram_passport_auth"></div>
</body>

<!--- Needs file from https://github.com/TelegramMessenger/TGPassportJsSDK downloaded␣
↪--->
<script src="telegram-passport.js"></script>
<script>
    "use strict";

    Telegram.Passport.createAuthButton('telegram_passport_auth', {
        bot_id: 1234567890, // YOUR BOT ID
        scope: {
            data: [{
                type: 'id_document',
                selfie: true
            }, 'address_document', 'phone_number', 'email'], v: 1
        }, // WHAT DATA YOU WANT TO RECEIVE
        public_key: '-----BEGIN PUBLIC KEY-----\n', // YOUR PUBLIC KEY
        nonce: 'thisisatest', // YOUR BOT WILL RECEIVE THIS DATA WITH THE REQUEST
        callback_url: 'https://example.org' // TELEGRAM WILL SEND YOUR USER BACK TO␣
↪THIS URL
    });

</script>
</html>
```

paymentbot.py

```python
#!/usr/bin/env python
# pylint: disable=unused-argument, wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""Basic example for a bot that can receive payment from user."""

import logging

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]
```

(continues on next page)

(continued from previous page)

```

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import LabeledPrice, ShippingOption, Update
from telegram.ext import (
    Application,
    CommandHandler,
    ContextTypes,
    MessageHandler,
    PreCheckoutQueryHandler,
    ShippingQueryHandler,
    filters,
)

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)

PAYMENT_PROVIDER_TOKEN = "PAYMENT_PROVIDER_TOKEN"


async def start_callback(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Displays info on how to use the bot."""
    msg = (
        "Use /shipping to get an invoice for shipping-payment, or /noshipping for an "
        "invoice without shipping."
    )

    await update.message.reply_text(msg)


async def start_with_shipping_callback(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Sends an invoice with shipping-payment."""
    chat_id = update.message.chat_id
    title = "Payment Example"
    description = "Payment Example using python-telegram-bot"
    # select a payload just for you to recognize its the donation from your bot
    payload = "Custom-Payload"
    # In order to get a provider_token see https://core.telegram.org/bots/payments#getting-a-token
    currency = "USD"
    # price in dollars
    price = 1
    # price * 100 so as to include 2 decimal points
    # check https://core.telegram.org/bots/payments#supported-currencies for more details
    prices = [LabeledPrice("Test", price * 100)]

```

(continues on next page)

(continued from previous page)

```
    # optionally pass need_name=True, need_phone_number=True,
    # need_email=True, need_shipping_address=True, is_flexible=True
    await context.bot.send_invoice(
        chat_id,
        title,
        description,
        payload,
        PAYMENT_PROVIDER_TOKEN,
        currency,
        prices,
        need_name=True,
        need_phone_number=True,
        need_email=True,
        need_shipping_address=True,
        is_flexible=True,
    )


async def start_without_shipping_callback(
    update: Update, context: ContextTypes.DEFAULT_TYPE
) -> None:
    """Sends an invoice without shipping-payment."""
    chat_id = update.message.chat_id
    title = "Payment Example"
    description = "Payment Example using python-telegram-bot"
    # select a payload just for you to recognize its the donation from your bot
    payload = "Custom-Payload"
    # In order to get a provider_token see https://core.telegram.org/bots/payments
↪#getting-a-token
    currency = "USD"
    # price in dollars
    price = 1
    # price * 100 so as to include 2 decimal points
    prices = [LabeledPrice("Test", price * 100)]

    # optionally pass need_name=True, need_phone_number=True,
    # need_email=True, need_shipping_address=True, is_flexible=True
    await context.bot.send_invoice(
        chat_id, title, description, payload, PAYMENT_PROVIDER_TOKEN, currency, prices
    )


async def shipping_callback(update: Update, context: ContextTypes.DEFAULT_TYPE) ->␣
↪None:
    """Answers the ShippingQuery with ShippingOptions"""
    query = update.shipping_query
    # check the payload, is this from your bot?
    if query.invoice_payload != "Custom-Payload":
        # answer False pre_checkout_query
        await query.answer(ok=False, error_message="Something went wrong...")
        return

    # First option has a single LabeledPrice
    options = [ShippingOption("1", "Shipping Option A", [LabeledPrice("A", 100)])]
    # second option has an array of LabeledPrice objects
    price_list = [LabeledPrice("B1", 150), LabeledPrice("B2", 200)]
```

(continues on next page)

(continued from previous page)

```
    options.append(ShippingOption("2", "Shipping Option B", price_list))
    await query.answer(ok=True, shipping_options=options)


# after (optional) shipping, it's the pre-checkout
async def precheckout_callback(update: Update, context: ContextTypes.DEFAULT_TYPE) ->␣
↪None:
    """Answers the PreQecheckoutQuery"""
    query = update.pre_checkout_query
    # check the payload, is this from your bot?
    if query.invoice_payload != "Custom-Payload":
        # answer False pre_checkout_query
        await query.answer(ok=False, error_message="Something went wrong...")
    else:
        await query.answer(ok=True)


# finally, after contacting the payment provider...
async def successful_payment_callback(update: Update, context: ContextTypes.DEFAULT_
↪TYPE) -> None:
    """Confirms the successful payment."""
    # do something after successfully receiving payment?
    await update.message.reply_text("Thank you for your payment!")


def main() -> None:
    """Run the bot."""
    # Create the Application and pass it your bot's token.
    application = Application.builder().token("TOKEN").build()

    # simple start function
    application.add_handler(CommandHandler("start", start_callback))

    # Add command handler to start the payment invoice
    application.add_handler(CommandHandler("shipping", start_with_shipping_callback))
    application.add_handler(CommandHandler("noshipping", start_without_shipping_
↪callback))

    # Optional handler if your product requires shipping
    application.add_handler(ShippingQueryHandler(shipping_callback))

    # Pre-checkout handler to final check
    application.add_handler(PreCheckoutQueryHandler(precheckout_callback))

    # Success! Notify your user!
    application.add_handler(
        MessageHandler(filters.SUCCESSFUL_PAYMENT, successful_payment_callback)
    )

    # Run the bot until the user presses Ctrl-C
    application.run_polling()


if __name__ == "__main__":
    main()
```

persistentconversationbot.py

```python
#!/usr/bin/env python
# pylint: disable=unused-argument, wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""
First, a few callback functions are defined. Then, those functions are passed to
the Application and registered at their respective places.
Then, the bot is started and runs until we press Ctrl-C on the command line.

Usage:
Example of a bot-user conversation using ConversationHandler.
Send /start to initiate the conversation.
Press Ctrl-C on the command line or send a signal to the process to stop the
bot.
"""

import logging
from typing import Dict

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import ReplyKeyboardMarkup, ReplyKeyboardRemove, Update
from telegram.ext import (
    Application,
    CommandHandler,
    ContextTypes,
    ConversationHandler,
    MessageHandler,
    PicklePersistence,
    filters,
)

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)

CHOOSING, TYPING_REPLY, TYPING_CHOICE = range(3)

reply_keyboard = [
    ["Age", "Favourite colour"],
    ["Number of siblings", "Something else..."],
```

(continues on next page)

(continued from previous page)

```
    ["Done"],
]
markup = ReplyKeyboardMarkup(reply_keyboard, one_time_keyboard=True)


def facts_to_str(user_data: Dict[str, str]) -> str:
    """Helper function for formatting the gathered user info."""
    facts = [f"{key} - {value}" for key, value in user_data.items()]
    return "\n".join(facts).join(["\n", "\n"])


async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Start the conversation, display any stored data and ask user for input."""
    reply_text = "Hi! My name is Doctor Botter."
    if context.user_data:
        reply_text += (
            f" You already told me your {', '.join(context.user_data.keys())}. Why don't you "
            f"tell me something more about yourself? Or change anything I already know."
        )
    else:
        reply_text += (
            " I will hold a more complex conversation with you. Why don't you tell me "
            "something about yourself?"
        )
    await update.message.reply_text(reply_text, reply_markup=markup)

    return CHOOSING


async def regular_choice(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Ask the user for info about the selected predefined choice."""
    text = update.message.text.lower()
    context.user_data["choice"] = text
    if context.user_data.get(text):
        reply_text = (
            f"Your {text}? I already know the following about that: {context.user_data[text]}"
        )
    else:
        reply_text = f"Your {text}? Yes, I would love to hear about that!"
    await update.message.reply_text(reply_text)

    return TYPING_REPLY


async def custom_choice(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Ask the user for a description of a custom category."""
    await update.message.reply_text(
        'Alright, please send me the category first, for example "Most impressive skill"'
    )

    return TYPING_CHOICE
```

(continues on next page)

(continued from previous page)

```
async def received_information(update: Update, context: ContextTypes.DEFAULT_TYPE) ->␣
↪int:
    """Store info provided by user and ask for the next category."""
    text = update.message.text
    category = context.user_data["choice"]
    context.user_data[category] = text.lower()
    del context.user_data["choice"]

    await update.message.reply_text(
        "Neat! Just so you know, this is what you already told me:"
        f"{facts_to_str(context.user_data)}"
        "You can tell me more, or change your opinion on something.",
        reply_markup=markup,
    )

    return CHOOSING


async def show_data(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Display the gathered info."""
    await update.message.reply_text(
        f"This is what you already told me: {facts_to_str(context.user_data)}"
    )


async def done(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Display the gathered info and end the conversation."""
    if "choice" in context.user_data:
        del context.user_data["choice"]

    await update.message.reply_text(
        f"I learned these facts about you: {facts_to_str(context.user_data)}Until␣
↪next time!",
        reply_markup=ReplyKeyboardRemove(),
    )
    return ConversationHandler.END


def main() -> None:
    """Run the bot."""
    # Create the Application and pass it your bot's token.
    persistence = PicklePersistence(filepath="conversationbot")
    application = Application.builder().token("TOKEN").persistence(persistence).
↪build()

    # Add conversation handler with the states CHOOSING, TYPING_CHOICE and TYPING_
↪REPLY
    conv_handler = ConversationHandler(
        entry_points=[CommandHandler("start", start)],
        states={
            CHOOSING: [
                MessageHandler(
                    filters.Regex("^(Age|Favourite colour|Number of siblings)$"),␣
↪regular_choice
```

(continues on next page)

(continued from previous page)

```
                ),
                MessageHandler(filters.Regex("^Something else...$"), custom_choice),
            ],
            TYPING_CHOICE: [
                MessageHandler(
                    filters.TEXT & ~(filters.COMMAND | filters.Regex("^Done$")), regular_choice
                )
            ],
            TYPING_REPLY: [
                MessageHandler(
                    filters.TEXT & ~(filters.COMMAND | filters.Regex("^Done$")),
                    received_information,
                )
            ],
        },
        fallbacks=[MessageHandler(filters.Regex("^Done$"), done)],
        name="my_conversation",
        persistent=True,
    )

    application.add_handler(conv_handler)

    show_data_handler = CommandHandler("show_data", show_data)
    application.add_handler(show_data_handler)

    # Run the bot until the user presses Ctrl-C
    application.run_polling()


if __name__ == "__main__":
    main()
```

pollbot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument, wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""
Basic example for a bot that works with polls. Only 3 people are allowed to interact with each
poll/quiz the bot generates. The preview command generates a closed poll/quiz, exactly like the
one the user sends the bot
"""
import logging

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

```

(continues on next page)

(continued from previous page)

```
if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import (
    KeyboardButton,
    KeyboardButtonPollType,
    Poll,
    ReplyKeyboardMarkup,
    ReplyKeyboardRemove,
    Update,
)
from telegram.constants import ParseMode
from telegram.ext import (
    Application,
    CommandHandler,
    ContextTypes,
    MessageHandler,
    PollAnswerHandler,
    PollHandler,
    filters,
)

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)


async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Inform user about what this bot can do"""
    await update.message.reply_text(
        "Please select /poll to get a Poll, /quiz to get a Quiz or /preview"
        " to generate a preview for your poll"
    )


async def poll(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Sends a predefined poll"""
    questions = ["Good", "Really good", "Fantastic", "Great"]
    message = await context.bot.send_poll(
        update.effective_chat.id,
        "How are you?",
        questions,
        is_anonymous=False,
        allows_multiple_answers=True,
    )
    # Save some info about the poll the bot_data for later use in receive_poll_answer
    payload = {
        message.poll.id: {
            "questions": questions,
            "message_id": message.message_id,
```

(continues on next page)

(continued from previous page)

```
            "chat_id": update.effective_chat.id,
            "answers": 0,
        }
    }
    context.bot_data.update(payload)


async def receive_poll_answer(update: Update, context: ContextTypes.DEFAULT_TYPE) ->␣
↪None:
    """Summarize a users poll vote"""
    answer = update.poll_answer
    answered_poll = context.bot_data[answer.poll_id]
    try:
        questions = answered_poll["questions"]
    # this means this poll answer update is from an old poll, we can't do our␣
↪answering then
    except KeyError:
        return
    selected_options = answer.option_ids
    answer_string = ""
    for question_id in selected_options:
        if question_id != selected_options[-1]:
            answer_string += questions[question_id] + " and "
        else:
            answer_string += questions[question_id]
    await context.bot.send_message(
        answered_poll["chat_id"],
        f"{update.effective_user.mention_html()} feels {answer_string}!",
        parse_mode=ParseMode.HTML,
    )
    answered_poll["answers"] += 1
    # Close poll after three participants voted
    if answered_poll["answers"] == 3:
        await context.bot.stop_poll(answered_poll["chat_id"], answered_poll["message_
↪id"])


async def quiz(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Send a predefined poll"""
    questions = ["1", "2", "4", "20"]
    message = await update.effective_message.reply_poll(
        "How many eggs do you need for a cake?", questions, type=Poll.QUIZ, correct_
↪option_id=2
    )
    # Save some info about the poll the bot_data for later use in receive_quiz_answer
    payload = {
        message.poll.id: {"chat_id": update.effective_chat.id, "message_id": message.
↪message_id}
    }
    context.bot_data.update(payload)


async def receive_quiz_answer(update: Update, context: ContextTypes.DEFAULT_TYPE) ->␣
↪None:
    """Close quiz after three participants took it"""
    # the bot can receive closed poll updates we don't care about
```

(continues on next page)

(continued from previous page)

```
    if update.poll.is_closed:
        return
    if update.poll.total_voter_count == 3:
        try:
            quiz_data = context.bot_data[update.poll.id]
        # this means this poll answer update is from an old poll, we can't stop it then
        except KeyError:
            return
        await context.bot.stop_poll(quiz_data["chat_id"], quiz_data["message_id"])


async def preview(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Ask user to create a poll and display a preview of it"""
    # using this without a type lets the user chooses what he wants (quiz or poll)
    button = [[KeyboardButton("Press me!", request_poll=KeyboardButtonPollType())]]
    message = "Press the button to let the bot generate a preview for your poll"
    # using one_time_keyboard to hide the keyboard
    await update.effective_message.reply_text(
        message, reply_markup=ReplyKeyboardMarkup(button, one_time_keyboard=True)
    )


async def receive_poll(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """On receiving polls, reply to it by a closed poll copying the received poll"""
    actual_poll = update.effective_message.poll
    # Only need to set the question and options, since all other parameters don't matter for
    # a closed poll
    await update.effective_message.reply_poll(
        question=actual_poll.question,
        options=[o.text for o in actual_poll.options],
        # with is_closed true, the poll/quiz is immediately closed
        is_closed=True,
        reply_markup=ReplyKeyboardRemove(),
    )


async def help_handler(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Display a help message"""
    await update.message.reply_text("Use /quiz, /poll or /preview to test this bot.")


def main() -> None:
    """Run bot."""
    # Create the Application and pass it your bot's token.
    application = Application.builder().token("TOKEN").build()
    application.add_handler(CommandHandler("start", start))
    application.add_handler(CommandHandler("poll", poll))
    application.add_handler(CommandHandler("quiz", quiz))
    application.add_handler(CommandHandler("preview", preview))
    application.add_handler(CommandHandler("help", help_handler))
    application.add_handler(MessageHandler(filters.POLL, receive_poll))
    application.add_handler(PollAnswerHandler(receive_poll_answer))
    application.add_handler(PollHandler(receive_quiz_answer))

```

(continues on next page)

(continued from previous page)

```
    # Run the bot until the user presses Ctrl-C
    application.run_polling()


if __name__ == "__main__":
    main()
```

rawapibot.py

This example uses only the pure, "bare-metal" API wrapper.

```
#!/usr/bin/env python
# pylint: disable=wrong-import-position
"""Simple Bot to reply to Telegram messages.

This is built on the API wrapper, see echobot.py to see the same example built
on the telegram.ext bot framework.
This program is dedicated to the public domain under the CC0 license.
"""
import asyncio
import logging
from typing import NoReturn

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]  # type:␣
↪ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To␣
↪view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import Bot
from telegram.error import Forbidden, NetworkError

logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)


async def main() -> NoReturn:
    """Run the bot."""
    # Here we use the `async with` syntax to properly initialize and shutdown␣
↪resources.
    async with Bot("TOKEN") as bot:
        # get the first pending update_id, this is so we can skip over it in case
        # we get a "Forbidden" exception.
        try:
```

(continues on next page)

(continued from previous page)

```
            update_id = (await bot.get_updates())[0].update_id
        except IndexError:
            update_id = None

        logger.info("listening for new messages...")
        while True:
            try:
                update_id = await echo(bot, update_id)
            except NetworkError:
                await asyncio.sleep(1)
            except Forbidden:
                # The user has removed or blocked the bot.
                update_id += 1


async def echo(bot: Bot, update_id: int) -> int:
    """Echo the message the user sent."""
    # Request updates after the last update_id
    updates = await bot.get_updates(offset=update_id, timeout=10)
    for update in updates:
        next_update_id = update.update_id + 1

        # your bot can receive updates without messages
        # and not all messages contain text
        if update.message and update.message.text:
            # Reply to the message
            logger.info("Found message %s!", update.message.text)
            await update.message.reply_text(update.message.text)
        return next_update_id
    return update_id


if __name__ == "__main__":
    try:
        asyncio.run(main())
    except KeyboardInterrupt:  # Ignore exception when Ctrl-C is pressed
        pass
```

timerbot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument, wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""
Simple Bot to send timed Telegram messages.

This Bot uses the Application class to handle the bot and the JobQueue to send
timed messages.

First, a few handler functions are defined. Then, those functions are passed to
the Application and registered at their respective places.
Then, the bot is started and runs until we press Ctrl-C on the command line.

Usage:
```

(continues on next page)

(continued from previous page)

```
Basic Alarm Bot example, sends a message after a set time.
Press Ctrl-C on the command line or send a signal to the process to stop the
bot.
"""

import logging

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To␣
↪view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import Update
from telegram.ext import Application, CommandHandler, ContextTypes

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)


# Define a few command handlers. These usually take the two arguments update and
# context.
# Best practice would be to replace context with an underscore,
# since context is an unused local variable.
# This being an example and not having context present confusing beginners,
# we decided to have it present as context.
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Sends explanation on how to use the bot."""
    await update.message.reply_text("Hi! Use /set <seconds> to set a timer")


async def alarm(context: ContextTypes.DEFAULT_TYPE) -> None:
    """Send the alarm message."""
    job = context.job
    await context.bot.send_message(job.chat_id, text=f"Beep! {job.data} seconds are␣
↪over!")


def remove_job_if_exists(name: str, context: ContextTypes.DEFAULT_TYPE) -> bool:
    """Remove job with given name. Returns whether job was removed."""
    current_jobs = context.job_queue.get_jobs_by_name(name)
    if not current_jobs:
        return False
    for job in current_jobs:
        job.schedule_removal()
    return True
```

(continues on next page)

(continued from previous page)

```
70
71
72  async def set_timer(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
73      """Add a job to the queue."""
74      chat_id = update.effective_message.chat_id
75      try:
76          # args[0] should contain the time for the timer in seconds
77          due = float(context.args[0])
78          if due < 0:
79              await update.effective_message.reply_text("Sorry we can not go back to␣
    ↪future!")
80              return
81
82          job_removed = remove_job_if_exists(str(chat_id), context)
83          context.job_queue.run_once(alarm, due, chat_id=chat_id, name=str(chat_id),␣
    ↪data=due)
84
85          text = "Timer successfully set!"
86          if job_removed:
87              text += " Old one was removed."
88          await update.effective_message.reply_text(text)
89
90      except (IndexError, ValueError):
91          await update.effective_message.reply_text("Usage: /set <seconds>")
92
93
94  async def unset(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
95      """Remove the job if the user changed their mind."""
96      chat_id = update.message.chat_id
97      job_removed = remove_job_if_exists(str(chat_id), context)
98      text = "Timer successfully cancelled!" if job_removed else "You have no active␣
    ↪timer."
99      await update.message.reply_text(text)
100
101
102 def main() -> None:
103     """Run bot."""
104     # Create the Application and pass it your bot's token.
105     application = Application.builder().token("TOKEN").build()
106
107     # on different commands - answer in Telegram
108     application.add_handler(CommandHandler(["start", "help"], start))
109     application.add_handler(CommandHandler("set", set_timer))
110     application.add_handler(CommandHandler("unset", unset))
111
112     # Run the bot until the user presses Ctrl-C
113     application.run_polling()
114
115
116 if __name__ == "__main__":
117     main()
```

webappbot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument,wrong-import-position
# This program is dedicated to the public domain under the CC0 license.

"""
Simple example of a Telegram WebApp which displays a color picker.
The static website for this website is hosted by the PTB team for your convenience.
Currently only showcases starting the WebApp via a KeyboardButton, as all other methods would
require a bot token.
"""
import json
import logging

from telegram import __version__ as TG_VER

try:
    from telegram import __version_info__
except ImportError:
    __version_info__ = (0, 0, 0, 0, 0)  # type: ignore[assignment]

if __version_info__ < (20, 0, 0, "alpha", 1):
    raise RuntimeError(
        f"This example is not compatible with your current PTB version {TG_VER}. To view the "
        f"{TG_VER} version of this example, "
        f"visit https://docs.python-telegram-bot.org/en/v{TG_VER}/examples.html"
    )
from telegram import KeyboardButton, ReplyKeyboardMarkup, ReplyKeyboardRemove, Update, WebAppInfo
from telegram.ext import Application, CommandHandler, ContextTypes, MessageHandler, filters

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)


# Define a `/start` command handler.
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Send a message with a button that opens a the web app."""
    await update.message.reply_text(
        "Please press the button below to choose a color via the WebApp.",
        reply_markup=ReplyKeyboardMarkup.from_button(
            KeyboardButton(
                text="Open the color picker!",
                web_app=WebAppInfo(url="https://python-telegram-bot.org/static/webappbot"),
            )
        ),
    )


```

(continues on next page)

(continued from previous page)

```python
# Handle incoming WebAppData
async def web_app_data(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Print the received data and remove the button."""
    # Here we use `json.loads`, since the WebApp sends the data JSON serialized string
    # (see webappbot.html)
    data = json.loads(update.effective_message.web_app_data.data)
    await update.message.reply_html(
        text=f"You selected the color with the HEX value <code>{data['hex']}</code>. The "
        f"corresponding RGB value is <code>{tuple(data['rgb'].values())}</code>.",
        reply_markup=ReplyKeyboardRemove(),
    )


def main() -> None:
    """Start the bot."""
    # Create the Application and pass it your bot's token.
    application = Application.builder().token("TOKEN").build()

    application.add_handler(CommandHandler("start", start))
    application.add_handler(MessageHandler(filters.StatusUpdate.WEB_APP_DATA, web_app_data))

    # Run the bot until the user presses Ctrl-C
    application.run_polling()


if __name__ == "__main__":
    main()
```

HTML Page

```html
<!--
    Simple static Telegram WebApp. Does not verify the WebAppInitData, as a bot token would be needed for that.
-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>python-telegram-bot Example WebApp</title>
    <script src="https://telegram.org/js/telegram-web-app.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/@jaames/iro@5"></script>
</head>
<script type="text/javascript">
    const colorPicker = new iro.ColorPicker('#picker', {
        borderColor: "#ffffff",
        borderWidth: 1,
        width: Math.round(document.documentElement.clientWidth / 2),
    });
    colorPicker.on('color:change', function (color) {
        document.body.style.background = color.hexString;
    });

```

(continues on next page)

(continued from previous page)

```
    Telegram.WebApp.ready();
    Telegram.WebApp.MainButton.setText('Choose Color').show().onClick(function () {
        const data = JSON.stringify({hex: colorPicker.color.hexString, rgb:␣
↪colorPicker.color.rgb});
        Telegram.WebApp.sendData(data);
        Telegram.WebApp.close();
    });
</script>
<body style="background-color: #ffffff">
<div style="position: absolute; margin-top: 5vh; margin-left: 5vw; height: 90vh;␣
↪width: 90vw; border-radius: 5vh; background-color: var(--tg-theme-bg-color); box-
↪shadow: 0 0 2vw
 #000000;">
    <div id="picker"
         style="display: flex; justify-content: center; align-items: center; height:␣
↪100%; width: 100%"></div>
</div>
</body>
<script type="text/javascript">
    Telegram.WebApp.expand();
</script>
</html>
```

10.5 Changelog

10.5.1 Version 20.0a3

Released 2022-08-27

This is the technical changelog for version 20.0a3. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Full Support for API 6.2 (#3195)

New Features

- New Rate Limiting Mechanism (#3148)
- Make `chat/user_data` Available in Error Handler for Errors in Jobs (#3152)
- Add `Application.post_shutdown` (#3126)

Bug Fixes

- Fix `helpers.mention_markdown` for Markdown V1 and Improve Related Unit Tests (#3155)
- Add `api_kwargs` Parameter to `Bot.log_out` and Improve Related Unit Tests (#3147)
- Make `Bot.delete_my_commands` a Coroutine Function (#3136)
- Fix `ConversationHandler.check_update` not respecting `per_user` (#3128)

Minor Changes, Documentation Improvements and CI

- Add Python 3.11 to Test Suite & Adapt Enum Behaviour (#3168)
- Drop Manual Token Validation (#3167)
- Simplify Unit Tests for `Bot.send_chat_action` (#3151)
- Drop `pre-commit` Dependencies from `requirements-dev.txt` (#3120)
- Change Default Values for `concurrent_updates` and `connection_pool_size` (#3127)
- Documentation Improvements (#3139, #3153, #3135)
- Type Hinting Fixes (#3202)

Dependencies

- Bump `sphinx` from 5.0.2 to 5.1.1 (#3177)
- Update `pre-commit` Dependencies (#3085)
- Bump `pytest-asyncio` from 0.18.3 to 0.19.0 (#3158)
- Update `tornado` requirement from ~=6.1 to ~=6.2 (#3149)
- Bump `black` from 22.3.0 to 22.6.0 (#3132)
- Bump `actions/setup-python` from 3 to 4 (#3131)

10.5.2 Version 20.0a2

Released 2022-06-27

This is the technical changelog for version 20.0a2. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Full Support for API 6.1 (#3112)

New Features

- Add Additional Shortcut Methods to `Chat` (#3115)
- Mermaid-based Example State Diagrams (#3090)

Minor Changes, Documentation Improvements and CI

- Documentation Improvements (#3103, #3121, #3098)
- Stabilize CI (#3119)
- Bump `pyupgrade` from 2.32.1 to 2.34.0 (#3096)
- Bump `furo` from 2022.6.4 to 2022.6.4.1 (#3095)
- Bump `mypy` from 0.960 to 0.961 (#3093)

10.5.3 Version 20.0a1

Released 2022-06-09

This is the technical changelog for version 20.0a1. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes:

- Drop Support for `ujson` and instead `BaseRequest.parse_json_payload` (#3037, #3072)
- Drop `InputFile.is_image` (#3053)
- Drop Explicit Type conversions in `__init__` s (#3056)
- Handle List-Valued Attributes More Consistently (#3057)
- Split `{Command, Prefix}Handler` And Make Attributes Immutable (#3045)
- Align Behavior Of `JobQueue.run_daily` With `cron` (#3046)
- Make PTB Specific Keyword-Only Arguments for PTB Specific in Bot methods (#3035)
- Adjust Equality Comparisons to Fit Bot API 6.0 (#3033)
- Add Tuple Based Version Info (#3030)- Improve Type Annotations for `CallbackContext` and Move Default Type Alias to `ContextTypes.DEFAULT_TYPE` (#3017, #3023)
- Rename `Job.context` to `Job.data` (#3028)
- Rename `Handler` to `BaseHandler` (#3019)

New Features:

- Add `Application.post_init` (#3078)
- Add Arguments `chat/user_id` to `CallbackContext` And Example On Custom Webhook Setups (#3059)
- Add Convenience Property `Message.id` (#3077)
- Add Example for `WebApp` (#3052)
- Rename `telegram.bot_api_version` to `telegram.__bot_api_version__` (#3030)

Bug Fixes:

- Fix Non-Blocking Entry Point in `ConversationHandler` (#3068)
- Escape Backslashes in `escape_markdown` (#3055)

Dependencies:

- Update `httpx` requirement from ~=0.22.0 to ~=0.23.0 (#3069)
- Update `cachetools` requirement from ~=5.0.0 to ~=5.2.0 (#3058, #3080)

Minor Changes, Documentation Improvements and CI:

- Move Examples To Documentation (#3089)
- Documentation Improvements and Update Dependencies (#3010, #3007, #3012, #3067, #3081, #3082)
- Improve Some Unit Tests (#3026)
- Update Code Quality dependencies (#3070, #3032,`#2998`_, #2999)
- Don't Set Signal Handlers On Windows By Default (#3065)
- Split `{Command, Prefix}Handler` And Make Attributes Immutable (#3045)
- Apply `isort` and Update `pre-commit.ci` Configuration (#3049)
- Adjust `pre-commit` Settings for `isort` (#3043)
- Add Version Check to Examples (#3036)
- Use `Collection` Instead of `List` and `Tuple` (#3025)
- Remove Client-Side Parameter Validation (#3024)
- Don't Pass Default Values of Optional Parameters to Telegram (#2978)
- Stabilize `Application.run_*` on Python 3.7 (#3009)
- Ignore Code Style Commits in `git blame` (#3003)
- Adjust Tests to Changed API Behavior (#3002)

10.5.4 Version 20.0a0

Released 2022-05-06

This is the technical changelog for version 20.0a0. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes:

- Refactor Initialization of Persistence Classes (#2604)
- Drop Non-`CallbackContext` API (#2617)
- Remove `__dict__` from `__slots__` and drop Python 3.6 (#2619, #2636)
- Move and Rename `TelegramDecryptionError` to `telegram.error.PassportDecryptionError` (#2621)
- Make `BasePersistence` Methods Abstract (#2624)
- Remove `day_is_strict` argument of `JobQueue.run_monthly` (#2634 by iota-008)
- Move `Defaults` to `telegram.ext` (#2648)

- Remove Deprecated Functionality (#2644, #2740, #2745)
- Overhaul of Filters (#2759, #2922)
- Switch to `asyncio` and Refactor PTBs Architecture (#2731)
- Improve `Job.__getattr__` (#2832)
- Remove `telegram.ReplyMarkup` (#2870)
- Persistence of `Bots`: Refactor Automatic Replacement and Integration with `TelegramObject` (#2893)

New Features:

- Introduce Builder Pattern (#2646)
- Add `Filters.update.edited` (#2705 by PhilippFr)
- Introduce `Enums` for `telegram.constants` (#2708)
- Accept File Paths for `private_key` (#2724)
- Associate `Jobs` with `chat/user_id` (#2731)
- Convenience Functionality for `ChatInviteLinks` (#2782)
- Add `Dispatcher.add_handlers` (#2823)
- Improve Error Messages in `CommandHandler.__init__` (#2837)
- `Defaults.protect_content` (#2840)
- Add `Dispatcher.migrate_chat_data` (#2848 by DonalDuck004)
- Add Method `drop_chat/user_data` to `Dispatcher` and Persistence (#2852)
- Add methods `ChatPermissions.{all, no}_permissions` (#2948)
- Full Support for API 6.0 (#2956)
- Add Python 3.10 to Test Suite (#2968)

Bug Fixes & Minor Changes:

- Improve Type Hinting for `CallbackContext` (#2587 by revolter)
- Fix Signatures and Improve `test_official` (#2643)
- Refine `Dispatcher.dispatch_error` (#2660)
- Make `InlineQuery.answer` Raise `ValueError` (#2675)
- Improve Signature Inspection for Bot Methods (#2686)
- Introduce `TelegramObject.set/get_bot` (#2712 by zpavloudis)
- Improve Subscription of `TelegramObject` (#2719 by SimonDamberg)
- Use Enums for Dynamic Types & Rename Two Attributes in `ChatMember` (#2817)
- Return Plain Dicts from `BasePersistence.get_*_data` (#2873)
- Fix a Bug in `ChatMemberUpdated.difference` (#2947)
- Update Dependency Policy (#2958)

Internal Restructurings & Improvements:

- Add User Friendly Type Check For Init Of `{Inline, Reply}KeyboardMarkup` (#2657)
- Warnings Overhaul (#2662)
- Clear Up Import Policy (#2671)
- Mark Internal Modules As Private (#2687 by kencx)
- Handle Filepaths via the `pathlib` Module (#2688 by eldbud)
- Refactor MRO of `InputMedia*` and Some File-Like Classes (#2717 by eldbud)
- Update Exceptions for Immutable Attributes (#2749)
- Refactor Warnings in `ConversationHandler` (#2755, #2784)
- Use `__all__` Consistently (#2805)

CI, Code Quality & Test Suite Improvements:

- Add Custom `pytest` Marker to Ease Development (#2628)
- Pass Failing Jobs to Error Handlers (#2692)
- Update Notification Workflows (#2695)
- Use Error Messages for `pylint` Instead of Codes (#2700 by Piraty)
- Make Tests Agnostic of the CWD (#2727 by eldbud)
- Update Code Quality Dependencies (#2748)
- Improve Code Quality (#2783)
- Update `pre-commit` Settings & Improve a Test (#2796)
- Improve Code Quality & Test Suite (#2843)
- Fix failing animation tests (#2865)
- Update and Expand Tests & pre-commit Settings and Improve Code Quality (#2925)
- Extend Code Formatting With Black (#2972)
- Update Workflow Permissions (#2984)
- Adapt Tests to Changed `Bot.get_file` Behavior (#2995)

Documentation Improvements:

- Doc Fixes (#2597)
- Add Code Comment Guidelines to Contribution Guide (#2612)
- Add Cross-References to External Libraries & Other Documentation Improvements (#2693, #2691 by joesinghh, #2739 by eldbud)
- Use Furo Theme, Make Parameters Referenceable, Add Documentation Building to CI, Improve Links to Source Code & Other Improvements (#2856, #2798, #2854, #2841)
- Documentation Fixes & Improvements (#2822)
- Replace `git.io` Links (#2872 by murugu-21)
- Overhaul Readmes, Update RTD Startpage & Other Improvements (#2969)

10.5.5 Version 13.11

Released 2022-02-02

This is the technical changelog for version 13.11. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes:

- Full Support for Bot API 5.7 (#2881)

10.5.6 Version 13.10

Released 2022-01-03

This is the technical changelog for version 13.10. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes:

- Full Support for API 5.6 (#2835)

Minor Changes & Doc fixes:

- Update Copyright to 2022 (#2836)
- Update Documentation of `BotCommand` (#2820)

10.5.7 Version 13.9

Released 2021-12-11

This is the technical changelog for version 13.9. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes:

- Full Support for Api 5.5 (#2809)

Minor Changes

- Adjust Automated Locking of Inactive Issues (#2775)

10.5.8 Version 13.8.1

Released 2021-11-08

This is the technical changelog for version 13.8.1. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Doc fixes:

- Add `ChatJoinRequest(Handler)` to Docs (#2771)

10.5.9 Version 13.8

Released 2021-11-08

This is the technical changelog for version 13.8. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes:

- Full support for API 5.4 (#2767)

Minor changes, CI improvements, Doc fixes and Type hinting:

- Create Issue Template Forms (#2689)
- Fix `camelCase` Functions in `ExtBot` (#2659)
- Fix Empty Captions not Being Passed by `Bot.copy_message` (#2651)
- Fix Setting Thumbs When Uploading A Single File (#2583)
- Fix Bug in `BasePersistence.insert/replace_bot` for Objects with `__dict__` not in `__slots__` (#2603)

10.5.10 Version 13.7

Released 2021-07-01

This is the technical changelog for version 13.7. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes:

- Full support for Bot API 5.3 (#2572)

Bug Fixes:

- Fix Bug in `BasePersistence.insert/replace_bot` for Objects with `__dict__` in their slots (#2561)
- Remove Incorrect Warning About `Defaults` and `ExtBot` (#2553)

Minor changes, CI improvements, Doc fixes and Type hinting:

- Type Hinting Fixes (#2552)
- Doc Fixes (#2551)
- Improve Deprecation Warning for `__slots__` (#2574)
- Stabilize CI (#2575)
- Fix Coverage Configuration (#2571)
- Better Exception-Handling for `BasePersistence.replace/insert_bot` (#2564)
- Remove Deprecated `pass_args` from Deeplinking Example (#2550)

10.5.11 Version 13.6

Released 2021-06-06

New Features:

- Arbitrary `callback_data` (#1844)
- Add `ContextTypes` & `BasePersistence.refresh_user/chat/bot_data` (#2262)
- Add `Filters.attachment` (#2528)
- Add `pattern` Argument to `ChosenInlineResultHandler` (#2517)

Major Changes:

- Add `slots` (#2345)

Minor changes, CI improvements, Doc fixes and Type hinting:

- Doc Fixes (#2495, #2510)
- Add `max_connections` Parameter to `Updater.start_webhook` (#2547)
- Fix for `Promise.done_callback` (#2544)
- Improve Code Quality (#2536, #2454)
- Increase Test Coverage of `CallbackQueryHandler` (#2520)
- Stabilize CI (#2522, #2537, #2541)
- Fix `send_phone_number_to_provider` argument for `Bot.send_invoice` (#2527)
- Handle Classes as Input for `BasePersistence.replace/insert_bot` (#2523)
- Bump Tornado Version and Remove Workaround from #2067 (#2494)

10.5.12 Version 13.5

Released 2021-04-30

Major Changes:

- Full support of Bot API 5.2 (#2489).

  Note: The `start_parameter` argument of `Bot.send_invoice` and the corresponding shortcuts is now optional, so the order of parameters had to be changed. Make sure to update your method calls accordingly.

- Update `ChatActions`, Deprecating `ChatAction.RECORD_AUDIO` and `ChatAction.UPLOAD_AUDIO` (#2460)

New Features:

- Convenience Utilities & Example for Handling `ChatMemberUpdated` (#2490)
- `Filters.forwarded_from` (#2446)

Minor changes, CI improvements, Doc fixes and Type hinting:

- Improve Timeouts in `ConversationHandler` (#2417)
- Stabilize CI (#2480)
- Doc Fixes (#2437)
- Improve Type Hints of Data Filters (#2456)
- Add Two `UserWarnings` (#2464)
- Improve Code Quality (#2450)
- Update Fallback Test-Bots (#2451)
- Improve Examples (#2441, #2448)

10.5.13 Version 13.4.1

Released 2021-03-14

Hot fix release:

- Fixed a bug in `setup.py` (#2431)

10.5.14 Version 13.4

Released 2021-03-14

Major Changes:

- Full support of Bot API 5.1 (#2424)

Minor changes, CI improvements, doc fixes and type hinting:

- Improve `Updater.set_webhook` (#2419)
- Doc Fixes (#2404)
- Type Hinting Fixes (#2425)
- Update `pre-commit` Settings (#2415)
- Fix Logging for Vendored `urllib3` (#2427)
- Stabilize Tests (#2409)

10.5.15 Version 13.3

Released 2021-02-19

Major Changes:

- Make `cryptography` Dependency Optional & Refactor Some Tests (#2386, #2370)
- Deprecate `MessageQueue` (#2393)

Bug Fixes:

- Refactor `Defaults` Integration (#2363)
- Add Missing `telegram.SecureValue` to init and Docs (#2398)

Minor changes:

- Doc Fixes (#2359)

10.5.16 Version 13.2

Released 2021-02-02

Major Changes:

- Introduce `python-telegram-bot-raw` (#2324)
- Explicit Signatures for Shortcuts (#2240)

New Features:

- Add Missing Shortcuts to `Message` (#2330)
- Rich Comparison for `Bot` (#2320)
- Add `run_async` Parameter to `ConversationHandler` (#2292)
- Add New Shortcuts to `Chat` (#2291)

- Add New Constant `MAX_ANSWER_CALLBACK_QUERY_TEXT_LENGTH` (#2282)
- Allow Passing Custom Filename For All Media (#2249)
- Handle Bytes as File Input (#2233)

Bug Fixes:

- Fix Escaping in Nested Entities in `Message` Properties (#2312)
- Adjust Calling of `Dispatcher.update_persistence` (#2285)
- Add `quote` kwarg to `Message.reply_copy` (#2232)
- `ConversationHandler`: Docs & `edited_channel_post` behavior (#2339)

Minor changes, CI improvements, doc fixes and type hinting:

- Doc Fixes (#2253, #2225)
- Reduce Usage of `typing.Any` (#2321)
- Extend Deeplinking Example (#2335)
- Add pyupgrade to pre-commit Hooks (#2301)
- Add PR Template (#2299)
- Drop Nightly Tests & Update Badges (#2323)
- Update Copyright (#2289, #2287)
- Change Order of Class DocStrings (#2256)
- Add macOS to Test Matrix (#2266)
- Start Using Versioning Directives in Docs (#2252)
- Improve Annotations & Docs of Handlers (#2243)

10.5.17 Version 13.1

Released 2020-11-29

Major Changes:

- Full support of Bot API 5.0 (#2181, #2186, #2190, #2189, #2183, #2184, #2188, #2185, #2192, #2196, #2193, #2223, #2199, #2187, #2147, #2205)

New Features:

- Add `Defaults.run_async` (#2210)
- Improve and Expand `CallbackQuery` Shortcuts (#2172)
- Add XOR Filters and make `Filters.name` a Property (#2179)
- Add `Filters.document.file_extension` (#2169)
- Add `Filters.caption_regex` (#2163)
- Add `Filters.chat_type` (#2128)
- Handle Non-Binary File Input (#2202)

Bug Fixes:

- Improve Handling of Custom Objects in `BasePersistence.insert/replace_bot` (#2151)
- Fix bugs in `replace/insert_bot` (#2218)

Minor changes, CI improvements, doc fixes and type hinting:

- Improve Type hinting (#2204, #2118, #2167, #2136)

- Doc Fixes & Extensions (#2201, #2161)
- Use F-Strings Where Possible (#2222)
- Rename kwargs to _kwargs where possible (#2182)
- Comply with PEP561 (#2168)
- Improve Code Quality (#2131)
- Switch Code Formatting to Black (#2122, #2159, #2158)
- Update Wheel Settings (#2142)
- Update `timerbot.py` to `v13.0` (#2149)
- Overhaul Constants (#2137)
- Add Python 3.9 to Test Matrix (#2132)
- Switch Codecov to `GitHub` Action (#2127)
- Specify Required pytz Version (#2121)

10.5.18 Version 13.0

Released 2020-10-07

For a detailed guide on how to migrate from v12 to v13, see this wiki page.

Major Changes:

- Deprecate old-style callbacks, i.e. set `use_context=True` by default (#2050)
- Refactor Handling of Message VS Update Filters (#2032)
- Deprecate `Message.default_quote` (#1965)
- Refactor persistence of Bot instances (#1994)
- Refactor `JobQueue` (#1981)
- Refactor handling of kwargs in Bot methods (#1924)
- Refactor `Dispatcher.run_async`, deprecating the `@run_async` decorator (#2051)

New Features:

- Type Hinting (#1920)
- Automatic Pagination for `answer_inline_query` (#2072)
- `Defaults.tzinfo` (#2042)
- Extend rich comparison of objects (#1724)
- Add `Filters.via_bot` (#2009)
- Add missing shortcuts (#2043)
- Allow `DispatcherHandlerStop` in `ConversationHandler` (#2059)
- Make Errors picklable (#2106)

Minor changes, CI improvements, doc fixes or bug fixes:

- Fix Webhook not working on Windows with Python 3.8+ (#2067)
- Fix setting thumbs with `send_media_group` (#2093)
- Make `MessageHandler` filter for `Filters.update` first (#2085)
- Fix `PicklePersistence.flush()` with only `bot_data` (#2017)
- Add test for clean argument of `Updater.start_polling/webhook` (#2002)

- Doc fixes, refinements and additions (#2005, #2008, #2089, #2094, #2090)
- CI fixes (#2018, #2061)
- Refine `pollbot.py` example (#2047)
- Refine Filters in examples (#2027)
- Rename `echobot` examples (#2025)
- Use Lock-Bot to lock old threads (#2048, #2052, #2049, #2053)

10.5.19 Version 12.8

Released 2020-06-22

Major Changes:

- Remove Python 2 support (#1715)
- Bot API 4.9 support (#1980)
- IDs/Usernames of `Filters.user` and `Filters.chat` can now be updated (#1757)

Minor changes, CI improvements, doc fixes or bug fixes:

- Update contribution guide and stale bot (#1937)
- Remove `NullHandlers` (#1913)
- Improve and expand examples (#1943, #1995, #1983, #1997)
- Doc fixes (#1940, #1962)
- Add `User.send_poll()` shortcut (#1968)
- Ignore private attributes en `TelegramObject.to_dict()` (#1989)
- Stabilize CI (#2000)

10.5.20 Version 12.7

Released 2020-05-02

Major Changes:

- Bot API 4.8 support. **Note:** The `Dice` object now has a second positional argument `emoji`. This is relevant, if you instantiate `Dice` objects manually. (#1917)
- Added `tzinfo` argument to `helpers.from_timestamp`. It now returns an timezone aware object. This is relevant for `Message.{date,forward_date,edit_date}`, `Poll.close_date` and `ChatMember.until_date` (#1621)

New Features:

- New method `run_monthly` for the `JobQueue` (#1705)
- `Job.next_t` now gives the datetime of the jobs next execution (#1685)

Minor changes, CI improvements, doc fixes or bug fixes:

- Stabalize CI (#1919, #1931)
- Use ABCs `@abstractmethod` instead of raising `NotImplementedError` for `Handler`, `BasePersistence` and `BaseFilter` (#1905)
- Doc fixes (#1914, #1902, #1910)

10.5.21 Version 12.6.1

Released 2020-04-11

Bug fixes:

- Fix serialization of `reply_markup` in media messages (#1889)

10.5.22 Version 12.6

Released 2020-04-10

Major Changes:

- Bot API 4.7 support. **Note:** In `Bot.create_new_sticker_set` and `Bot.add_sticker_to_set`, the order of the parameters had be changed, as the `png_sticker` parameter is now optional. (#1858)

Minor changes, CI improvements or bug fixes:

- Add tests for `swtich_inline_query(_current_chat)` with empty string (#1635)
- Doc fixes (#1854, #1874, #1884)
- Update issue templates (#1880)
- Favor concrete types over "Iterable" (#1882)
- Pass last valid `CallbackContext` to `TIMEOUT` handlers of `ConversationHandler` (#1826)
- Tweak handling of persistence and update persistence after job calls (#1827)
- Use checkout@v2 for GitHub actions (#1887)

10.5.23 Version 12.5.1

Released 2020-03-30

Minor changes, doc fixes or bug fixes:

- Add missing docs for *PollHandler* and *PollAnswerHandler* (#1853)
- Fix wording in *Filters* docs (#1855)
- Reorder tests to make them more stable (#1835)
- Make *ConversationHandler* attributes immutable (#1756)
- Make *PrefixHandler* attributes *command* and *prefix* editable (#1636)
- Fix UTC as default *tzinfo* for *Job* (#1696)

10.5.24 Version 12.5

Released 2020-03-29

New Features:

- *Bot.link* gives the *t.me* link of the bot (#1770)

Major Changes:

- Bot API 4.5 and 4.6 support. (#1508, #1723)

Minor changes, CI improvements or bug fixes:

- Remove legacy CI files (#1783, #1791)
- Update pre-commit config file (#1787)

- Remove builtin names (#1792)
- CI improvements (#1808, #1848)
- Support Python 3.8 (#1614, #1824)
- Use stale bot for auto closing stale issues (#1820, #1829, #1840)
- Doc fixes (#1778, #1818)
- Fix typo in *edit_message_media* (#1779)
- In examples, answer CallbackQueries and use *edit_message_text* shortcut (#1721)
- Revert accidental change in vendored urllib3 (#1775)

10.5.25 Version 12.4.2

Released 2020-02-10

Bug Fixes

- Pass correct parse_mode to InlineResults if bot.defaults is None (#1763)
- Make sure PP can read files that dont have bot_data (#1760)

10.5.26 Version 12.4.1

Released 2020-02-08

This is a quick release for #1744 which was accidently left out of v12.4.0 though mentioned in the release notes.

10.5.27 Version 12.4.0

Released 2020-02-08

New features:

- Set default values for arguments appearing repeatedly. We also have a wiki page for the new defaults. (#1490)
- Store data in `CallbackContext.bot_data` to access it in every callback. Also persists. (#1325)
- `Filters.poll` allows only messages containing a poll (#1673)

Major changes:

- `Filters.text` now accepts messages that start with a slash, because `CommandHandler` checks for `MessageEntity.BOT_COMMAND` since v12. This might lead to your MessageHandlers receiving more updates than before (#1680).
- `Filters.command` new checks for `MessageEntity.BOT_COMMAND` instead of just a leading slash. Also by `Filters.command(False)` you can now filters for messages containing a command *anywhere* in the text (#1744).

Minor changes, CI improvements or bug fixes:

- Add `disptacher` argument to `Updater` to allow passing a customized `Dispatcher` (#1484)
- Add missing names for `Filters` (#1632)
- Documentation fixes (#1624, #1647, #1669, #1703, #1718, #1734, #1740, #1642, #1739, #1746)
- CI improvements (#1716, #1731, #1738, #1748, #1749, #1750, #1752)
- Fix spelling issue for `encode_conversations_to_json` (#1661)
- Remove double assignement of `Dispatcher.job_queue` (#1698)

- Expose dispatcher as property for `CallbackContext` (#1684)
- Fix `None` check in `JobQueue._put()` (#1707)
- Log datetimes correctly in `JobQueue` (#1714)
- Fix false `Message.link` creation for private groups (#1741)
- Add option `--with-upstream-urllib3` to *setup.py* to allow using non-vendored version (#1725)
- Fix persistence for nested `ConversationHandlers` (#1679)
- Improve handling of non-decodable server responses (#1623)
- Fix download for files without `file_path` (#1591)
- test_webhook_invalid_posts is now considered flaky and retried on failure (#1758)

10.5.28 Version 12.3.0

Released 2020-01-11

New features:

- *Filters.caption* allows only messages with caption (#1631).
- Filter for exact messages/captions with new capability of *Filters.text* and *Filters.caption*. Especially useful in combination with ReplyKeyboardMarkup. (#1631).

Major changes:

- Fix inconsistent handling of naive datetimes (#1506).

Minor changes, CI improvements or bug fixes:

- Documentation fixes (#1558, #1569, #1579, #1572, #1566, #1577, #1656).
- Add mutex protection on *ConversationHandler* (#1533).
- Add *MAX_PHOTOSIZE_UPLOAD* constant (#1560).
- Add args and kwargs to *Message.forward()* (#1574).
- Transfer to GitHub Actions CI (#1555, #1556, #1605, #1606, #1607, #1612, #1615, #1645).
- Fix deprecation warning with Py3.8 by vendored urllib3 (#1618).
- Simplify assignements for optional arguments (#1600)
- Allow private groups for *Message.link* (#1619).
- Fix wrong signature call for *ConversationHandler.TIMEOUT* handlers (#1653).

10.5.29 Version 12.2.0

Released 2019-10-14

New features:

- Nested ConversationHandlers (#1512).

Minor changes, CI improvments or bug fixes:

- Fix CI failures due to non-backward compat attrs depndency (#1540).
- travis.yaml: TEST_OFFICIAL removed from allowed_failures.
- Fix typos in examples (#1537).
- Fix Bot.to_dict to use proper first_name (#1525).
- Refactor `test_commandhandler.py` (#1408).

- Add Python 3.8 (RC version) to Travis testing matrix (#1543).
- test_bot.py: Add to_dict test (#1544).
- Flake config moved into setup.cfg (#1546).

10.5.30 Version 12.1.1

Released 2019-09-18

Hot fix release

Fixed regression in the vendored urllib3 (#1517).

10.5.31 Version 12.1.0

Released 2019-09-13

Major changes:

- Bot API 4.4 support (#1464, #1510)
- Add *get_file* method to *Animation* & *ChatPhoto*. Add, *get_small_file* & *get_big_file* methods to *ChatPhoto* (#1489)
- Tools for deep linking (#1049)

Minor changes and/or bug fixes:

- Documentation fixes (#1500, #1499)
- Improved examples (#1502)

10.5.32 Version 12.0.0

Released 2019-08-29

Well… This felt like decades. But here we are with a new release.

Expect minor releases soon (mainly complete Bot API 4.4 support)

Major and/or breaking changes:

- Context based callbacks
- Persistence
- PrefixHandler added (Handler overhaul)
- Deprecation of RegexHandler and edited_messages, channel_post, etc. arguments (Filter overhaul)
- Various ConversationHandler changes and fixes
- Bot API 4.1, 4.2, 4.3 support
- Python 3.4 is no longer supported
- Error Handler now handles all types of exceptions (#1485)
- Return UTC from from_timestamp() (#1485)

See the wiki page at https://github.com/python-telegram-bot/python-telegram-bot/wiki/Transition-guide-to-Version-12.0 for a detailed guide on how to migrate from version 11 to version 12.

Context based callbacks (#1100)

- Use of `pass_` in handlers is deprecated.
- Instead use `use_context=True` on `Updater` or `Dispatcher` and change callback from (bot, update, others…) to (update, context).
- This also applies to error handlers `Dispatcher.add_error_handler` and JobQueue jobs (change (bot, job) to (context) here).
- For users with custom handlers subclassing Handler, this is mostly backwards compatible, but to use the new context based callbacks you need to implement the new collect_additional_context method.
- Passing bot to `JobQueue.__init__` is deprecated. Use JobQueue.set_dispatcher with a dispatcher instead.
- Dispatcher makes sure to use a single *CallbackContext* for a entire update. This means that if an update is handled by multiple handlers (by using the group argument), you can add custom arguments to the *CallbackContext* in a lower group handler and use it in higher group handler. NOTE: Never use with @run_async, see docs for more info. (#1283)
- If you have custom handlers they will need to be updated to support the changes in this release.
- Update all examples to use context based callbacks.

Persistence (#1017)

- Added PicklePersistence and DictPersistence for adding persistence to your bots.
- BasePersistence can be subclassed for all your persistence needs.
- Add a new example that shows a persistent ConversationHandler bot

Handler overhaul (#1114)

- CommandHandler now only triggers on actual commands as defined by telegram servers (everything that the clients mark as a tabable link).
- PrefixHandler can be used if you need to trigger on prefixes (like all messages starting with a "/" (old CommandHandler behaviour) or even custom prefixes like "#" or "!").

Filter overhaul (#1221)

- RegexHandler is deprecated and should be replaced with a MessageHandler with a regex filter.
- Use update filters to filter update types instead of arguments (message_updates, channel_post_updates and edited_updates) on the handlers.
- Completely remove allow_edited argument - it has been deprecated for a while.
- data_filters now exist which allows filters that return data into the callback function. This is how the regex filter is implemented.
- All this means that it no longer possible to use a list of filters in a handler. Use bitwise operators instead!

ConversationHandler

- Remove `run_async_timeout` and `timed_out_behavior` arguments (#1344)
- Replace with `WAITING` constant and behavior from states (#1344)
- Only emit one warning for multiple CallbackQueryHandlers in a ConversationHandler (#1319)
- Use warnings.warn for ConversationHandler warnings (#1343)
- Fix unresolvable promises (#1270)

Bug fixes & improvements

- Handlers should be faster due to deduped logic.
- Avoid compiling compiled regex in regex filter. (#1314)
- Add missing `left_chat_member` to Message.MESSAGE_TYPES (#1336)
- Make custom timeouts actually work properly (#1330)
- Add convenience classmethods (from_button, from_row and from_column) to InlineKeyboardMarkup
- Small typo fix in setup.py (#1306)
- Add Conflict error (HTTP error code 409) (#1154)
- Change MAX_CAPTION_LENGTH to 1024 (#1262)
- Remove some unnecessary clauses (#1247, #1239)
- Allow filenames without dots in them when sending files (#1228)
- Fix uploading files with unicode filenames (#1214)
- Replace http.server with Tornado (#1191)
- Allow SOCKSConnection to parse username and password from URL (#1211)
- Fix for arguments in passport/data.py (#1213)
- Improve message entity parsing by adding text_mention (#1206)
- Documentation fixes (#1348, #1397, #1436)
- Merged filters short-circuit (#1350)
- Fix webhook listen with tornado (#1383)
- Call task_done() on update queue after update processing finished (#1428)
- Fix send_location() - latitude may be 0 (#1437)
- Make MessageEntity objects comparable (#1465)
- Add prefix to thread names (#1358)

Buf fixes since v12.0.0b1

- Fix setting bot on ShippingQuery (#1355)
- Fix _trigger_timeout() missing 1 required positional argument: 'job' (#1367)
- Add missing message.text check in PrefixHandler check_update (#1375)
- Make updates persist even on DispatcherHandlerStop (#1463)
- Dispatcher force updating persistence object's chat data attribute(#1462)

Internal improvements

- Finally fix our CI builds mostly (too many commits and PRs to list)
- Use multiple bots for CI to improve testing times significantly.
- Allow pypy to fail in CI.
- Remove the last CamelCase CheckUpdate methods from the handlers we missed earlier.
- test_official is now executed in a different job

10.5.33 Version 11.1.0

Released 2018-09-01

Fixes and updates for Telegram Passport: (#1198)

- Fix passport decryption failing at random times
- Added support for middle names.
- Added support for translations for documents
- Add errors for translations for documents
- Added support for requesting names in the language of the user's country of residence
- Replaced the payload parameter with the new parameter nonce
- Add hash to EncryptedPassportElement

10.5.34 Version 11.0.0

Released 2018-08-29

Fully support Bot API version 4.0! (also some bugfixes :))

Telegram Passport (#1174):

- **Add full support for telegram passport.**
  - New types: PassportData, PassportFile, EncryptedPassportElement, EncryptedCredentials, PassportElementError, PassportElementErrorDataField, PassportElementErrorFrontSide, PassportElementErrorReverseSide, PassportElementErrorSelfie, PassportElementErrorFile and PassportElementErrorFiles.
  - New bot method: set_passport_data_errors
  - New filter: Filters.passport_data
  - Field passport_data field on Message
  - PassportData can be easily decrypted.
  - PassportFiles are automatically decrypted if originating from decrypted PassportData.
- See new passportbot.py example for details on how to use, or go to our telegram passport wiki page for more info
- NOTE: Passport decryption requires new dependency *cryptography*.

Inputfile rework (#1184):

- Change how Inputfile is handled internally
- This allows support for specifying the thumbnails of photos and videos using the thumb= argument in the different send_ methods.
- Also allows Bot.send_media_group to actually finally send more than one media.

- Add thumb to Audio, Video and Videonote
- Add Bot.edit_message_media together with InputMediaAnimation, InputMediaAudio, and inputMediaDocument.

Other Bot API 4.0 changes:

- Add forusquare_type to Venue, InlineQueryResultVenue, InputVenueMessageContent, and Bot.send_venue. (#1170)
- Add vCard support by adding vcard field to Contact, InlineQueryResultContact, InputContactMessageContent, and Bot.send_contact. (#1166)
- **Support new message entities: CASHTAG and PHONE_NUMBER. (#1179)**
  - Cashtag seems to be things like *$USD* and *$GBP*, but it seems telegram doesn't currently send them to bots.
  - Phone number also seems to have limited support for now
- Add Bot.send_animation, add width, height, and duration to Animation, and add Filters.animation. (#1172)

Non Bot API 4.0 changes:

- Minor integer comparison fix (#1147)
- Fix Filters.regex failing on non-text message (#1158)
- Fix ProcessLookupError if process finishes before we kill it (#1126)
- Add t.me links for User, Chat and Message if available and update User.mention_* (#1092)
- Fix mention_markdown/html on py2 (#1112)

10.5.35 Version 10.1.0

Released 2018-05-02

Fixes changing previous behaviour:

- Add urllib3 fix for socks5h support (#1085)
- Fix send_sticker() timeout=20 (#1088)

Fixes:

- Add a caption_entity filter for filtering caption entities (#1068)
- Inputfile encode filenames (#1086)
- InputFile: Fix proper naming of file when reading from subprocess.PIPE (#1079)
- Remove pytest-catchlog from requirements (#1099)
- Documentation fixes (#1061, #1078, #1081, #1096)

10.5.36 Version 10.0.2

Released 2018-04-17

Important fix:

- Handle utf8 decoding errors (#1076)

New features:

- Added Filter.regex (#1028)
- Filters for Category and file types (#1046)
- Added video note filter (#1067)

Fixes:

- Fix in telegram.Message (#1042)
- Make chat_id a positional argument inside shortcut methods of Chat and User classes (#1050)
- Make Bot.full_name return a unicode object. (#1063)
- CommandHandler faster check (#1074)
- Correct documentation of Dispatcher.add_handler (#1071)
- Various small fixes to documentation.

10.5.37 Version 10.0.1

Released 2018-03-05

Fixes:

- Fix conversationhandler timeout (PR #1032)
- Add missing docs utils (PR #912)

10.5.38 Version 10.0.0

Released 2018-03-02

Non backward compatabile changes and changed defaults

- JobQueue: Remove deprecated prevent_autostart & put() (PR #1012)
- Bot, Updater: Remove deprecated network_delay (PR #1012)
- Remove deprecated Message.new_chat_member (PR #1012)
- Retry bootstrap phase indefinitely (by default) on network errors (PR #1018)

New Features

- Support v3.6 API (PR #1006)
- User.full_name convinience property (PR #949)
- Add *send_phone_number_to_provider* and *send_email_to_provider* arguments to send_invoice (PR #986)
- Bot: Add shortcut methods reply_{markdown,html} (PR #827)
- Bot: Add shortcut method reply_media_group (PR #994)
- Added utils.helpers.effective_message_type (PR #826)
- Bot.get_file now allows passing a file in addition to file_id (PR #963)
- Add .get_file() to Audio, Document, PhotoSize, Sticker, Video, VideoNote and Voice (PR #963)
- Add .send_*() methods to User and Chat (PR #963)
- Get jobs by name (PR #1011)
- Add Message caption html/markdown methods (PR #1013)
- File.download_as_bytearray - new method to get a d/led file as bytearray (PR #1019)
- File.download(): Now returns a meaningful return value (PR #1019)
- Added conversation timeout in ConversationHandler (PR #895)

Changes

- Store bot in PreCheckoutQuery (PR #953)

- Updater: Issue INFO log upon received signal (PR #951)
- JobQueue: Thread safety fixes (PR #977)
- WebhookHandler: Fix exception thrown during error handling (PR #985)
- Explicitly check update.effective_chat in ConversationHandler.check_update (PR #959)
- Updater: Better handling of timeouts during get_updates (PR #1007)
- Remove unnecessary to_dict() (PR #834)
- CommandHandler - ignore strings in entities and "/" followed by whitespace (PR #1020)
- Documentation & style fixes (PR #942, PR #956, PR #962, PR #980, PR #983)

10.5.39 Version 9.0.0

Released 2017-12-08

Breaking changes (possibly)

- Drop support for python 3.3 (PR #930)

New Features

- Support Bot API 3.5 (PR #920)

Changes

- Fix race condition in dispatcher start/stop (#887)
- Log error trace if there is no error handler registered (#694)
- Update examples with consistent string formatting (#870)
- Various changes and improvements to the docs.

10.5.40 Version 8.1.1

Released 2017-10-15

- Fix Commandhandler crashing on single character messages (PR #873).

10.5.41 Version 8.1.0

Released 2017-10-14

New features - Support Bot API 3.4 (PR #865).

Changes - MessageHandler & RegexHandler now consider channel_updates. - Fix command not recognized if it is directly followed by a newline (PR #869). - Removed Bot._message_wrapper (PR #822). - Unitests are now also running on AppVeyor (Windows VM). - Various unitest improvements. - Documentation fixes.

10.5.42 Version 8.0.0

Released 2017-09-01

New features

- Fully support Bot Api 3.3 (PR #806).
- DispatcherHandlerStop (see docs).
- Regression fix for text_html & text_markdown (PR #777).
- Added effective_attachment to message (PR #766).

Non backward compatible changes

- Removed Botan support from the library (PR #776).
- Fully support Bot Api 3.3 (PR #806).
- Remove de_json() (PR #789).

Changes

- Sane defaults for tcp socket options on linux (PR #754).
- Add RESTRICTED as constant to ChatMember (PR #761).
- Add rich comparison to CallbackQuery (PR #764).
- Fix get_game_high_scores (PR #771).
- Warn on small con_pool_size during custom initalization of Updater (PR #793).
- Catch exceptions in error handlerfor errors that happen during polling (PR #810).
- For testing we switched to pytest (PR #788).
- Lots of small improvements to our tests and documentation.

10.5.43 Version 7.0.1

Released 2017-07-28

- Fix TypeError exception in RegexHandler (PR #751).
- Small documentation fix (PR #749).

10.5.44 Version 7.0.0

Released 2017-07-25

- Fully support Bot API 3.2.
- New filters for handling messages from specific chat/user id (PR #677).
- Add the possibility to add objects as arguments to send_* methods (PR #742).
- Fixed download of URLs with UTF-8 chars in path (PR #688).
- Fixed URL parsing for `Message` text properties (PR #689).
- Fixed args dispatching in `MessageQueue`'s decorator (PR #705).
- Fixed regression preventing IPv6 only hosts from connnecting to Telegram servers (Issue #720).
- ConvesationHandler - check if a user exist before using it (PR #699).
- Removed deprecated `telegram.Emoji`.
- Removed deprecated `Botan` import from `utils` (`Botan` is still available through `contrib`).

- Removed deprecated `ReplyKeyboardHide`.
- Removed deprecated `edit_message` argument of `bot.set_game_score`.
- Internal restructure of files.
- Improved documentation.
- Improved unitests.

10.5.45 Pre-version 7.0

2017-06-18

Released 6.1.0

- Fully support Bot API 3.0
- Add more fine-grained filters for status updates
- Bug fixes and other improvements

2017-05-29

Released 6.0.3

- Faulty PyPI release

2017-05-29

Released 6.0.2

- Avoid confusion with user's `urllib3` by renaming vendored `urllib3` to `ptb_urllib3`

2017-05-19

Released 6.0.1

- Add support for `User.language_code`
- Fix `Message.text_html` and `Message.text_markdown` for messages with emoji

2017-05-19

Released 6.0.0

- Add support for Bot API 2.3.1
- Add support for `deleteMessage` API method
- New, simpler API for `JobQueue` - https://github.com/python-telegram-bot/python-telegram-bot/pull/484
- Download files into file-like objects - https://github.com/python-telegram-bot/python-telegram-bot/pull/459
- Use vendor `urllib3` to address issues with timeouts - The default timeout for messages is now 5 seconds. For sending media, the default timeout is now 20 seconds.
- String attributes that are not set are now `None` by default, instead of empty strings
- Add `text_markdown` and `text_html` properties to `Message` - https://github.com/python-telegram-bot/python-telegram-bot/pull/507
- Add support for Socks5 proxy - https://github.com/python-telegram-bot/python-telegram-bot/pull/518
- Add support for filters in `CommandHandler` - https://github.com/python-telegram-bot/python-telegram-bot/pull/536
- Add the ability to invert (not) filters - https://github.com/python-telegram-bot/python-telegram-bot/pull/552
- Add `Filters.group` and `Filters.private`

- Compatibility with GAE via `urllib3.contrib` package - https://github.com/python-telegram-bot/python-telegram-bot/pull/583
- Add equality rich comparision operators to telegram objects - https://github.com/python-telegram-bot/python-telegram-bot/pull/604
- Several bugfixes and other improvements
- Remove some deprecated code

2017-04-17

Released 5.3.1

- Hotfix release due to bug introduced by urllib3 version 1.21

2016-12-11

Released 5.3

- Implement API changes of November 21st (Bot API 2.3)
- `JobQueue` now supports `datetime.timedelta` in addition to seconds
- `JobQueue` now supports running jobs only on certain days
- New `Filters.reply` filter
- Bugfix for `Message.edit_reply_markup`
- Other bugfixes

2016-10-25

Released 5.2

- Implement API changes of October 3rd (games update)
- Add `Message.edit_*` methods
- Filters for the `MessageHandler` can now be combined using bitwise operators (`&` and `|`)
- Add a way to save user- and chat-related data temporarily
- Other bugfixes and improvements

2016-09-24

Released 5.1

- Drop Python 2.6 support
- Deprecate `telegram.Emoji`
- Use `ujson` if available
- Add instance methods to `Message`, `Chat`, `User`, `InlineQuery` and `CallbackQuery`
- RegEx filtering for `CallbackQueryHandler` and `InlineQueryHandler`
- New `MessageHandler` filters: `forwarded` and `entity`
- Add `Message.get_entity` to correctly handle UTF-16 codepoints and `MessageEntity` offsets
- Fix bug in `ConversationHandler` when first handler ends the conversation
- Allow multiple `Dispatcher` instances
- Add `ChatMigrated` Exception
- Properly split and handle arguments in `CommandHandler`

2016-07-15

Released 5.0

- Rework `JobQueue`

- Introduce `ConversationHandler`
- Introduce `telegram.constants` - https://github.com/python-telegram-bot/python-telegram-bot/pull/342

2016-07-12

Released 4.3.4

- Fix proxy support with `urllib3` when proxy requires auth

2016-07-08

Released 4.3.3

- Fix proxy support with `urllib3`

2016-07-04

Released 4.3.2

- Fix: Use `timeout` parameter in all API methods

2016-06-29

Released 4.3.1

- Update wrong requirement: `urllib3>=1.10`

2016-06-28

Released 4.3

- Use `urllib3.PoolManager` for connection re-use
- Rewrite `run_async` decorator to re-use threads
- New requirements: `urllib3` and `certifi`

2016-06-10

Released 4.2.1

- Fix `CallbackQuery.to_dict()` bug (thanks to @jlmadurga)
- Fix `editMessageText` exception when receiving a `CallbackQuery`

2016-05-28

Released 4.2

- Implement Bot API 2.1
- Move `botan` module to `telegram.contrib`
- New exception type: `BadRequest`

2016-05-22

Released 4.1.2

- Fix `MessageEntity` decoding with Bot API 2.1 changes

2016-05-16

Released 4.1.1

- Fix deprecation warning in `Dispatcher`

2016-05-15

Released 4.1

- Implement API changes from May 6, 2016
- Fix bug when `start_polling` with `clean=True`

- Methods now have snake_case equivalent, for example `telegram.Bot.send_message` is the same as `telegram.Bot.sendMessage`

2016-05-01

Released 4.0.3

- Add missing attribute `location` to `InlineQuery`

2016-04-29

Released 4.0.2

- Bugfixes
- `KeyboardReplyMarkup` now accepts `str` again

2016-04-27

Released 4.0.1

- Implement Bot API 2.0
- Almost complete recode of `Dispatcher`
- Please read the Transition Guide to 4.0
- **Changes from 4.0rc1**
  - The syntax of filters for `MessageHandler` (upper/lower cases)
  - Handler groups are now identified by `int` only, and ordered
- **Note:** v4.0 has been skipped due to a PyPI accident

2016-04-22

Released 4.0rc1

- Implement Bot API 2.0
- Almost complete recode of `Dispatcher`
- Please read the Transistion Guide to 4.0

2016-03-22

Released 3.4

- Move `Updater`, `Dispatcher` and `JobQueue` to new `telegram.ext` submodule (thanks to @rahiel)
- Add `disable_notification` parameter (thanks to @aidarbiktimirov)
- Fix bug where commands sent by Telegram Web would not be recognized (thanks to @shelomentsevd)
- Add option to skip old updates on bot startup
- Send files from `BufferedReader`

2016-02-28

Released 3.3

- Inline bots
- Send any file by URL
- Specialized exceptions: `Unauthorized`, `InvalidToken`, `NetworkError` and `TimedOut`
- Integration for botan.io (thanks to @ollmer)
- HTML Parsemode (thanks to @jlmadurga)
- Bugfixes and under-the-hood improvements

Very special thanks to Noam Meltzer (@tsnoam) for all of his work!

2016-01-09

Released 3.3b1

- Implement inline bots (beta)

2016-01-05

Released 3.2.0

- Introducing `JobQueue` (original author: @franciscod)
- Streamlining all exceptions to `TelegramError` (Special thanks to @tsnoam)
- Proper locking of `Updater` and `Dispatcher` `start` and `stop` methods
- Small bugfixes

2015-12-29

Released 3.1.2

- Fix custom path for file downloads
- Don't stop the dispatcher thread on uncaught errors in handlers

2015-12-21

Released 3.1.1

- Fix a bug where asynchronous handlers could not have additional arguments
- Add `groups` and `groupdict` as additional arguments for regex-based handlers

2015-12-16

Released 3.1.0

- The `chat`-field in `Message` is now of type `Chat`. (API update Oct 8 2015)
- `Message` now contains the optional fields `supergroup_chat_created`, `migrate_to_chat_id`, `migrate_from_chat_id` and `channel_chat_created`. (API update Nov 2015)

2015-12-08

Released 3.0.0

- Introducing the `Updater` and `Dispatcher` classes

2015-11-11

Released 2.9.2

- Error handling on request timeouts has been improved

2015-11-10

Released 2.9.1

- Add parameter `network_delay` to Bot.getUpdates for slow connections

2015-11-10

Released 2.9

- Emoji class now uses `bytes_to_native_str` from `future` 3rd party lib
- Make `user_from` optional to work with channels
- Raise exception if Telegram times out on long-polling

Special thanks to @jh0ker for all hard work

2015-10-08

Released 2.8.7

- Type as optional for `GroupChat` class

2015-10-08

Released 2.8.6

- Adds type to `User` and `GroupChat` classes (pre-release Telegram feature)

2015-09-24

Released 2.8.5

- Handles HTTP Bad Gateway (503) errors on request
- Fixes regression on `Audio` and `Document` for unicode fields

2015-09-20

Released 2.8.4

- `getFile` and `File.download` is now fully supported

2015-09-10

Released 2.8.3

- Moved `Bot._requestURL` to its own class (`telegram.utils.request`)
- Much better, such wow, Telegram Objects tests
- Add consistency for `str` properties on Telegram Objects
- Better design to test if `chat_id` is invalid
- Add ability to set custom filename on `Bot.sendDocument(..,filename='')`
- Fix Sticker as `InputFile`
- Send JSON requests over urlencoded post data
- Markdown support for `Bot.sendMessage(..., parse_mode=ParseMode.MARKDOWN)`
- Refactor of `TelegramError` class (no more handling `IOError` or `URLError`)

2015-09-05

Released 2.8.2

- Fix regression on Telegram ReplyMarkup
- Add certificate to `is_inputfile` method

2015-09-05

Released 2.8.1

- Fix regression on Telegram objects with thumb properties

2015-09-04

Released 2.8

- TelegramError when `chat_id` is empty for send* methods
- `setWebhook` now supports sending self-signed certificate
- Huge redesign of existing Telegram classes
- Added support for PyPy
- Added docstring for existing classes

2015-08-19

Released 2.7.1

- Fixed JSON serialization for `message`

2015-08-17

Released 2.7

- Added support for `Voice` object and `sendVoice` method
- Due backward compatibility performer or/and title will be required for `sendAudio`
- Fixed JSON serialization when forwarded message

2015-08-15

Released 2.6.1

- Fixed parsing image header issue on < Python 2.7.3

2015-08-14

Released 2.6.0

- Depreciation of `require_authentication` and `clearCredentials` methods
- Giving `AUTHORS` the proper credits for their contribution for this project
- `Message.date` and `Message.forward_date` are now `datetime` objects

2015-08-12

Released 2.5.3

- `telegram.Bot` now supports to be unpickled

2015-08-11

Released 2.5.2

- New changes from Telegram Bot API have been applied
- `telegram.Bot` now supports to be pickled
- Return empty `str` instead `None` when `message.text` is empty

2015-08-10

Released 2.5.1

- Moved from GPLv2 to LGPLv3

2015-08-09

Released 2.5

- Fixes logging calls in API

2015-08-08

Released 2.4

- Fixes `Emoji` class for Python 3
- `PEP8` improvements

2015-08-08

Released 2.3

- Fixes `ForceReply` class
- Remove `logging.basicConfig` from library

2015-07-25

Released 2.2

- Allows `debug=True` when initializing `telegram.Bot`

2015-07-20

Released 2.1

- Fix `to_dict` for `Document` and `Video`

2015-07-19

Released 2.0

- Fixes bugs
- Improves `__str__` over `to_json()`
- Creates abstract class `TelegramObject`

2015-07-15

Released 1.9

- Python 3 officially supported
- `PEP8` improvements

2015-07-12

Released 1.8

- Fixes crash when replying an unicode text message (special thanks to JRoot3D)

2015-07-11

Released 1.7

- Fixes crash when `username` is not defined on `chat` (special thanks to JRoot3D)

2015-07-10

Released 1.6

- Improvements for GAE support

2015-07-10

Released 1.5

- Fixes randomly unicode issues when using `InputFile`

2015-07-10

Released 1.4

- `requests` lib is no longer required
- Google App Engine (GAE) is supported

2015-07-10

Released 1.3

- Added support to `setWebhook` (special thanks to macrojames)

2015-07-09

Released 1.2

- `CustomKeyboard` classes now available
- Emojis available
- `PEP8` improvements

2015-07-08

Released 1.1

- PyPi package now available

2015-07-08

Released 1.0

- Initial checkin of python-telegram-bot

10.6 How To Contribute

Every open source project lives from the generous help by contributors that sacrifice their time and `python-telegram-bot` is no different. To make participation as pleasant as possible, this project adheres to the Code of Conduct by the Python Software Foundation.

10.6.1 Setting things up

1. Fork the `python-telegram-bot` repository to your GitHub account.
2. Clone your forked repository of `python-telegram-bot` to your computer:

```
$ git clone https://github.com/<your username>/python-telegram-bot --recursive
$ cd python-telegram-bot
```

3. Add a track to the original repository:

```
$ git remote add upstream https://github.com/python-telegram-bot/python-telegram-
↪bot
```

4. Install dependencies:

```
$ pip install -r requirements-all.txt
```

5. Install pre-commit hooks:

```
$ pre-commit install
```

10.6.2 Finding something to do

If you already know what you'd like to work on, you can skip this section.

If you have an idea for something to do, first check if it's already been filed on the issue tracker. If so, add a comment to the issue saying you'd like to work on it, and we'll help you get started! Otherwise, please file a new issue and assign yourself to it.

Another great way to start contributing is by writing tests. Tests are really important because they help prevent developers from accidentally breaking existing code, allowing them to build cool things faster. If you're interested in helping out, let the development team know by posting to the Telegram group, and we'll help you get started.

That being said, we want to mention that we are very hesitant about adding new requirements to our projects. If you intend to do this, please state this in an issue and get a verification from one of the maintainers.

10.6.3 Instructions for making a code change

The central development branch is `master`, which should be clean and ready for release at any time. In general, all changes should be done as feature branches based off of `master`.

If you want to do solely documentation changes, base them and PR to the branch `doc-fixes`. This branch also has its own RTD build.

Here's how to make a one-off code change.

1. **Choose a descriptive branch name.** It should be lowercase, hyphen-separated, and a noun describing the change (so, `fuzzy-rules`, but not `implement-fuzzy-rules`). Also, it shouldn't start with `hotfix` or `release`.
2. **Create a new branch with this name, starting from** `master`. In other words, run:

```
$ git fetch upstream
$ git checkout master
$ git merge upstream/master
$ git checkout -b your-branch-name
```

3. **Make a commit to your feature branch**. Each commit should be self-contained and have a descriptive commit message that helps other developers understand why the changes were made.
   - You can refer to relevant issues in the commit message by writing, e.g., "#105".
   - Your code should adhere to the PEP 8 Style Guide, with the exception that we have a maximum line length of 99.
   - Provide static typing with signature annotations. The documentation of MyPy will be a good start, the cheat sheet is here. We also have some custom type aliases in `telegram._utils.types`.
   - Document your code. This step is pretty important to us, so it has its own *section*.
   - For consistency, please conform to Google Python Style Guide and Google Python Style Docstrings.
   - The following exceptions to the above (Google's) style guides applies:
     - Documenting types of global variables and complex types of class members can be done using the Sphinx docstring convention.
   - In addition, PTB uses some formatting/styling and linting tools in the pre-commit setup. Some of those tools also have command line tools that can help to run these tools outside of the pre-commit step. If you'd like to leverage that, please have a look at the pre-commit config file for an overview of which tools (and which versions of them) are used. For example, we use Black for code formatting. Plugins for Black exist for some popular editors. You can use those instead of manually formatting everything.
   - Please ensure that the code you write is well-tested.
     - In addition to that, we provide the *dev* marker for pytest. If you write one or multiple tests and want to run only those, you can decorate them via *@pytest.mark.dev* and then run it with minimal overhead with *pytest ./path/to/test_file.py -m dev*.
   - Don't break backward compatibility.
   - Add yourself to the AUTHORS.rst file in an alphabetical fashion.
   - Before making a commit ensure that all automated tests still pass:

```
$ pytest -v
```

     To run `test_official` (particularly useful if you made API changes), run

```
$ export TEST_OFFICIAL=true
```

     prior to running the tests.
   - If you want run style & type checks before committing run

```
$ pre-commit run -a
```

- To actually make the commit (this will trigger tests style & type checks automatically):

```
$ git add your-file-changed.py
```

- Finally, push it to your GitHub fork, run:

```
$ git push origin your-branch-name
```

4. **When your feature is ready to merge, create a pull request.**
   - Go to your fork on GitHub, select your branch from the dropdown menu, and click "New pull request".
   - Add a descriptive comment explaining the purpose of the branch (e.g. "Add the new API feature to create inline bot queries."). This will tell the reviewer what the purpose of the branch is.
   - Click "Create pull request". An admin will assign a reviewer to your commit.
5. **Address review comments until all reviewers give LGTM ('looks good to me').**
   - When your reviewer has reviewed the code, you'll get a notification. You'll need to respond in two ways:
     - Make a new commit addressing the comments you agree with, and push it to the same branch. Ideally, the commit message would explain what the commit does (e.g. "Fix lint error"), but if there are lots of disparate review comments, it's fine to refer to the original commit message and add something like "(address review comments)".
     - In order to keep the commit history intact, please avoid squashing or amending history and then force-pushing to the PR. Reviewers often want to look at individual commits.
     - In addition, please reply to each comment. Each reply should be either "Done" or a response explaining why the corresponding suggestion wasn't implemented. All comments must be resolved before LGTM can be given.
   - Resolve any merge conflicts that arise. To resolve conflicts between 'your-branch-name' (in your fork) and 'master' (in the `python-telegram-bot` repository), run:

```
$ git checkout your-branch-name
$ git fetch upstream
$ git merge upstream/master
$ ...[fix the conflicts]...
$ ...[make sure the tests pass before committing]...
$ git commit -a
$ git push origin your-branch-name
```

   - At the end, the reviewer will merge the pull request.
6. **Tidy up!** Delete the feature branch from both your local clone and the GitHub repository:

```
$ git branch -D your-branch-name
$ git push origin --delete your-branch-name
```

7. **Celebrate.** Congratulations, you have contributed to `python-telegram-bot`!

10.6.4 Documenting

The documentation of this project is separated in two sections: User facing and dev facing.

User facing docs are hosted at RTD. They are the main way the users of our library are supposed to get information about the objects. They don't care about the internals, they just want to know what they have to pass to make it work, what it actually does. You can/should provide examples for non obvious cases (like the Filter module), and notes/warnings.

Dev facing, on the other side, is for the devs/maintainers of this project. These doc strings don't have a separate documentation site they generate, instead, they document the actual code.

User facing documentation

We use sphinx to generate static HTML docs. To build them, first make sure you have the required dependencies:

```
$ pip install -r docs/requirements-docs.txt
```

then run the following from the PTB root directory:

```
$ make -C docs html
```

or, if you don't have `make` available (e.g. on Windows):

```
$ sphinx-build docs/source docs/build/html
```

Once the process terminates, you can view the built documentation by opening `docs/build/html/index.html` with a browser.

- Add `.. versionadded:: version`, `.. versionchanged:: version` or `.. deprecated:: version` to the associated documentation of your changes, depending on what kind of change you made. This only applies if the change you made is visible to an end user. The directives should be added to class/method descriptions if their general behaviour changed and to the description of all arguments & attributes that changed.

Dev facing documentation

We adhere to the CSI standard. This documentation is not fully implemented in the project, yet, but new code changes should comply with the *CSI* standard. The idea behind this is to make it very easy for you/a random maintainer or even a totally foreign person to drop anywhere into the code and more or less immediately understand what a particular line does. This will make it easier for new to make relevant changes if said lines don't do what they are supposed to.

10.6.5 Style commandments

Assert comparison order

Assert statements should compare in **actual == expected** order. For example (assuming `test_call` is the thing being tested):

```
# GOOD
assert test_call() == 5

# BAD
assert 5 == test_call()
```

Properly calling callables

Methods, functions and classes can specify optional parameters (with default values) using Python's keyword arg syntax. When providing a value to such a callable we prefer that the call also uses keyword arg syntax. For example:

```
# GOOD
f(0, optional=True)

# BAD
f(0, True)
```

This gives us the flexibility to re-order arguments and more importantly to add new required arguments. It's also more explicit and easier to read.

Properly defining optional arguments

It's always good to not initialize optional arguments at class creation, instead use `**kwargs` to get them. It's well known Telegram API can change without notice, in that case if a new argument is added it won't break the API classes. For example:

```
# GOOD
def __init__(self, id, name, last_name=None, **kwargs):
    self.last_name = last_name


# BAD
def __init__(self, id, name, last_name=None):
    self.last_name = last_name
```

10.7 Contributor Covenant Code of Conduct

10.7.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

10.7.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Publication of any content supporting, justifying or otherwise affiliating with terror and/or hate towards others

- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

10.7.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

10.7.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

10.7.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at devs@python-telegram-bot.org. The project team will review and investigate all complaints, and will respond in a way that it deems appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

10.7.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4.

PYTHON MODULE INDEX

t

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z