python-telegram-bot Documentation

Release 22.8

Leandro Toledo

Jul 05, 2026

REFERENCE

1 Introduction **3**
- 1.1 Telegram API support . . . 3
- 1.2 Notable Features . . . 3

2 Installing **5**
- 2.1 Verifying Releases . . . 5
- 2.2 Dependencies & Their Versions . . . 5
  - 2.2.1 Optional Dependencies . . . 6

3 Working with PTB **7**
- 3.1 Quick Start . . . 7
- 3.2 Resources . . . 7
- 3.3 Getting help . . . 7
- 3.4 Concurrency . . . 7
- 3.5 Free threading . . . 8

4 Contributing **9**

5 Donating **11**

6 License **13**
- 6.1 telegram package . . . 13
  - 6.1.1 Version Constants . . . 13
  - 6.1.2 Classes in this package . . . 13
- 6.2 telegram.ext package . . . 803
  - 6.2.1 Application . . . 803
  - 6.2.2 ApplicationBuilder . . . 820
  - 6.2.3 ApplicationHandlerStop . . . 837
  - 6.2.4 BaseUpdateProcessor . . . 837
  - 6.2.5 CallbackContext . . . 839
  - 6.2.6 ContextTypes . . . 844
  - 6.2.7 Defaults . . . 845
  - 6.2.8 ExtBot . . . 849
  - 6.2.9 Job . . . 851
  - 6.2.10 JobQueue . . . 855
  - 6.2.11 SimpleUpdateProcessor . . . 862
  - 6.2.12 Updater . . . 862
  - 6.2.13 Handlers . . . 867
  - 6.2.14 Persistence . . . 937
  - 6.2.15 Arbitrary Callback Data . . . 952
  - 6.2.16 Rate Limiting . . . 956
- 6.3 Auxiliary modules . . . 960
  - 6.3.1 telegram.constants Module . . . 960
  - 6.3.2 telegram.error Module . . . 1036
  - 6.3.3 telegram.helpers Module . . . 1040

6.3.4 telegram.request Module . . . 1042
6.3.5 telegram.warnings Module . . . 1049
6.4 Examples . . . 1050
6.4.1 *echobot.py* . . . 1051
6.4.2 *timerbot.py* . . . 1051
6.4.3 *conversationbot.py* . . . 1051
6.4.4 *conversationbot2.py* . . . 1051
6.4.5 *nestedconversationbot.py* . . . 1051
6.4.6 *persistentconversationbot.py* . . . 1051
6.4.7 *inlinekeyboard.py* . . . 1051
6.4.8 *inlinekeyboard2.py* . . . 1051
6.4.9 *deeplinking.py* . . . 1051
6.4.10 *inlinebot.py* . . . 1051
6.4.11 *pollbot.py* . . . 1051
6.4.12 *passportbot.py* . . . 1052
6.4.13 *paymentbot.py* . . . 1052
6.4.14 *errorhandlerbot.py* . . . 1052
6.4.15 *chatmemberbot.py* . . . 1052
6.4.16 *webappbot.py* . . . 1052
6.4.17 *contexttypesbot.py* . . . 1052
6.4.18 *customwebhookbot.py* . . . 1052
6.4.19 *arbitrarycallbackdatabot.py* . . . 1052
6.4.20 Pure API . . . 1052
6.5 Stability Policy . . . 1120
6.5.1 What does this policy cover? . . . 1120
6.5.2 What doesn't this policy cover? . . . 1120
6.5.3 Versioning . . . 1121
6.6 Changelog . . . 1123
6.6.1 Unreleased . . . 1123
6.6.2 22.8 . . . 1123
6.6.3 22.7 . . . 1126
6.6.4 22.6 . . . 1128
6.6.5 22.5 . . . 1130
6.6.6 22.4 . . . 1131
6.6.7 22.3 . . . 1133
6.6.8 22.2 . . . 1134
6.6.9 22.1 . . . 1136
6.6.10 22.0 . . . 1138
6.6.11 Version 21.11.1 . . . 1139
6.6.12 Version 21.11 . . . 1139
6.6.13 Version 21.10 . . . 1140
6.6.14 Version 21.9 . . . 1140
6.6.15 Version 21.8 . . . 1141
6.6.16 Version 21.7 . . . 1141
6.6.17 Version 21.6 . . . 1142
6.6.18 Version 21.5 . . . 1143
6.6.19 Version 21.4 . . . 1144
6.6.20 Version 21.3 . . . 1145
6.6.21 Version 21.2 . . . 1145
6.6.22 Version 21.1.1 . . . 1146
6.6.23 Version 21.1 . . . 1146
6.6.24 Version 21.0.1 . . . 1147
6.6.25 Version 21.0 . . . 1147
6.6.26 Version 20.8 . . . 1148
6.6.27 Version 20.7 . . . 1149
6.6.28 Version 20.6 . . . 1150
6.6.29 Version 20.5 . . . 1151
6.6.30 Version 20.4 . . . 1152

6.6.31 Version 20.3 . . . 1153
6.6.32 Version 20.2 . . . 1154
6.6.33 Version 20.1 . . . 1155
6.6.34 Version 20.0 . . . 1156
6.6.35 Version 20.0b0 . . . 1156
6.6.36 Version 20.0a6 . . . 1157
6.6.37 Version 20.0a5 . . . 1157
6.6.38 Version 20.0a4 . . . 1159
6.6.39 Version 20.0a3 . . . 1159
6.6.40 Version 20.0a2 . . . 1160
6.6.41 Version 20.0a1 . . . 1160
6.6.42 Version 20.0a0 . . . 1161
6.6.43 Version 13.11 . . . 1164
6.6.44 Version 13.10 . . . 1164
6.6.45 Version 13.9 . . . 1164
6.6.46 Version 13.8.1 . . . 1164
6.6.47 Version 13.8 . . . 1164
6.6.48 Version 13.7 . . . 1165
6.6.49 Version 13.6 . . . 1165
6.6.50 Version 13.5 . . . 1166
6.6.51 Version 13.4.1 . . . 1166
6.6.52 Version 13.4 . . . 1166
6.6.53 Version 13.3 . . . 1167
6.6.54 Version 13.2 . . . 1167
6.6.55 Version 13.1 . . . 1168
6.6.56 Version 13.0 . . . 1169
6.6.57 Version 12.8 . . . 1169
6.6.58 Version 12.7 . . . 1170
6.6.59 Version 12.6.1 . . . 1170
6.6.60 Version 12.6 . . . 1170
6.6.61 Version 12.5.1 . . . 1171
6.6.62 Version 12.5 . . . 1171
6.6.63 Version 12.4.2 . . . 1171
6.6.64 Version 12.4.1 . . . 1171
6.6.65 Version 12.4.0 . . . 1172
6.6.66 Version 12.3.0 . . . 1172
6.6.67 Version 12.2.0 . . . 1173
6.6.68 Version 12.1.1 . . . 1173
6.6.69 Version 12.1.0 . . . 1173
6.6.70 Version 12.0.0 . . . 1174
6.6.71 Version 11.1.0 . . . 1176
6.6.72 Version 11.0.0 . . . 1176
6.6.73 Version 10.1.0 . . . 1177
6.6.74 Version 10.0.2 . . . 1177
6.6.75 Version 10.0.1 . . . 1178
6.6.76 Version 10.0.0 . . . 1178
6.6.77 Version 9.0.0 . . . 1179
6.6.78 Version 8.1.1 . . . 1179
6.6.79 Version 8.1.0 . . . 1179
6.6.80 Version 8.0.0 . . . 1179
6.6.81 Version 7.0.1 . . . 1180
6.6.82 Version 7.0.0 . . . 1180
6.6.83 Pre-version 7.0 . . . 1181
6.7 Contributor Covenant Code of Conduct . . . 1188
6.7.1 Our Pledge . . . 1188
6.7.2 Our Standards . . . 1189
6.7.3 Our Responsibilities . . . 1189
6.7.4 Scope . . . 1189

6.7.5 Enforcement . . . 1189
6.7.6 Attribution . . . 1189
6.8 How To Contribute . . . 1189
6.8.1 Setting things up . . . 1190
6.8.2 Finding something to do . . . 1190
6.8.3 Instructions for making a code change . . . 1190
6.8.4 Documenting . . . 1193
6.8.5 Style commandments . . . 1194
6.9 Testing in PTB . . . 1194
6.9.1 Running tests . . . 1194
6.9.2 Writing tests . . . 1195
6.9.3 Debugging tests . . . 1195
6.9.4 Bots used in tests . . . 1196

Python Module Index **1197**

Index **1199**

This is just here to get furo to display the right sidebar.

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.

CHAPTER

ONE

INTRODUCTION

This library provides a pure Python, asynchronous interface for the Telegram Bot API. It's compatible with Python versions **3.10+**.

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

After *installing* the library, be sure to check out the section on *working with PTB*.

1.1 Telegram API support

All types and methods of the Telegram Bot API **10.0** are natively supported by this library. In addition, Bot API functionality not yet natively included can still be used as described in our wiki.

1.2 Notable Features

- Fully asynchronous
- Convenient shortcut methods, e.g. Message.reply_text
- Fully annotated with static type hints
- Customizable and extendable interface
- Seamless integration with webhooks and polling
- *Comprehensive documentation and examples*

CHAPTER

TWO

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
$ pip install build
$ python -m build
```

You can also use your favored package manager (such as `uv`, `hatch`, `poetry`, etc.) instead of `pip`.

2.1 Verifying Releases

To enable you to verify that a release file that you downloaded was indeed provided by the `python-telegram-bot` team, we have taken the following measures.

Starting with v21.4, all releases are signed via sigstore. The corresponding signature files are uploaded to the GitHub releases page. To verify the signature, please install the sigstore Python client and follow the instructions for verifying signatures from GitHub Actions. As input for the `--repository` parameter, please use the value `python-telegram-bot/python-telegram-bot`.

Earlier releases are signed with a GPG key. The signatures are uploaded to both the GitHub releases page and the PyPI project and end with a suffix `.asc`. Please find the public keys here. The keys are named in the format `<first_version>-<last_version>.gpg`.

In addition, the GitHub release page also contains the sha1 hashes of the release files in the files with the suffix `.sha1`.

2.2 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. As these features are *optional*, the corresponding 3rd party dependencies are not installed by default. Instead, they are listed as optional dependencies. This allows to avoid unnecessary dependency conflicts for users who don't need the optional features.

The only required dependency is httpx >=0.27,<0.29 for `telegram.request.HTTPXRequest`, the default networking backend.

`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 (optional) 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.

2.2.1 Optional Dependencies

PTB can be installed with optional dependencies:

- `pip install "python-telegram-bot[passport]"` installs the cryptography>=39.0.1 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[http2]"` installs httpx[http2]. Use this, if you want to use HTTP/2.
- `pip install "python-telegram-bot[rate-limiter]"` installs aiolimiter~=1.1,<1.3. Use this, if you want to use `telegram.ext.AIORateLimiter`.
- `pip install "python-telegram-bot[webhooks]"` installs the tornado~=6.4 library. Use this, if you want to use `telegram.ext.Updater.start_webhook`/`telegram.ext.Application.run_webhook`.
- `pip install "python-telegram-bot[callback-data]"` installs the cachetools>=5.3.3,<6.3.0 library. Use this, if you want to use arbitrary callback_data.
- `pip install "python-telegram-bot[job-queue]"` installs the APScheduler>=3.10.4,<3.12.0 library. Use this, if you want to use the `telegram.ext.JobQueue`.

To install multiple optional dependencies, separate them by commas, e.g. `pip install "python-telegram-bot[socks,webhooks]"`.

Additionally, two shortcuts are provided:

- `pip install "python-telegram-bot[all]"` installs all optional dependencies.
- `pip install "python-telegram-bot[ext]"` installs all optional dependencies that are related to `telegram.ext`, i.e. `[rate-limiter, webhooks, callback-data, job-queue]`.

CHAPTER

THREE

WORKING WITH PTB

Once you have installed the library, you can begin working with it - so let's get started!

3.1 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.

3.2 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.

3.3 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.

3.4 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` currently does 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

3.5 Free threading

While `python-telegram-bot` is tested to work with Python 3.14 free threading, we do not guarantee that PTB is thread-safe for all use cases. Please see issue #4873 for more information.

CHAPTER

FOUR

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

FIVE

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

SIX

LICENSE

You may copy, distribute and modify the software provided that modifications are described and licensed for free under LGPL-3. Derivative 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.

6.1 telegram package

6.1.1 Version Constants

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

telegram.**__bot_api_version__ = '10.0'**

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__ = (10, 0)**

Shortcut for *telegram.constants.BOT_API_VERSION_INFO*.

Added in version 20.0.

Type
`typing.NamedTuple`

telegram.**__version__ = '22.8'**

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__ = (22, 8, 0, 'final', 0)**

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.

Added in version 20.0.

Type
`typing.NamedTuple`

6.1.2 Classes in this package

Bot

class telegram.**Bot**(*token*, *base_url='https://api.telegram.org/bot'*, *base_file_url='https://api.telegram.org/file/bot'*, *request=None*, *get_updates_request=None*, *private_key=None*, *private_key_password=None*, *local_mode=False*)

Bases: *telegram.TelegramObject*, contextlib.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 bot.shutdown()
```

See also

__aenter__() and *__aexit__()*.

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. The limitations to this argument are the same as the ones described in *do_api_request()*.
- 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. Trying to deepcopy a bot instance will raise TypeError.

Examples

Raw API Bot

See also

Your First Bot, Builder Pattern

Use In

telegram.ext.ApplicationBuilder.bot()

Available In

- *telegram.ext.Application.bot*

- *telegram.ext.CallbackContext.bot*
- *telegram.ext.Updater.bot*

Added 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`.
- Attempting to deepcopy a bot instance will now raise `TypeError`.
- 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.
- For uploading files, file paths are now always accepted. If *local_mode* is `False`, the file contents will be read in binary mode and uploaded. Otherwise, the file path will be passed in the file URI scheme.

Changed in version 20.5: Removed deprecated methods `set_sticker_set_thumb` and `setStickerSetThumb`. Use *set_sticker_set_thumbnail()* and *setStickerSetThumbnail()* instead.

Parameters

- ***token*** (`str`) – Bot's unique authentication token.
- ***base_url*** (`str` | Callable[[`str`], `str`], optional) – Telegram Bot API service URL. If the string contains `{token}`, it will be replaced with the bot's token. If a callable is passed, it will be called with the bot's token as the only argument and must return the base URL. Otherwise, the token will be appended to the string. Defaults to `"https://api.telegram.org/bot"`.

  Tip

  Customizing the base URL can be used to run a bot against Local Bot API Server or using Telegrams test environment.

  Example:

  ```
  "https://api.telegram.org/bot{token}/test"
  ```

  Changed in version 21.11: Supports callable input and string formatting.

- ***base_file_url*** (`str`, optional) – Telegram Bot API file URL. If the string contains `{token}`, it will be replaced with the bot's token. If a callable is passed, it will be called with the bot's token as the only argument and must return the base URL. Otherwise, the token will be appended to the string. Defaults to `"https://api.telegram.org/bot"`.

  Tip

  Customizing the base URL can be used to run a bot against Local Bot API Server or using Telegrams test environment.

Example:
"https://api.telegram.org/file/bot{token}/test"

Changed in version 21.11: Supports callable input and string formatting.

- ***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.
- ***local_mode*** (bool, optional) – Set to True, if the *base_url* is the URI of a Local Bot API Server that runs with the `--local` flag. Currently, the only effect of this is that files are uploaded using their local path in the file URI scheme. Defaults to False.

  Added in version 20.0..

send_animation(	Used for sending animations
send_audio()	Used for sending audio files
send_chat_actio	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_gift()	Used for sending a gift
send_invoice()	Used for sending an invoice
send_location()	Used for sending location
send_media_grou	Used for sending media grouped together
send_message()	Used for sending text messages
send_message_dr	Used for streaming partial text messages
send_paid_media	Used for sending paid media to channels
send_photo()	Used for sending photos
send_live_photo	Used for sending live 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
copy_messages()	Used for copying the contents of an multiple arbitrary messages
forward_message	Used for forwarding messages
forward_message	Used for forwarding multiple messages at once

answer_callback	Used for answering the callback query
answer_inline_q	Used for answering the inline query
answer_guest_qu	Used for replying to a received guest message
answer_pre_chec	Used for answering a pre checkout query
answer_shipping	Used for answering a shipping query
answer_web_app_	Used for answering a web app query
delete_message(	Used for deleting messages.
delete_messages	Used for deleting multiple messages as once.
edit_message_ca	Used for editing captions
edit_message_me	Used for editing the media on messages
edit_message_li	Used for editing the location in live location messages
edit_message_re	Used for editing the reply markup on messages
edit_message_te	Used for editing text messages
stop_poll()	Used for stopping the running poll
set_message_rea	Used for setting reactions on messages
delete_message_	Used for deleting reactions on messages
delete_all_mess	Used for deleting all reactions by a chat or user

approve_chat_jo	Used for approving a chat join request
decline_chat_jo	Used for declining a chat join request
approve_suggest	Used for approving a suggested post
decline_suggest	Used for declining a suggested post
ban_chat_member	Used for banning a member from the chat
unban_chat_memb	Used for unbanning a member from the chat
ban_chat_sender	Used for banning a channel in a channel or supergroup
unban_chat_send	Used for unbanning a channel in a channel or supergroup
restrict_chat_m	Used for restricting a chat member
promote_chat_me	Used for promoting a chat member
set_chat_admini	Used for assigning a custom admin title to an admin
set_chat_permis	Used for setting the permissions of a chat
export_chat_inv	Used for creating a new primary invite link for a chat
create_chat_inv	Used for creating an additional invite link for a chat
edit_chat_invit	Used for editing a non-primary invite link
revoke_chat_inv	Used for revoking an invite link created by the bot
set_chat_photo(	Used for setting a photo to a chat
delete_chat_pho	Used for deleting a chat photo
set_chat_title(	Used for setting a chat title
set_chat_descri	Used for setting the description of a chat
set_user_emoji_	Used for setting the users status emoji
pin_chat_messag	Used for pinning a message
unpin_chat_mess	Used for unpinning a message
unpin_all_chat_	Used for unpinning all pinned chat messages
get_user_person	Used for obtaining the personal chat messages of a user
get_user_profil	Used for obtaining user's profile audios
get_user_profil	Used for obtaining user's profile pictures
get_chat()	Used for getting information about a chat
get_chat_admini	Used for getting the list of admins in a chat
get_chat_member	Used for getting the number of members in a chat
get_chat_member	Used for getting a member of a chat
get_user_chat_b	Used for getting the list of boosts added to a chat
leave_chat()	Used for leaving a chat
set_chat_member	Used for setting the tag of a chat member

verify_chat()	Used for verifying a chat
verify_user()	Used for verifying a user
remove_chat_ver	Used for removing the verification from a chat
remove_user_ver	Used for removing the verification from a user

set_my_commands	Used for setting the list of commands
delete_my_comma	Used for deleting the list of commands
get_my_commands	Used for obtaining the list of commands
get_my_default_	Used for obtaining the default administrator rights for the bot
set_my_default_	Used for setting the default administrator rights for the bot
get_chat_menu_b	Used for obtaining the menu button of a private chat or the default menu button
set_chat_menu_b	Used for setting the menu button of a private chat or the default menu button
set_managed_bot	Used for changing the access settings of a managed bot
get_managed_bot	Used for obtaining the access settings of a managed bot
set_my_descript	Used for setting the description of the bot
get_my_descript	Used for obtaining the description of the bot
set_my_short_de	Used for setting the short description of the bot
get_my_short_de	Used for obtaining the short description of the bot
set_my_name()	Used for setting the name of the bot
get_my_name()	Used for obtaining the name of the bot
set_my_profile_	Used for setting the profile photo of the bot
remove_my_profi	Used for removing the profile photo of the bot

add_sticker_to_	Used for adding a sticker to a set
delete_sticker_	Used for deleting a sticker from a set
create_new_stic	Used for creating a new sticker set
delete_sticker_	Used for deleting a sticker set made by a bot
set_chat_sticke	Used for setting a sticker set of a chat
delete_chat_sti	Used for deleting the set sticker set of a chat
replace_sticker	Used for replacing a sticker in a set
set_sticker_pos	Used for moving a sticker's position in the set
set_sticker_set	Used for setting the title of a sticker set
set_sticker_emo	Used for setting the emoji list of a sticker
set_sticker_key	Used for setting the keywords of a sticker
set_sticker_mas	Used for setting the mask position of a mask sticker
set_sticker_set	Used for setting the thumbnail of a sticker set
set_custom_emoj	Used for setting the thumbnail of a custom emoji sticker set
get_sticker_set	Used for getting a sticker set
upload_sticker_	Used for uploading a sticker file
get_custom_emoj	Used for getting custom emoji files based on their IDs

get_game_high_s	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_inf	Used for getting current webhook status
set_webhook()	Used for setting a webhook to receive updates
delete_webhook(	Used for removing webhook integration

close_forum_top	Used for closing a forum topic
close_general_f	Used for closing the general forum topic
create_forum_to	Used to create a topic
delete_forum_to	Used for deleting a forum topic
edit_forum_topi	Used to edit a topic
edit_general_fo	Used to edit the general topic
get_forum_topic	Used to get custom emojis to use as topic icons
hide_general_fo	Used to hide the general topic
unhide_general_	Used to unhide the general topic
reopen_forum_to	Used to reopen a topic
reopen_general_	Used to reopen the general topic
unpin_all_forum	Used to unpin all messages in a forum topic
unpin_all_gener	Used to unpin all messages in the general forum topic

create_invoice_	Used to generate an HTTP link for an invoice
edit_user_star_	Used for editing a user's star subscription
get_my_star_bal	Used for obtaining the bot's Telegram Stars balance
get_star_transa	Used for obtaining the bot's Telegram Stars transactions
refund_star_pay	Used for refunding a payment in Telegram Stars
gift_premium_su	Used for gifting Telegram Premium to another user.

get_business_co	Used for getting information about the business account.
get_business_ac	Used for getting gifts owned by the business account.
get_business_ac	Used for getting the amount of Stars owned by the business account.
read_business_m	Used for marking a message as read.
delete_story()	Used for deleting business stories posted by the bot.
delete_business	Used for deleting business messages.
remove_business	Used for removing the business accounts profile photo
set_business_ac	Used for setting the business account name.
set_business_ac	Used for setting the business account username.
set_business_ac	Used for setting the business account bio.
set_business_ac	Used for setting the business account gift settings.
set_business_ac	Used for setting the business accounts profile photo
post_story()	Used for posting a story on behalf of business account.
repost_story()	Used for reposting an existing story on behalf of business account.
edit_story()	Used for editing business stories posted by the bot.
convert_gift_to	Used for converting owned requlars gifts to stars.
upgrade_gift()	Used for upgrading owned regular gifts to unique ones.
transfer_gift()	Used for transferring owned unique gifts to another user.
transfer_busine	Used for transfering Stars from the business account balance to the bot's balance.
send_checklist(	Used for sending a checklist on behalf of the business account.
edit_message_ch	Used for editing a checklist on behalf of the business account.

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_available_g	Used for getting information about gifts available for sending
get_chat_gifts(	Used for getting information about gifts owned and hosted by a chat
get_managed_bot	Used for getting the token of a managed bot
replace_managed	Used for replacing the token of a managed bot
get_me()	Used for getting basic information about the bot
get_user_gifts(	Used for getting information about gifts owned and hosted by a user
save_prepared_i	Used for storing a message to be sent by a user of a Mini App
save_prepared_k	Used for saving a keyboard button to be used in a Mini App

base_file_url	Telegram Bot API file URL
base_url	Telegram Bot API service URL
bot	The user instance of the bot as returned by get_me()
can_join_groups	Whether the bot can join groups
can_read_all_gr	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
local_mode	Whether the bot is running in local mode
username	The username of the bot, without leading @
link	The t.me link of the bot
private_key	Deserialized private key for decryption of telegram passport data
supports_inline	Whether the bot supports inline queries
token	Bot's unique authentication token

async __aenter__()

Asynchronous context manager which *initializes* the Bot.

Returns
The initialized Bot instance.

Raises
Exception – If an exception is raised during initialization, *shutdown()* is called in this case.

async __aexit__(*exc_type*, *exc_val*, *exc_tb*)

Asynchronous context manager which *shuts down* the Bot.

__deepcopy__(*memodict*)

Customizes how `copy.deepcopy()` processes objects of this type. Bots can not be deepcopied and this method will always raise an exception.

Added in version 20.0.

Raises
TypeError –

__eq__(*other*)

Defines equality condition for the *telegram.Bot* object. Two objects of this class are considered to be equal if their attributes *bot* are equal.

Returns
`True` if both attributes *bot* are equal. `False` otherwise.

__hash__()

See *telegram.TelegramObject.__hash__()*

__reduce__()

Customizes how `copy.deepcopy()` processes objects of this type. Bots can not be pickled and this method will always raise an exception.

Added in version 20.0.

Raises

`pickle.PicklingError` –

__repr__()

Give a string representation of the bot in the form `Bot[token=...]`.

As this class doesn't implement `object.__str__()`, the default implementation will be used, which is equivalent to *__repr__()*.

Returns

`str`

async addStickerToSet(*user_id*, *name*, *sticker*, ** (Keyword-only parameters separator (PEP 3102))*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *add_sticker_to_set()*

async add_sticker_to_set(*user_id*, *name*, *sticker*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to add a new sticker to a set created by the bot. The format of the added sticker must match the format of the other stickers in the set. Emoji sticker sets can have up to *200* stickers. Other sticker sets can have up to *120* stickers.

Changed in version 20.2: Since Bot API 6.6, the parameter *sticker* replace the parameters `png_sticker`, `tgs_sticker`, `webm_sticker`, `emojis`, and `mask_position`.

Changed in version 20.5: Removed deprecated parameters `png_sticker`, `tgs_sticker`, `webm_sticker`, `emojis`, and `mask_position`.

Parameters

- ***user_id*** (`int`) – User identifier of created sticker set owner.
- ***name*** (`str`) – Sticker set name.
- ***sticker*** (*telegram.InputSticker*) – An object with information about the added sticker. If exactly the same sticker had already been added to the set, then the set isn't changed.

  Added in version 20.2.

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*. By default, `20` seconds are used as write timeout.

  Changed in version 22.0: The default value changed 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. See *do_api_request()* for limitations.

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 answerGuestQuery(*guest_query_id*, *result*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *answer_guest_query()*

async answerInlineQuery(*inline_query_id*, *results*, *cache_time=None*, *is_personal=None*, *next_offset=None*, *button=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.

Shortcuts

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 | datetime.timedelta, optional) – The maximum amount of time in seconds that the result of the callback query may be cached client-side. Defaults to 0.

  Changed in version 21.11: datetime.timedelta objects are accepted in addition to plain int values.

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. See *do_api_request()* for limitations.

Returns

bool On success, True is returned.

Raises

telegram.error.TelegramError –

async answer_guest_query(*guest_query_id*, *result*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to reply to a received guest message.

Shortcuts

telegram.Message.answer_guest_query()

Added in version 22.8.

Parameters

- ***guest_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. See *do_api_request()* for limitations.

Returns
On success, a *telegram.SentGuestMessage* is returned.

Return type
telegram.SentGuestMessage

Raises
telegram.error.TelegramError –

async answer_inline_query(*inline_query_id*, *results*, *cache_time=None*, *is_personal=None*, *next_offset=None*, *button=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

Working with Files and Media

Shortcuts

telegram.InlineQuery.answer()

Changed in version 20.5: Removed deprecated arguments switch_pm_text and switch_pm_parameter.

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 | datetime.timedelta, optional) – The maximum amount of time in seconds that the result of the inline query may be cached on the server. Defaults to 300.

  Changed in version 21.11: datetime.timedelta objects are accepted in addition to plain int values.
- ***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.
- **button** (*telegram.InlineQueryResultsButton*, optional) – A button to be shown above the inline query results.

  Added in version 20.3.

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. See *do_api_request()* for limitations.

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.

Shortcuts

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. See *do_api_request()* for limitations.

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.

Shortcuts

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*** (Sequence[*telegram.ShippingOption*]), optional) – Required if *ok* is True. A sequence of available shipping options.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list.

- ***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. See *do_api_request()* for limitations.

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.

Added 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. See *do_api_request()* for limitations.

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 approveSuggestedPost(*chat_id*, *message_id*, *send_date=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *approve_suggested_post()*

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.

Shortcuts

- *telegram.Chat.approve_join_request()*
- *telegram.ChatFullInfo.approve_join_request()*
- *telegram.ChatJoinRequest.approve()*
- *telegram.User.approve_join_request()*

Added 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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

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

Use this method to approve a suggested post in a direct messages chat. The bot must have the *can_post_messages* administrator right in the corresponding channel chat.

Shortcuts

- *telegram.Chat.approve_suggested_post()*
- *telegram.ChatFullInfo.approve_suggested_post()*
- *telegram.Message.approve_suggested_post()*

Added in version 22.4.

Parameters

- ***chat_id*** (int) – Unique identifier of the target direct messages chat.
- ***message_id*** (int) – Identifier of a suggested post message to approve.
- ***send_date*** (int | datetime.datetime, optional) – Date when the post is expected to be published; omit if the date has already been specified when the suggested post was created. If specified, then the date must be not more than *2678400* seconds (30 days) in the future.

  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.

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. See *do_api_request()* for limitations.

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.

Shortcuts

- *telegram.Chat.ban_member()*
- *telegram.ChatFullInfo.ban_member()*

Added 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.

  Added 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. See *do_api_request()* for limitations.

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.

Shortcuts

- *telegram.Chat.ban_chat()*
- *telegram.Chat.ban_sender_chat()*
- *telegram.ChatFullInfo.ban_chat()*
- *telegram.ChatFullInfo.ban_sender_chat()*

Added 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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

property base_file_url

Telegram Bot API file URL, built from *Bot.base_file_url* and *Bot.token*.

Added in version 20.0.

Type

str

property base_url

Telegram Bot API service URL, built from *Bot.base_url* and *Bot.token*.

Added in version 20.0.

Type

str

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
True

Raises
telegram.error.TelegramError –

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

Alias for *close_forum_topic()*

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

Alias for *close_general_forum_topic()*

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

Use this method to close an open topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have *can_manage_topics* administrator rights, unless it is the creator of the topic.

Shortcuts

- *telegram.Chat.close_forum_topic()*
- *telegram.ChatFullInfo.close_forum_topic()*
- *telegram.Message.close_forum_topic()*

Added in version 20.0.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target supergroup (in the format `@supergroupusername`).
- ***message_thread_id*** (int) – Unique identifier for the target message thread of the forum topic.

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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

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

Use this method to close an open 'General' topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have *can_manage_topics* administrator rights.

Shortcuts

- *telegram.Chat.close_general_forum_topic()*
- *telegram.ChatFullInfo.close_general_forum_topic()*

Added in version 20.0.

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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

async convertGiftToStars(*business_connection_id*, *owned_gift_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *convert_gift_to_stars()*

async convert_gift_to_stars(*business_connection_id*, *owned_gift_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Converts a given regular gift to Telegram Stars. Requires the *can_convert_gifts_to_stars* business bot right.

Added in version 22.1.

Parameters

- ***business_connection_id*** (str) – Unique identifier of the business connection
- ***owned_gift_id*** (str) – Unique identifier of the regular gift that should be converted to Telegram Stars.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

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_markup=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *show_caption_above_media=None*, *allow_paid_broadcast=None*, *video_start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_effect_id=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *copy_message()*

async copyMessages(*chat_id*, *from_chat_id*, *message_ids*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *remove_caption=None*, *direct_messages_topic_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *copy_messages()*

async copy_message(*chat_id*, *from_chat_id*, *message_id*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *disable_notification=None*, *reply_markup=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *show_caption_above_media=None*, *allow_paid_broadcast=None*, *video_start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_effect_id=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=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, paid media messages, giveaway messages, giveaway winners 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.

Shortcuts

- *telegram.Chat.copy_message()*
- *telegram.Chat.send_copy()*
- *telegram.ChatFullInfo.copy_message()*
- *telegram.ChatFullInfo.send_copy()*
- *telegram.Message.copy()*
- *telegram.Message.reply_copy()*
- *telegram.User.copy_message()*
- *telegram.User.send_copy()*

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.
- ***video_start_timestamp*** (int, optional) – New start timestamp for the copied video in the message

  Added in version 21.11.
- ***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*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of `parse_mode`.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list.
- ***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.

  Added in version 13.10.
- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.
- ***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.
- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.

  Added in version 20.8.
- ***show_caption_above_media*** (bool, optional) – Pass `True`, if the caption must be shown above the message media.

  Added in version 21.3.
- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.
- ***suggested_post_parameters*** (*telegram.SuggestedPostParameters*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only. If the message is sent as a reply to another suggested post, then that suggested post is automatically declined.

  Added in version 22.4.

- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.

  Added in version 22.4.

- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; only available when copying to private chats

  Added in version 22.6.

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***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. See *do_api_request()* for limitations.

Returns

On success, the *telegram.MessageId* of the sent message is returned.

Return type

telegram.MessageId

Raises

telegram.error.TelegramError –

async copy_messages(*chat_id*, *from_chat_id*, *message_ids*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *remove_caption=None*, *direct_messages_topic_id=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. If some of the specified messages can't be found or copied, they are skipped. Service messages, paid media messages, giveaway messages, giveaway winners messages, and invoice messages can't be copied. A quiz poll can be copied only if the value of the field *telegram.Poll.correct_option_id* is known to the bot. The method is analogous to

the method *forward_messages()*, but the copied messages don't have a link to the original message. Album grouping is kept for copied messages.

Shortcuts

- *telegram.Chat.copy_messages()*
- *telegram.Chat.send_copies()*
- *telegram.ChatFullInfo.copy_messages()*
- *telegram.ChatFullInfo.send_copies()*
- *telegram.User.copy_messages()*
- *telegram.User.send_copies()*

Added in version 20.8.

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_ids*** (Sequence[int]) – A list of *1* - *100* identifiers of messages in the chat *from_chat_id* to copy. The identifiers must be specified in a strictly increasing order.
- ***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.
- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.
- ***remove_caption*** (bool, optional) – Pass `True` to copy the messages without their captions.
- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.

  Added in version 22.4.

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. See *do_api_request()* for limitations.

Returns

On success, a tuple of *MessageId* of the sent messages is returned.

Return type
tuple[*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 createChatSubscriptionInviteLink(*chat_id*, *subscription_period*, *subscription_price*, *name=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *create_chat_subscription_invite_link()*

async createForumTopic(*chat_id*, *name*, *icon_color=None*, *icon_custom_emoji_id=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *create_forum_topic()*

async createInvoiceLink(*title*, *description*, *payload*, *currency*, *prices*, *provider_token=None*, *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*, *subscription_period=None*, *business_connection_id=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*, *stickers*, *sticker_type=None*, *needs_repainting=None*, *, *read_timeout=None*, *write_timeout=None*, *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()*.

Note

When joining *public* groups via an invite link, Telegram clients may display the usual "Join" button, effectively ignoring the invite link. In particular, the parameter *creates_join_request* has no effect in this case. However, this behavior is undocument and may be subject to change. See this GitHub thread for some discussion.

Shortcuts

- *telegram.Chat.create_invite_link()*
- *telegram.ChatFullInfo.create_invite_link()*

Added 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.

  Added 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.

  Added 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. See *do_api_request()* for limitations.

Returns

telegram.ChatInviteLink

Raises

telegram.error.TelegramError –

async create_chat_subscription_invite_link(*chat_id*, *subscription_period*, *subscription_price*, *name=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to create a subscription invite link for a channel chat. The bot must have the *can_invite_users* administrator right. The link can be edited using the *edit_chat_subscription_invite_link()* or revoked using the *revoke_chat_invite_link()*.

Shortcuts

- *telegram.Chat.create_subscription_invite_link()*
- *telegram.ChatFullInfo.create_subscription_invite_link()*

Added in version 21.5.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***subscription_period*** (int | datetime.timedelta) – The number of seconds the subscription will be active for before the next payment. Currently, it must always be *2592000* (30 days).

  Changed in version 21.11: datetime.timedelta objects are accepted in addition to plain int values.
- ***subscription_price*** (int) – The number of Telegram Stars a user must pay initially and after each subsequent subscription period to be a member of the chat; *1*- *10000*.
- ***name*** (str, optional) – Invite link name; 0-*32* 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. See *do_api_request()* for limitations.

Returns

telegram.ChatInviteLink

Raises

telegram.error.TelegramError –

async create_forum_topic(*chat_id*, *name*, *icon_color=None*, *icon_custom_emoji_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to create a topic in a forum supergroup chat or a private chat with a user. The bot must be an administrator in the chat for this to work and must have *can_manage_topics* administrator rights.

Shortcuts

- *telegram.Chat.create_forum_topic()*
- *telegram.ChatFullInfo.create_forum_topic()*

Added in version 20.0.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target supergroup (in the format `@supergroupusername`).
- ***name*** (str) – New topic name, *1- 128* characters.
- ***icon_color*** (int, optional) – Color of the topic icon in RGB format. Currently, must be one of *telegram.constants.ForumIconColor.BLUE*, *telegram.constants.ForumIconColor.YELLOW*, *telegram.constants.ForumIconColor.PURPLE*, *telegram.constants.ForumIconColor.GREEN*, *telegram.constants.ForumIconColor.PINK*, or *telegram.constants.ForumIconColor.RED*.
- ***icon_custom_emoji_id*** (str, optional) – New unique identifier of the custom emoji shown as the topic icon. Use *get_forum_topic_icon_stickers()* to get all allowed custom emoji identifiers.

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. See *do_api_request()* for limitations.

Returns

telegram.ForumTopic

Raises

telegram.error.TelegramError –

async create_invoice_link(*title*, *description*, *payload*, *currency*, *prices*, *provider_token=None*, *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*, *subscription_period=None*, *business_connection_id=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.

Added in version 20.0.

Parameters

- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be sent. For payments in Telegram Stars only.

  Added in version 21.8.

- ***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 it for your internal processes.

- ***provider_token*** (str, optional) – Payments provider token, obtained via @BotFather. Pass an empty string for payments in Telegram Stars.

  Changed in version 21.11: Bot API 7.4 made this parameter is optional and this is now reflected in the function signature.

- ***currency*** (str) – Three-letter ISO 4217 currency code, see more on currencies. Pass XTR for payments in Telegram Stars.

- ***prices*** (Sequence[*telegram.LabeledPrice*) – Price breakdown, a sequence of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.). Must contain exactly one item for payments in Telegram Stars.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list.

- ***subscription_period*** (int | datetime.timedelta, optional) – The time the subscription will be active for before the next payment, either as number of seconds or as datetime.timedelta object. The currency must be set to "XTR" (Telegram Stars) if the parameter is used. Currently, it must always be *2592000* if specified. Any number of subscriptions can be active for a given bot at the same time, including multiple concurrent subscriptions from the same user. Subscription price must not exceed *10000* Telegram Stars.

  Added in version 21.8.

- ***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. Not supported for payments in Telegram Stars.

- ***suggested_tip_amounts*** (Sequence[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*.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list.

- ***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. Ignored for payments in Telegram Stars.

- ***need_phone_number*** (bool, optional) – Pass True, if you require the user's phone number to complete the order. Ignored for payments in Telegram Stars.

- ***need_email*** (bool, optional) – Pass True, if you require the user's email address to complete the order. Ignored for payments in Telegram Stars.

- ***need_shipping_address*** (bool, optional) – Pass True, if you require the user's shipping address to complete the order. Ignored for payments in Telegram Stars.

- ***send_phone_number_to_provider*** (bool, optional) – Pass True, if user's phone number should be sent to provider. Ignored for payments in Telegram Stars.
- ***send_email_to_provider*** (bool, optional) – Pass True, if user's email address should be sent to provider. Ignored for payments in Telegram Stars.
- ***is_flexible*** (bool, optional) – Pass True, if the final price depends on the shipping method. Ignored for payments in Telegram Stars.

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. See *do_api_request()* for limitations.

Returns

On success, the created invoice link is returned.

Return type

str

Raises

telegram.error.TelegramError –

async create_new_sticker_set(*user_id*, *name*, *title*, *stickers*, *sticker_type=None*, *needs_repainting=None*, ***, *read_timeout=None*, *write_timeout=None*, *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 thus created.

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

Changed in version 20.2: Since Bot API 6.6, the parameters *stickers* and *sticker_format* replace the parameters `png_sticker`, `tgs_sticker`,``webm_sticker``, `emojis`, and `mask_position`.

Changed in version 20.5: Removed the deprecated parameters mentioned above and adjusted the order of the parameters.

Removed in version 21.2: Removed the deprecated parameter `sticker_format`.

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.
- ***stickers*** (Sequence[*telegram.InputSticker*]) – A sequence of *1*- *50* initial stickers to be added to the sticker set.

  Added in version 20.2.

- ***sticker_type*** (str, optional) – Type of stickers in the set, pass *telegram.Sticker.REGULAR* or *telegram.Sticker.MASK*, or *telegram.Sticker.CUSTOM_EMOJI*. By default, a regular sticker set is created

  Added in version 20.0.

- ***needs_repainting*** (bool, optional) – Pass True if stickers in the sticker set must be repainted to the color of text when used in messages, the accent color if used as emoji status, white on chat photos, or another appropriate color based on context; for custom emoji sticker sets only.

  Added in version 20.2.

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*. By default, 20 seconds are used as write timeout.

  Changed in version 22.0: The default value changed 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. See *do_api_request()* for limitations.

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 declineSuggestedPost(*chat_id*, *message_id*, *comment=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *decline_suggested_post()*

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.

Shortcuts

- *telegram.Chat.decline_join_request()*
- *telegram.ChatFullInfo.decline_join_request()*

- *telegram.ChatJoinRequest.decline()*
- *telegram.User.decline_join_request()*

Added 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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

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

Use this method to decline a suggested post in a direct messages chat. The bot must have the *can_manage_direct_messages* administrator right in the corresponding channel chat.

Shortcuts

- *telegram.Chat.decline_suggested_post()*
- *telegram.ChatFullInfo.decline_suggested_post()*
- *telegram.Message.decline_suggested_post()*

Added in version 22.4.

Parameters

- ***chat_id*** (int) – Unique identifier of the target direct messages chat.
- ***message_id*** (int) – Identifier of a suggested post message to decline.
- ***comment*** (str, optional) – Comment for the creator of the suggested post. 0-*128* 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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

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

Alias for *delete_all_message_reactions()*

async deleteBusinessMessages(*business_connection_id*, *message_ids*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *delete_business_messages()*

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 deleteForumTopic(*chat_id*, *message_thread_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *delete_forum_topic()*

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 deleteMessageReaction(*chat_id*, *message_id*, *user_id=None*, *actor_chat_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *delete_message_reaction()*

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

Alias for *delete_messages()*

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

Alias for *delete_sticker_set()*

async deleteStory(*business_connection_id*, *story_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *delete_story()*

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_all_message_reactions(*chat_id*, *user_id=None*, *actor_chat_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to remove up to `10000` recent reactions in a group or a supergroup chat added by a given user or chat. The bot must have the *can_delete_messages* administrator right in the chat.

Shortcuts

- *telegram.Chat.delete_all_reactions()*
- *telegram.ChatFullInfo.delete_all_reactions()*
- *telegram.User.delete_all_reactions()*

Added in version 22.8.

Parameters

- ***chat_id*** (`int` | `str`) – Unique identifier for the target chat or username of the target supergroup (in the format `@supergroupusername`).
- ***user_id*** (`int`, optional) – Identifier of the user whose reactions will be removed, if the reactions were added by a user.
- ***actor_chat_id*** (`int`, optional) – Identifier of the chat whose reactions will be removed, if the reactions were added by a 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. See *do_api_request()* for limitations.

Returns
On success, `True` is returned.

Return type
`bool`

Raises
telegram.error.TelegramError –

async delete_business_messages(*business_connection_id*, *message_ids*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Delete messages on behalf of a business account. Requires the *can_delete_sent_messages* business bot right to delete messages sent by the bot itself, or the *can_delete_all_messages* business bot right to delete any message.

Shortcuts

telegram.Message.delete()

Added in version 22.1.

Parameters

- ***business_connection_id*** (`int` | `str`) – Unique identifier of the business connection on behalf of which to delete the messages
- ***message_ids*** (Sequence[`int`]) – A list of *1*- *100* identifiers of messages to delete. See *delete_message()* for limitations on which messages can be deleted.

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. See *do_api_request()* for limitations.

Returns
On success, `True` is returned.

Return type
`bool`

Raises
telegram.error.TelegramError –

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.

Shortcuts

- *telegram.Chat.delete_photo()*
- *telegram.ChatFullInfo.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. See *do_api_request()* for limitations.

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.ChatFullInfo.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. See *do_api_request()* for limitations.

Returns
On success, `True` is returned.

Return type
`bool`

Raises
telegram.error.TelegramError –

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

Use this method to delete a forum topic along with all its messages in a forum supergroup chat or a private chat with a user. In the case of a supergroup chat the bot must be an administrator in the chat for this to work and must have the *can_delete_messages* administrator rights.

Shortcuts

- *telegram.Chat.delete_forum_topic()*
- *telegram.ChatFullInfo.delete_forum_topic()*
- *telegram.Message.delete_forum_topic()*

Added in version 20.0.

Parameters

- ***chat_id*** (`int` | `str`) – Unique identifier for the target chat or username of the target supergroup (in the format `@supergroupusername`).
- ***message_thread_id*** (`int`) – Unique identifier for the target message thread of the forum topic.

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. See *do_api_request()* for limitations.

Returns
On success, `True` is returned.

Return type
`bool`

Raises
telegram.error.TelegramError –

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.

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

See also

telegram.CallbackQuery.delete_message() (calls *delete_message()* indirectly, via *telegram.Message.delete()*)

Shortcuts

- *telegram.Chat.delete_message()*
- *telegram.ChatFullInfo.delete_message()*
- *telegram.Message.delete()*
- *telegram.User.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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

async delete_message_reaction(*chat_id*, *message_id*, *user_id=None*, *actor_chat_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to remove a reaction from a message in a group or a supergroup chat. The bot must have the *can_delete_messages* administrator right in the chat.

Shortcuts

- *telegram.Chat.delete_reaction()*
- *telegram.ChatFullInfo.delete_reaction()*
- *telegram.Message.delete_reaction()*
- *telegram.User.delete_reaction()*

Added in version 22.8.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target supergroup (in the format `@supergroupusername`).
- ***message_id*** (int) – Identifier of the target message.
- ***user_id*** (int, optional) – Identifier of the user whose reaction will be removed, if the reaction were added by a user.
- ***actor_chat_id*** (int, optional) – Identifier of the chat whose reaction will be removed, if the reaction were added by a 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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

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

Use this method to delete multiple messages simultaneously. If some of the specified messages can't be found, they are skipped.

Shortcuts

- *telegram.Chat.delete_messages()*
- *telegram.ChatFullInfo.delete_messages()*
- *telegram.User.delete_messages()*

Added in version 20.8.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***message_ids*** (Sequence[int]) – A list of *1*- *100* identifiers of messages to delete. See *delete_message()* for limitations on which messages can be deleted.

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. See *do_api_request()* for limitations.

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.

Added in version 13.7.

See also

get_my_commands(), *set_my_commands()*

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. See *do_api_request()* for limitations.

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 | *telegram.Sticker*) – File identifier of the sticker or the sticker object.

Changed in version 21.10: Accepts also *telegram.Sticker* 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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

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

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

Added in version 20.2.

Parameters
name (str) – Sticker set name.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async delete_story(*business_connection_id*, *story_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Deletes a story previously posted by the bot on behalf of a managed business account. Requires the *can_manage_stories* business bot right.

Added in version 22.1.

Parameters

- ***business_connection_id*** (str) – Unique identifier of the business connection.
- ***story_id*** (int) – Unique identifier of the story 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. See *do_api_request()* for limitations.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async do_api_request(*endpoint*, *api_kwargs=None*, *return_type=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*)

Do a request to the Telegram API.

This method is here to make it easier to use new API methods that are not yet supported by this library.

Hint

Since PTB does not know which arguments are passed to this method, some caution is necessary in terms of PTBs utility functionalities. In particular

- passing objects of any class defined in the *telegram* module is supported
- when uploading files, a *telegram.InputFile* must be passed as the value for the corresponding argument. Passing a file path or file-like object will not work. File paths will work only in combination with *local_mode*.
- when uploading files, PTB can still correctly determine that a special write timeout value should be used instead of the default *telegram.request.HTTPXRequest.write_timeout*.
- insertion of default values specified via *telegram.ext.Defaults* will not work (only relevant for *telegram.ext.ExtBot*).
- The only exception is *telegram.ext.Defaults.tzinfo*, which will be correctly applied to datetime.datetime objects.

Added in version 20.8.

Parameters

- ***endpoint*** (str) – The API endpoint to use, e.g. `getMe` or `get_me`.
- ***api_kwargs*** (dict, optional) – The keyword arguments to pass to the API call. If not specified, no arguments are passed.
- ***return_type*** (*telegram.TelegramObject*, optional) – If specified, the result of the API call will be deserialized into an instance of this class or tuple of instances of this class. If not specified, the raw result of the API call 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*.

Returns

The result of the API call. If *return_type* is not specified, this is a dict or bool, otherwise an instance of *return_type* or a tuple of *return_type*.

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 editChatSubscriptionInviteLink(*chat_id*, *invite_link*, *name=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *edit_chat_subscription_invite_link()*

async editForumTopic(*chat_id*, *message_thread_id*, *name=None*, *icon_custom_emoji_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *edit_forum_topic()*

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

Alias for *edit_general_forum_topic()*

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

Alias for *edit_message_caption()*

async editMessageChecklist(*business_connection_id*, *chat_id*, *message_id*, *checklist*, *reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *edit_message_checklist()*

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*, *live_period=None*, *business_connection_id=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*, *business_connection_id=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*, *business_connection_id=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*, *reply_markup=None*, *entities=None*, *link_preview_options=None*, *business_connection_id=None*, ***, *disable_web_page_preview=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *edit_message_text()*

async editStory(*business_connection_id*, *story_id*, *content*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *areas=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *edit_story()*

async editUserStarSubscription(*user_id*, *telegram_payment_charge_id*, *is_canceled*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *edit_user_star_subscription()*

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.

Shortcuts

- *telegram.Chat.edit_invite_link()*
- *telegram.ChatFullInfo.edit_invite_link()*

Added 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.

  Added 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.

  Added 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. See *do_api_request()* for limitations.

Returns

telegram.ChatInviteLink

Raises

telegram.error.TelegramError –

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

Use this method to edit a subscription invite link created by the bot. The bot must have *telegram.ChatPermissions.can_invite_users* administrator right.

Shortcuts

- *telegram.Chat.edit_subscription_invite_link()*
- *telegram.ChatFullInfo.edit_subscription_invite_link()*

Added in version 21.5.

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.
- ***name*** (str, optional) – Invite link name; 0-*32* characters.

> **Tip**
>
> Omitting this argument removes the name of the invite link.

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. See *do_api_request()* for limitations.

Returns

telegram.ChatInviteLink

Raises

telegram.error.TelegramError –

async edit_forum_topic(*chat_id*, *message_thread_id*, *name=None*, *icon_custom_emoji_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to edit name and icon of a topic in a forum supergroup chat or a private chat with a user. In the case of a supergroup chat the bot must be an administrator in the chat for this to work and must have the *can_manage_topics* administrator rights, unless it is the creator of the topic.

> **Shortcuts**
>
> - *telegram.Chat.edit_forum_topic()*
> - *telegram.ChatFullInfo.edit_forum_topic()*
> - *telegram.Message.edit_forum_topic()*

Added in version 20.0.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target supergroup (in the format `@supergroupusername`).
- ***message_thread_id*** (int) – Unique identifier for the target message thread of the forum topic.
- ***name*** (str, optional) – New topic name, *1*- *128* characters. If not specified or empty, the current name of the topic will be kept.

- ***icon_custom_emoji_id*** (str, optional) – New unique identifier of the custom emoji shown as the topic icon. Use *get_forum_topic_icon_stickers()* to get all allowed custom emoji identifiers.Pass an empty string to remove the icon. If not specified, the current icon will be kept.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

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

Use this method to edit the name of the 'General' topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have the *can_manage_topics* administrator rights.

Shortcuts

- *telegram.Chat.edit_general_forum_topic()*
- *telegram.ChatFullInfo.edit_general_forum_topic()*

Added in version 20.0.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target supergroup (in the format `@supergroupusername`).
- ***name*** (str) – New topic name, *1*- *128* characters.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

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*, *show_caption_above_media=None*, *business_connection_id=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.
- Note that business messages that were not sent by the bot and do not contain an inline keyboard can only be edited within *48 hours* from the time they were sent.

Shortcuts

- *telegram.CallbackQuery.edit_message_caption()*
- *telegram.Message.edit_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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of parse_mode.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – An object for an inline keyboard.

- ***show_caption_above_media*** (bool, optional) – Pass True, if the caption must be shown above the message media.

  Added in version 21.3.

- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message to be edited was sent

  Added in version 21.4.

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. See *do_api_request()* for limitations.

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_checklist(*business_connection_id*, *chat_id*, *message_id*, *checklist*, *reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to edit a checklist on behalf of a connected business account.

Shortcuts

telegram.Message.edit_checklist()

Added in version 22.3.

Parameters

- ***business_connection_id*** (str) – Unique identifier of the business connection on behalf of which the message will be sent.
- ***chat_id*** (int) – Unique identifier for the target chat or username of the target bot in the format `@username`.
- ***message_id*** (int) – Unique identifier for the target message.
- ***checklist*** (*telegram.InputChecklist*) – The new checklist.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – An object for the new inline keyboard for the 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. See *do_api_request()* for limitations.

Returns
On success, the sent Message 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*, *live_period=None*, *business_connection_id=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*.

Shortcuts

- *telegram.CallbackQuery.edit_message_live_location()*
- *telegram.Message.edit_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 *100000* if specified.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – An object for a new inline keyboard.
- ***live_period*** (int | datetime.timedelta, optional) – New period in seconds during which the location can be updated, starting from the message send date. If *2147483647* is specified, then the location can be updated forever. Otherwise, the new value must not exceed the current `live_period` by more than a day, and the live location expiration date must remain within the next 90 days. If not specified, then `live_period` remains unchanged

  Added in version 21.2..

  Changed in version 21.11: datetime.timedelta objects are accepted in addition to plain int values.
- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message to be edited was sent

  Added in version 21.4.

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. See *do_api_request()* for limitations.

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

Use this method to edit animation, audio, document, live photo, photo, or video messages, or to add media to text 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, live 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.
- Note that business messages that were not sent by the bot and do not contain an inline keyboard can only be edited within *48 hours* from the time they were sent.

See also

Working with Files and Media

Shortcuts

- *telegram.CallbackQuery.edit_message_media()*
- *telegram.Message.edit_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.
- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message to be edited was sent

  Added in version 21.4.

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. See *do_api_request()* for limitations.

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*, *business_connection_id=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.
- Note that business messages that were not sent by the bot and do not contain an inline keyboard can only be edited within *48 hours* from the time they were sent.

Shortcuts

- *telegram.CallbackQuery.edit_message_reply_markup()*
- *telegram.Message.edit_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.
- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message to be edited was sent

  Added in version 21.4.

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. See *do_api_request()* for limitations.

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*, *reply_markup=None*, *entities=None*, *link_preview_options=None*, *business_connection_id=None*, ***, *disable_web_page_preview=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.
- Note that business messages that were not sent by the bot and do not contain an inline keyboard can only be edited within *48 hours* from the time they were sent.

See also

telegram.Game.text

Shortcuts

- *telegram.CallbackQuery.edit_message_text()*
- *telegram.Message.edit_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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in message text, which can be specified instead of *parse_mode*.

Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list.

- **link_preview_options** (*LinkPreviewOptions*, optional) – Link preview generation options for the message. Mutually exclusive with *disable_web_page_preview*.

  Added in version 20.8.

- **reply_markup** (*telegram.InlineKeyboardMarkup*, optional) – An object for an inline keyboard.

- **business_connection_id** (str, optional) – Unique identifier of the business connection on behalf of which the message to be edited was sent

  Added in version 21.4.

Keyword Arguments

- **disable_web_page_preview** (bool, optional) – Disables link previews for links in this message. Convenience parameter for setting *link_preview_options*. Mutually exclusive with *link_preview_options*.

  Changed in version 20.8: Bot API 7.0 introduced *link_preview_options* replacing this argument. PTB will automatically convert this argument to that one, but for advanced options, please use *link_preview_options* directly.

  Changed in version 21.0: This argument is now a keyword-only argument.

- **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. See *do_api_request()* for limitations.

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

- **ValueError** – If both *disable_web_page_preview* and *link_preview_options* are passed.

- **telegram.error.TelegramError** – For other errors.

async edit_story(*business_connection_id*, *story_id*, *content*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *areas=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Edits a story previously posted by the bot on behalf of a managed business account. Requires the *can_manage_stories* business bot right.

Added in version 22.1.

Parameters

- **business_connection_id** (str) – Unique identifier of the business connection.

- ***story_id*** (int) – Unique identifier of the story to edit.
- ***content*** (*telegram.InputStoryContent*) – Content of the story.
- ***caption*** (str, optional) – Caption of the story, 0-*2048* characters after entities parsing.
- ***parse_mode*** (str, optional) – Mode for parsing entities in the story caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of parse_mode.
- ***areas*** (Sequence[*telegram.StoryArea*], optional) – Sequence of clickable areas to be shown on the story.

> **Note**
>
> Each type of clickable area in *areas* has its own maximum limit:
>
> – Up to *10* of *telegram.StoryAreaTypeLocation*.
>
> – Up to *5* of *telegram.StoryAreaTypeSuggestedReaction*.
>
> – Up to *3* of *telegram.StoryAreaTypeLink*.
>
> – Up to *3* of *telegram.StoryAreaTypeWeather*.
>
> – Up to *1* of *telegram.StoryAreaTypeUniqueGift*.

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. See *do_api_request()* for limitations.

Returns

Story

Raises

telegram.error.TelegramError –

async edit_user_star_subscription(*user_id*, *telegram_payment_charge_id*, *is_canceled*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Allows the bot to cancel or re-enable extension of a subscription paid in Telegram Stars.

Added in version 21.8.

Parameters

- ***user_id*** (int) – Identifier of the user whose subscription will be edited.
- ***telegram_payment_charge_id*** (str) – Telegram payment identifier for the subscription.

- ***is_canceled*** (bool) – Pass True to cancel extension of the user subscription; the subscription must be active up to the end of the current subscription period. Pass False to allow the user to re-enable a subscription that was previously canceled by the bot.

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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

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.

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.

Shortcuts

- *telegram.Chat.export_invite_link()*
- *telegram.ChatFullInfo.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. See *do_api_request()* for limitations.

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*, *message_thread_id=None*, *video_start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_effect_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *forward_message()*

async forwardMessages(*chat_id*, *from_chat_id*, *message_ids*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *direct_messages_topic_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *forward_messages()*

async forward_message(*chat_id*, *from_chat_id*, *message_id*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *video_start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_effect_id=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.ChatFullInfo.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.

Shortcuts

- *telegram.Chat.forward_from()*
- *telegram.Chat.forward_to()*
- *telegram.ChatFullInfo.forward_from()*
- *telegram.ChatFullInfo.forward_to()*
- *telegram.Message.forward()*
- *telegram.User.forward_from()*
- *telegram.User.forward_to()*

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*.
- ***video_start_timestamp*** (int, optional) – New start timestamp for the forwarded video in the message

  Added in version 21.11.
- ***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.

  Added in version 13.10.
- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.
- ***suggested_post_parameters*** (*telegram.SuggestedPostParameters*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only.

  Added in version 22.4.
- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the message will be forwarded; required if the message is forwarded to a direct messages chat.

  Added in version 22.4.
- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; only available when forwarding to private chats

  Added in version 22.6.

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. See *do_api_request()* for limitations.

Returns
On success, the sent Message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async forward_messages(*chat_id*, *from_chat_id*, *message_ids*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *direct_messages_topic_id=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. If some of the specified messages can't be found or forwarded, they are skipped. Service messages and messages with protected content can't be forwarded. Album grouping is kept for forwarded messages.

Shortcuts

- *telegram.Chat.forward_messages_from()*
- *telegram.Chat.forward_messages_to()*
- *telegram.ChatFullInfo.forward_messages_from()*
- *telegram.ChatFullInfo.forward_messages_to()*
- *telegram.User.forward_messages_from()*
- *telegram.User.forward_messages_to()*

Added in version 20.8.

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_ids*** (Sequence[int]) – A list of *1*- *100* identifiers of messages in the chat *from_chat_id* to forward. The identifiers must be specified in a strictly increasing order.
- ***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.
- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the messages will be forwarded; required if the messages are forwarded to a direct messages chat.

  Added in version 22.4.

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. See *do_api_request()* for limitations.

Returns
On success, a tuple of MessageId of sent messages is returned.

Return type
tuple[*telegram.Message*]

Raises
telegram.error.TelegramError –

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

Alias for *get_available_gifts()*

async getBusinessAccountGifts(*business_connection_id, exclude_unsaved=None, exclude_saved=None, exclude_unlimited=None, exclude_unique=None, sort_by_price=None, offset=None, limit=None, exclude_limited_upgradable=None, exclude_limited_non_upgradable=None, exclude_from_blockchain=None, *, read_timeout=None, write_timeout=None, connect_timeout=None, pool_timeout=None, api_kwargs=None*)

Alias for *get_business_account_gifts()*

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

Alias for *get_business_account_star_balance()*

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

Alias for *get_business_connection()*

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, return_bots=None, *, read_timeout=None, write_timeout=None, connect_timeout=None, pool_timeout=None, api_kwargs=None*)

Alias for *get_chat_administrators()*

async getChatGifts(*chat_id*, *exclude_unsaved=None*, *exclude_saved=None*, *exclude_unlimited=None*, *exclude_limited_upgradable=None*, *exclude_limited_non_upgradable=None*, *exclude_from_blockchain=None*, *exclude_unique=None*, *sort_by_price=None*, *offset=None*, *limit=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_chat_gifts()*

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 getForumTopicIconStickers(***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_forum_topic_icon_stickers()*

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

Alias for *get_managed_bot_access_settings()*

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

Alias for *get_managed_bot_token()*

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 getMyDescription(*language_code=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_my_description()*

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

Alias for *get_my_name()*

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

Alias for *get_my_short_description()*

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

Alias for *get_my_star_balance()*

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

Alias for *get_star_transactions()*

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=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_updates()*

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

Alias for *get_user_chat_boosts()*

async getUserGifts(*user_id*, *exclude_unlimited=None*, *exclude_limited_upgradable=None*, *exclude_limited_non_upgradable=None*, *exclude_from_blockchain=None*, *exclude_unique=None*, *sort_by_price=None*, *offset=None*, *limit=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *get_user_gifts()*

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

Alias for *get_user_personal_chat_messages()*

async getUserProfileAudios(*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_audios()*

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_available_gifts(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Returns the list of gifts that can be sent by the bot to users and channel chats. Requires no parameters.

Added in version 21.8.

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. See *do_api_request()* for limitations.

Returns

telegram.Gifts

Raises

telegram.error.TelegramError –

async get_business_account_gifts(*business_connection_id*, *exclude_unsaved=None*, *exclude_saved=None*, *exclude_unlimited=None*, *exclude_unique=None*, *sort_by_price=None*, *offset=None*, *limit=None*, *exclude_limited_upgradable=None*, *exclude_limited_non_upgradable=None*, *exclude_from_blockchain=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Returns the gifts received and owned by a managed business account. Requires the *can_view_gifts_and_stars* business bot right.

Added in version 22.1.

Removed in version 22.7: Bot API 9.3 removed the *exclude_limited* parameter. Use *exclude_limited_upgradable* and *exclude_limited_non_upgradable* instead.

Parameters

- ***business_connection_id*** (str) – Unique identifier of the business connection.
- ***exclude_unsaved*** (bool, optional) – Pass True to exclude gifts that aren't saved to the account's profile page.
- ***exclude_saved*** (bool, optional) – Pass True to exclude gifts that are saved to the account's profile page.
- ***exclude_unlimited*** (bool, optional) – Pass True to exclude gifts that can be purchased an unlimited number of times.
- ***exclude_limited_upgradable*** (bool, optional) – Pass True to exclude gifts that can be purchased a limited number of times and can be upgraded to unique.

  Added in version 22.6.
- ***exclude_limited_non_upgradable*** (bool, optional) – Pass True to exclude gifts that can be purchased a limited number of times and can't be upgraded to unique

  Added in version 22.6.

- ***exclude_unique*** (bool, optional) – Pass True to exclude unique gifts.
- ***exclude_from_blockchain*** (bool, optional) – Pass True to exclude gifts that were assigned from the TON blockchain and can't be resold or transferred in Telegram.

  Added in version 22.6.
- ***sort_by_price*** (bool, optional) – Pass True to sort results by gift price instead of send date. Sorting is applied before pagination.
- ***offset*** (str, optional) – Offset of the first entry to return as received from the previous request; use empty string to get the first chunk of results.
- ***limit*** (int, optional) – The maximum number of gifts to be returned; *1*- *100*. 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. See *do_api_request()* for limitations.

Returns

telegram.OwnedGifts

Raises

telegram.error.TelegramError –

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

Returns the amount of Telegram Stars owned by a managed business account. Requires the *can_view_gifts_and_stars* business bot right.

Added in version 22.1.

Parameters

business_connection_id (str) – Unique identifier of the business connection.

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. See *do_api_request()* for limitations.

Returns

telegram.StarAmount

Raises
telegram.error.TelegramError –

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

Use this method to get information about the connection of the bot with a business account.

Added in version 21.1.

Parameters
business_connection_id (str) – Unique identifier of the business connection.

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. See *do_api_request()* for limitations.

Returns

On success, the object containing the business
connection information is returned.

Return type
telegram.BusinessConnection

Raises
telegram.error.TelegramError –

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.).

Changed in version 21.2: In accordance to Bot API 7.3, this method now returns a *telegram.ChatFullInfo*.

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. See *do_api_request()* for limitations.

Returns
telegram.ChatFullInfo

Raises
telegram.error.TelegramError –

async get_chat_administrators(*chat_id*, *return_bots=None*, ***, *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.

Shortcuts

- *telegram.Chat.get_administrators()*
- *telegram.ChatFullInfo.get_administrators()*

Changed in version 20.0: Returns a tuple instead of a list.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***return_bots*** (bool, optional) – Pass `True` to additionally receive all bots that are administrators of the chat. By default, bots other than the current bot are omitted.

  Added in version 22.8.

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. See *do_api_request()* for limitations.

Returns
On success, returns a tuple 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
tuple[*telegram.ChatMember*]

Raises
telegram.error.TelegramError –

async get_chat_gifts(*chat_id*, *exclude_unsaved=None*, *exclude_saved=None*, *exclude_unlimited=None*, *exclude_limited_upgradable=None*, *exclude_limited_non_upgradable=None*, *exclude_from_blockchain=None*, *exclude_unique=None*, *sort_by_price=None*, *offset=None*, *limit=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to get gifts owned by a chat.

Shortcuts

- *telegram.Chat.get_gifts()*
- *telegram.ChatFullInfo.get_gifts()*

Added in version 22.6.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***exclude_unsaved*** (bool, optional) – Pass True to exclude gifts that aren't saved to the chat's profile page. Always True, unless the bot has the *can_post_messages* administrator right in the channel.
- ***exclude_saved*** (bool, optional) – Pass True to exclude gifts that are saved to the chat's profile page. Always False, unless the bot has the *can_post_messages* administrator right in the channel.
- ***exclude_unlimited*** (bool, optional) – Pass True to exclude gifts that can be purchased an unlimited number of times
- ***exclude_limited_upgradable*** (bool, optional) – Pass True to exclude gifts that can be purchased a limited number of times and can be upgraded to unique
- ***exclude_limited_non_upgradable*** (bool, optional) – Pass True to exclude gifts that can be purchased a limited number of times and can't be upgraded to unique.
- ***exclude_from_blockchain*** (bool, optional) – Pass True to exclude gifts that were assigned from the TON blockchain and can't be resold or transferred in Telegram.
- ***exclude_unique*** (bool, optional) – Pass True to exclude unique gifts
- ***sort_by_price*** (bool, optional) – Pass True to sort results by gift price instead of send date. Sorting is applied before pagination.
- ***offset*** (str, optional) – Offset of the first entry to return as received from the previous request; use an empty string to get the first chunk of results
- ***limit*** (int, optional) – The maximum number of gifts to be returned; *1* - *100*. 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. See *do_api_request()* for limitations.

Returns

The owned gifts for the chat.

Return type
telegram.OwnedGifts

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. The method is only guaranteed to work for other users if the bot is an administrator in the chat.

Shortcuts

- *telegram.Chat.get_member()*
- *telegram.ChatFullInfo.get_member()*

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. See *do_api_request()* for limitations.

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.

Shortcuts

- *telegram.Chat.get_member_count()*
- *telegram.ChatFullInfo.get_member_count()*

Added in version 13.7.

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. See *do_api_request()* for limitations.

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.set_menu_button()*, *telegram.User.set_menu_button()*

Shortcuts

- *telegram.Chat.get_menu_button()*
- *telegram.ChatFullInfo.get_menu_button()*
- *telegram.User.get_menu_button()*

Added in version 20.0.

Parameters
chat_id (int, optional) – Unique identifier for the target private chat. If not specified, the bot's default 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. See *do_api_request()* for limitations.

Returns
On success, the current menu button is returned.

Return type
telegram.MenuButton

Raises
telegram.error.TelegramError –

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.

Changed in version 20.0: Returns a tuple instead of a list.

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

Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list.

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. See *do_api_request()* for limitations.

Returns
tuple[*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 e.g. downloaded with *telegram.File.download_to_drive()*. 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.

See also

Working with Files and Media

Shortcuts

- *telegram.Animation.get_file()*
- *telegram.Audio.get_file()*
- *telegram.ChatPhoto.get_big_file()*
- *telegram.ChatPhoto.get_small_file()*
- *telegram.Document.get_file()*
- telegram.LivePhoto.get_file()
- *telegram.PhotoSize.get_file()*
- *telegram.Sticker.get_file()*
- *telegram.Video.get_file()*
- *telegram.VideoNote.get_file()*
- *telegram.VideoQuality.get_file()*
- *telegram.Voice.get_file()*

Parameters

file_id (str | *telegram.Animation* | *telegram.Audio* | *telegram.ChatPhoto* | *telegram.Document* | *telegram.PhotoSize* | *telegram.Sticker* | *telegram.Video* | *telegram.VideoNote* | *telegram.Voice*) – Either the file identifier or an object that has a file_id attribute to get file information about.

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. See *do_api_request()* for limitations.

Returns

telegram.File

Raises

telegram.error.TelegramError –

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

Use this method to get custom emoji stickers, which can be used as a forum topic icon by any user. Requires no parameters.

Added 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. See *do_api_request()* for limitations.

Returns

tuple[*telegram.Sticker*]

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.

Shortcuts

- *telegram.CallbackQuery.get_game_high_scores()*
- *telegram.Message.get_game_high_scores()*

Changed in version 20.0: Returns a tuple instead of a list.

Parameters

- ***user_id*** (int) – Target user id.
- ***chat_id*** (int, 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. See *do_api_request()* for limitations.

Returns
tuple[*telegram.GameHighScore*]

Raises
telegram.error.TelegramError –

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

Use this method to get the access settings of a managed bot.

Shortcuts

telegram.User.get_managed_bot_access_settings()

Added in version 22.8.

Parameters
user_id (int) – User identifier of the managed bot whose access settings 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. See *do_api_request()* for limitations.

Returns
The access settings of the managed bot.

Return type
telegram.BotAccessSettings

Raises
telegram.error.TelegramError –

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

Use this method to get the token of a managed bot.

Added in version 22.8.

Parameters
user_id (int) – User identifier of the managed bot whose token 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. See *do_api_request()* for limitations.

Returns
The token of the managed bot.

Return type
str

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. See *do_api_request()* for limitations.

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.

> **See also**
>
> *set_my_commands()*, *delete_my_commands()*

Changed in version 20.0: Returns a tuple instead of a list.

Parameters

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

  Added in version 13.7.

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

  Added 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. See *do_api_request()* for limitations.

Returns

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

Return type

tuple[*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()*

Added 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. See *do_api_request()* for limitations.

Returns
On success.

Return type
telegram.ChatAdministratorRights

Raises
telegram.error.TelegramError –

async get_my_description(*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 bot description for the given user language.

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

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. See *do_api_request()* for limitations.

Returns
On success, the bot description is returned.

Return type
telegram.BotDescription

Raises
telegram.error.TelegramError –

async get_my_name(*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 bot name for the given user language.

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

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. See *do_api_request()* for limitations.

Returns
On success, the bot name is returned.

Return type
telegram.BotName

Raises
telegram.error.TelegramError –

async get_my_short_description(*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 bot short description for the given user language.

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

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. See *do_api_request()* for limitations.

Returns

On success, the bot short description is returned.

Return type
telegram.BotShortDescription

Raises
telegram.error.TelegramError –

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

A method to get the current Telegram Stars balance of the bot. Requires no parameters.

Added in version 22.3.

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. See *do_api_request()* for limitations.

Returns
telegram.StarAmount

Raises
telegram.error.TelegramError –

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

Returns the bot's Telegram Star transactions in chronological order.

Added in version 21.4.

Parameters

- ***offset*** (int, optional) – Number of transactions to skip in the response.
- ***limit*** (int, optional) – The maximum number of transactions 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. See *do_api_request()* for limitations.

Returns
On success.

Return type
telegram.StarTransactions

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. See *do_api_request()* for limitations.

Returns
telegram.StickerSet

Raises
telegram.error.TelegramError –

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

Use this method to receive incoming updates using long polling.

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*

See also

telegram.ext.Application.run_polling(), *telegram.ext.Updater.start_polling()*

Changed in version 20.0: Returns a tuple instead of a list.

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 be forgotten.
- ***limit*** (int, optional) – Limits the number of updates to be retrieved. Values between *1*- *100* are accepted. Defaults to `100`.
- ***timeout*** (int | datetime.timedelta, 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.

  Changed in version v22.2: datetime.timedelta objects are accepted in addition to plain int values.
- ***allowed_updates*** (Sequence[str]), optional) – A sequence 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 sequence to receive all updates except *telegram.Update.chat_member*, *telegram.Update.message_reaction* and *telegram.Update.message_reaction_count* (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.

Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list.

Keyword Arguments

- ***read_timeout*** (`float` | `None`, optional) – Value to pass to *telegram.request.BaseRequest.post.read_timeout*. Defaults to *DEFAULT_NONE*. *timeout* will be added to this value.

  Changed in version 20.7: Defaults to *DEFAULT_NONE* instead of `2`.

- ***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. See *do_api_request()* for limitations.

Returns

tuple[*telegram.Update*]

Raises

telegram.error.TelegramError –

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

Use this method to get the list of boosts added to a chat by a user. Requires administrator rights in the chat.

Shortcuts

- *telegram.Chat.get_user_chat_boosts()*
- *telegram.ChatFullInfo.get_user_chat_boosts()*
- *telegram.User.get_chat_boosts()*

Added in version 20.8.

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. See *do_api_request()* for limitations.

Returns

On success, the object containing the list of boosts is returned.

Return type

telegram.UserChatBoosts

Raises

telegram.error.TelegramError –

async get_user_gifts(*user_id*, *exclude_unlimited=None*, *exclude_limited_upgradable=None*, *exclude_limited_non_upgradable=None*, *exclude_from_blockchain=None*, *exclude_unique=None*, *sort_by_price=None*, *offset=None*, *limit=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Returns the gifts owned and hosted by a user.

Shortcuts

telegram.User.get_gifts()

Added in version 22.6.

Parameters

- **user_id** (int) – Unique identifier of the user
- **exclude_unlimited** (bool, optional) – Pass True to exclude gifts that can be purchased an unlimited number of times
- **exclude_limited_upgradable** (bool, optional) – Pass True to exclude gifts that can be purchased a limited number of times and can be upgraded to unique
- **exclude_limited_non_upgradable** (bool, optional) – Pass True to exclude gifts that can be purchased a limited number of times and can't be upgraded to unique.
- **exclude_from_blockchain** (bool, optional) – Pass True to exclude gifts that were assigned from the TON blockchain and can't be resold or transferred in Telegram.
- **exclude_unique** (bool, optional) – Pass True to exclude unique gifts
- **sort_by_price** (bool, optional) – Pass True to sort results by gift price instead of send date. Sorting is applied before pagination.
- **offset** (str, optional) – Offset of the first entry to return as received from the previous request; use an empty string to get the first chunk of results
- **limit** (int, optional) – The maximum number of gifts to be returned; *1* - *100*. 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. See *do_api_request()* for limitations.

Returns
The owned gifts for the user.

Return type
telegram.OwnedGifts

Raises
telegram.error.TelegramError –

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

Use this method to get the last messages from the personal chat (i.e., the chat currently added to their profile) of a given user.

Shortcuts

telegram.User.get_personal_chat_messages()

Added in version 22.8.

Parameters

- ***user_id*** (int) – Unique identifier of the target user.
- ***limit*** (int) – The maximum number of messages to return; *1*- *20*.

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. See *do_api_request()* for limitations.

Returns
On success, a tuple of *telegram.Message* objects is returned.

Return type
tuple[*telegram.Message*, ...]

Raises
telegram.error.TelegramError –

async get_user_profile_audios(*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 audios for a user.

Shortcuts

telegram.User.get_profile_audios()

Added in version 22.7.

Parameters

- ***user_id*** (int) – Unique identifier of the target user.
- ***offset*** (int, optional) – Sequential number of the first audio to be returned. By default, all audios are returned.
- ***limit*** (int, optional) – Limits the number of audios 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. See *do_api_request()* for limitations.

Returns

telegram.UserProfileAudios

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.

Shortcuts

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. See *do_api_request()* for limitations.

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. See *do_api_request()* for limitations.

Returns
telegram.WebhookInfo

Raises
telegram.error.TelegramError –

async giftPremiumSubscription(*user_id*, *month_count*, *star_count*, *text=None*, *text_parse_mode=None*, *text_entities=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *gift_premium_subscription()*

async gift_premium_subscription(*user_id*, *month_count*, *star_count*, *text=None*, *text_parse_mode=None*, *text_entities=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Gifts a Telegram Premium subscription to the given user.

Shortcuts

telegram.User.gift_premium_subscription()

Added in version 22.1.

Parameters

- ***user_id*** (int) – Unique identifier of the target user who will receive a Telegram Premium subscription.
- ***month_count*** (int) – Number of months the Telegram Premium subscription will be active for the user; must be one of *3*, *6*, or *12*.
- ***star_count*** (int) – Number of Telegram Stars to pay for the Telegram Premium subscription; must be *1000* for *3* months, *1500* for *6* months, and *2500* for *12* months.
- ***text*** (str, optional) – Text that will be shown along with the service message about the subscription; 0-*128* characters.
- ***text_parse_mode*** (str, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details. Entities other than *BOLD*, *ITALIC*, *UNDERLINE*, *STRIKETHROUGH*, *SPOILER* *CUSTOM_EMOJI*, and *DATE_TIME* are ignored.
- ***text_entities*** (Sequence[*telegram.MessageEntity*], optional) – A list of special entities that appear in the gift text. It can be specified instead of *text_parse_mode*. Entities other than *BOLD*, *ITALIC*, *UNDERLINE*, *STRIKETHROUGH*, *SPOILER*, *CUSTOM_EMOJI*, and *DATE_TIME* are ignored.

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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

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

Alias for *hide_general_forum_topic()*

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

Use this method to hide the 'General' topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have *can_manage_topics* administrator rights. The topic will be automatically closed if it was open.

Shortcuts

- *telegram.Chat.hide_general_forum_topic()*
- *telegram.ChatFullInfo.hide_general_forum_topic()*

Added in version 20.0.

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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

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()

Added 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.

Shortcuts

- *telegram.Chat.leave()*
- *telegram.ChatFullInfo.leave()*

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. See *do_api_request()* for limitations.

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

property local_mode

Whether this bot is running in local mode.

Added in version 20.0.

Type

bool

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

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*, *business_connection_id=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*, *business_connection_id=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.

Shortcuts

- *telegram.Chat.pin_message()*
- *telegram.ChatFullInfo.pin_message()*
- *telegram.Message.pin()*
- *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.
- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be pinned.

  Added in version 21.5.

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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

async postStory(*business_connection_id*, *content*, *active_period*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *areas=None*, *post_to_chat_page=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *post_story()*

async post_story(*business_connection_id*, *content*, *active_period*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *areas=None*, *post_to_chat_page=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Posts a story on behalf of a managed business account. Requires the *can_manage_stories* business bot right.

Added in version 22.1.

Parameters

- ***business_connection_id*** (str) – Unique identifier of the business connection.
- ***content*** (*telegram.InputStoryContent*) – Content of the story.
- ***active_period*** (int | datetime.timedelta, optional) – Period after which the story is moved to the archive, in seconds; must be one of *21600*, *43200*, *86400*, or *172800*.
- ***caption*** (str, optional) – Caption of the story, 0-*2048* characters after entities parsing.
- ***parse_mode*** (str, optional) – Mode for parsing entities in the story caption. See the constants in *telegram.constants.ParseMode* for the available modes.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of parse_mode.
- ***areas*** (Sequence[*telegram.StoryArea*], optional) – Sequence of clickable areas to be shown on the story.

> **Note**
>
> Each type of clickable area in *areas* has its own maximum limit:
>
> - Up to *10* of *telegram.StoryAreaTypeLocation*.
> - Up to *5* of *telegram.StoryAreaTypeSuggestedReaction*.
> - Up to *3* of *telegram.StoryAreaTypeLink*.
> - Up to *3* of *telegram.StoryAreaTypeWeather*.
> - Up to *1* of *telegram.StoryAreaTypeUniqueGift*.

- ***post_to_chat_page*** (*telegram.InputStoryContent*, optional) – Pass `True` to keep the story accessible after it expires.
- ***protect_content*** (`bool`, optional) – Pass `True` if the content of the story must be protected from forwarding and screenshotting

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. See *do_api_request()* for limitations.

Returns

Story

Raises

telegram.error.TelegramError –

property private_key

Deserialized private key for decryption of telegram passport data.

Added in version 20.0.

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*, *can_manage_topics=None*, *can_post_stories=None*, *can_edit_stories=None*, *can_delete_stories=None*, *can_manage_direct_messages=None*, *can_manage_tags=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*, *can_manage_topics=None*, *can_post_stories=None*, *can_edit_stories=None*, *can_delete_stories=None*, *can_manage_direct_messages=None*, *can_manage_tags=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.

Shortcuts

- *telegram.Chat.promote_member()*
- *telegram.ChatFullInfo.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, get boost list, see hidden supergroup and channel members, report spam messages and ignore slow mode. Implied by any other administrator privilege.

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

  Added 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 post messages in the channel, or access channel statistics; for channels only.
- ***can_edit_messages*** (`bool`, optional) – Pass `True`, if the administrator can edit messages of other users and can pin messages, for 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, or access supergroup statistics. For backward compatibility, defaults to `True` for promotions of channel administrators.

- ***can_pin_messages*** (bool, optional) – Pass True, if the administrator can pin messages, for supergroups only.
- ***can_promote_members*** (bool, optional) – Pass True, if the administrator can add new administrators with a subset of their own privileges or demote administrators that they have promoted, directly or indirectly (promoted by administrators that were appointed by the user).
- ***can_manage_topics*** (bool, optional) – Pass True, if the user is allowed to create, rename, close, and reopen forum topics; for supergroups only.

  Added in version 20.0.
- ***can_post_stories*** (bool, optional) – Pass True, if the administrator can post stories to the chat.

  Added in version 20.6.
- ***can_edit_stories*** (bool, optional) – Pass True, if the administrator can edit stories posted by other users, post stories to the chat page, pin chat stories, and access the chat's story archive.

  Added in version 20.6.
- ***can_delete_stories*** (bool, optional) – Pass True, if the administrator can delete stories posted by other users.

  Added in version 20.6.
- ***can_manage_direct_messages*** (bool, optional) – Pass True, if the administrator can manage direct messages within the channel and decline suggested posts; for channels only.

  Added in version 22.4.
- ***can_manage_tags*** (bool, optional) – Pass True, if the administrator can edit the tags of regular members; for groups and supergroups only.

  Added in version 22.7.

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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

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

Alias for *read_business_message()*

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

Marks incoming message as read on behalf of a business account. Requires the *can_read_messages* business bot right.

Shortcuts

- *telegram.Chat.read_business_message()*
- *telegram.ChatFullInfo.read_business_message()*
- *telegram.Message.read_business_message()*

Added in version 22.1.

Parameters

- ***business_connection_id*** (str) – Unique identifier of the business connection on behalf of which to read the message.
- ***chat_id*** (int) – Unique identifier of the chat in which the message was received. The chat must have been active in the last *86400* seconds.
- ***message_id*** (int) – Unique identifier of the message to mark as read.

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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

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

Alias for *refund_star_payment()*

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

Refunds a successful payment in Telegram Stars.

Shortcuts

telegram.User.refund_star_payment()

Added in version 21.3.

Parameters

- ***user_id*** (int) – User identifier of the user whose payment will be refunded.
- ***telegram_payment_charge_id*** (str) – Telegram payment identifier.

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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

async removeBusinessAccountProfilePhoto(*business_connection_id*, *is_public=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *remove_business_account_profile_photo()*

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

Alias for *remove_chat_verification()*

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

Alias for *remove_my_profile_photo()*

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

Alias for *remove_user_verification()*

async remove_business_account_profile_photo(*business_connection_id*, *is_public=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Removes the current profile photo of a managed business account. Requires the *can_edit_profile_photo* business bot right.

Added in version 22.1.

Parameters

- ***business_connection_id*** (str) – Unique identifier of the business connection.
- ***is_public*** (bool, optional) – Pass True to remove the public photo, which will be visible even if the main photo is hidden by the business account's privacy settings. After the main photo is removed, the previous profile photo (if present) becomes the main photo.

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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

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

Removes verification from a chat that is currently verified on behalf of the organization represented by the bot.

Shortcuts

- *telegram.Chat.remove_verification()*
- *telegram.ChatFullInfo.remove_verification()*

Added in version 21.10.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

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

Removes the profile photo of the bot. Requires no parameters.

Added in version 22.7.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

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

Removes verification from a user who is currently verified on behalf of the organization represented by the bot.

Shortcuts

telegram.User.remove_verification()

Added in version 21.10.

Parameters
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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

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

Alias for *reopen_forum_topic()*

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

Alias for *reopen_general_forum_topic()*

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

Use this method to reopen a closed topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have *can_manage_topics* administrator rights, unless it is the creator of the topic.

Shortcuts

- *telegram.Chat.reopen_forum_topic()*
- *telegram.ChatFullInfo.reopen_forum_topic()*
- *telegram.Message.reopen_forum_topic()*

Added in version 20.0.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target supergroup (in the format `@supergroupusername`).
- ***message_thread_id*** (int) – Unique identifier for the target message thread of the forum topic.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

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

Use this method to reopen a closed 'General' topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have *can_manage_topics* administrator rights. The topic will be automatically unhidden if it was hidden.

Shortcuts

- *telegram.Chat.reopen_general_forum_topic()*
- *telegram.ChatFullInfo.reopen_general_forum_topic()*

Added in version 20.0.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

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

Alias for *replace_managed_bot_token()*

async replaceStickerInSet(*user_id*, *name*, *old_sticker*, *sticker*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *replace_sticker_in_set()*

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

Use this method to revoke the current token of a managed bot and generate a new one.

Shortcuts

telegram.User.replace_token()

Added in version 22.8.

Parameters

user_id (int) – User identifier of the managed bot whose token will be replaced.

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. See *do_api_request()* for limitations.

Returns

The new token of the managed bot.

Return type

str

Raises

telegram.error.TelegramError –

async replace_sticker_in_set(*user_id*, *name*, *old_sticker*, *sticker*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to replace an existing sticker in a sticker set with a new one. The method is equivalent to calling *delete_sticker_from_set()*, then *add_sticker_to_set()*, then *set_sticker_position_in_set()*.

Added in version 21.1.

Parameters

- ***user_id*** (int) – User identifier of the sticker set owner.
- ***name*** (str) – Sticker set name.
- ***old_sticker*** (str | *Sticker*) – File identifier of the replaced sticker or the sticker object itself.

  Changed in version 21.10: Accepts also *telegram.Sticker* instances.
- ***sticker*** (*telegram.InputSticker*) – An object with information about the added sticker. If exactly the same sticker had already been added to the set, then the set remains unchanged.

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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

async repostStory(*business_connection_id*, *from_chat_id*, *from_story_id*, *active_period*, *post_to_chat_page=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *repost_story()*

async repost_story(*business_connection_id*, *from_chat_id*, *from_story_id*, *active_period*, *post_to_chat_page=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Reposts a story on behalf of a business account from another business account. Both business accounts must be managed by the same bot, and the story on the source account must have been posted (or reposted) by the bot. Requires the *can_manage_stories* business bot right for both business accounts.

Shortcuts

- *telegram.Chat.repost_story()*
- *telegram.ChatFullInfo.repost_story()*
- *telegram.Story.repost()*
- *telegram.User.repost_story()*

Added in version 22.6.

Parameters

- ***business_connection_id*** (str) – Unique identifier of the business
- ***from_chat_id*** (int) – Unique identifier of the chat which posted the story that should be reposted
- ***from_story_id*** (int) – Unique identifier of the story that should be reposted
- ***active_period*** (int | datetime.timedelta) – Period after which the story is moved to the archive, in seconds; must be one of *21600*, *43200*, *86400*, or *172800*.
- ***post_to_chat_page*** (bool, optional) – Pass True to keep the story accessible after it expires.

- ***protect_content*** (bool, optional) – Pass True if the content of the story must be protected from forwarding and screenshotting

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. See *do_api_request()* for limitations.

Returns

telegram.Story

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*, *use_independent_chat_permissions=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*, *use_independent_chat_permissions=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()*

> **Shortcuts**
>
> - *telegram.Chat.restrict_member()*
> - *telegram.ChatFullInfo.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.
- ***use_independent_chat_permissions*** (bool, optional) – Pass `True` if chat permissions are set independently. Otherwise, the *can_send_other_messages* and *can_add_web_page_previews* permissions will imply the *can_send_messages*, *can_send_audios*, *can_send_documents*, *can_send_photos*, *can_send_videos*, *can_send_video_notes*, and *can_send_voice_notes* permissions; the *can_send_polls* permission will imply the *can_send_messages* permission.

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. See *do_api_request()* for limitations.

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.

Shortcuts

- *telegram.Chat.revoke_invite_link()*
- *telegram.ChatFullInfo.revoke_invite_link()*

Added 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. See *do_api_request()* for limitations.

Returns

telegram.ChatInviteLink

Raises

telegram.error.TelegramError –

async savePreparedInlineMessage(*user_id*, *result*, *allow_user_chats=None*, *allow_bot_chats=None*, *allow_group_chats=None*, *allow_channel_chats=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *save_prepared_inline_message()*

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

Alias for *save_prepared_keyboard_button()*

async save_prepared_inline_message(*user_id*, *result*, *allow_user_chats=None*, *allow_bot_chats=None*, *allow_group_chats=None*, *allow_channel_chats=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Stores a message that can be sent by a user of a Mini App.

Added in version 21.8.

Parameters

- ***user_id*** (int) – Unique identifier of the target user that can use the prepared message.
- ***result*** (*telegram.InlineQueryResult*) – The result to store.
- ***allow_user_chats*** (bool, optional) – Pass True if the message can be sent to private chats with users
- ***allow_bot_chats*** (bool, optional) – Pass True if the message can be sent to private chats with bots
- ***allow_group_chats*** (bool, optional) – Pass True if the message can be sent to group and supergroup chats

- ***allow_channel_chats*** (bool, optional) – Pass True if the message can be sent to channels

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. See *do_api_request()* for limitations.

Returns

On success, the prepared message is returned.

Return type

telegram.PreparedInlineMessage

Raises

telegram.error.TelegramError –

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

Stores a keyboard button that can be used by a user within a Mini App.

Added in version 22.8.

Parameters

- ***user_id*** (int) – Unique identifier of the target user that can use the button.
- ***button*** (*telegram.KeyboardButton*) – An object describing the button to be saved. The button must be of the type *request_users*, *request_chat*, or *request_managed_bot*.

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. See *do_api_request()* for limitations.

Returns

telegram.PreparedKeyboardButton

Raises

telegram.error.TelegramError –

async sendAnimation(*chat_id*, *animation*, *duration=None*, *width=None*, *height=None*, *caption=None*, *parse_mode=None*, *disable_notification=None*, *reply_markup=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_audio()*

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

Alias for *send_chat_action()*

async sendChecklist(*business_connection_id*, *chat_id*, *checklist*, *disable_notification=None*, *protect_content=None*, *message_effect_id=None*, *reply_parameters=None*, *reply_markup=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_checklist()*

async sendContact(*chat_id*, *phone_number=None*, *first_name=None*, *last_name=None*, *disable_notification=None*, *reply_markup=None*, *vcard=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=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_markup=None*, *emoji=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=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_markup=None*, *parse_mode=None*, *disable_content_type_detection=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_document()*

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

Alias for *send_game()*

async sendGift(*gift_id*, *text=None*, *text_parse_mode=None*, *text_entities=None*, *pay_for_upgrade=None*, *chat_id=None*, *user_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_gift()*

async sendInvoice(*chat_id*, *title*, *description*, *payload*, *currency*, *prices*, *provider_token=None*, *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_markup=None*, *provider_data=None*, *send_phone_number_to_provider=None*, *send_email_to_provider=None*, *max_tip_amount=None*, *suggested_tip_amounts=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_invoice()*

async sendLivePhoto(*chat_id*, *live_photo*, *photo*, *business_connection_id=None*, *message_thread_id=None*, *direct_messages_topic_id=None*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, *has_spoiler=None*, *disable_notification=None*, *protect_content=None*, *allow_paid_broadcast=None*, *message_effect_id=None*, *suggested_post_parameters=None*, *reply_parameters=None*, *reply_markup=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_live_photo()*

async sendLocation(*chat_id*, *latitude=None*, *longitude=None*, *disable_notification=None*, *reply_markup=None*, *live_period=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=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*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*, *caption=None*, *parse_mode=None*, *caption_entities=None*)

Alias for *send_media_group()*

async sendMessage(*chat_id*, *text*, *parse_mode=None*, *entities=None*, *disable_notification=None*, *protect_content=None*, *reply_markup=None*, *message_thread_id=None*, *link_preview_options=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *disable_web_page_preview=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_message()*

async sendMessageDraft(*chat_id*, *draft_id*, *text=None*, *message_thread_id=None*, *parse_mode=None*, *entities=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_message_draft()*

async sendPaidMedia(*chat_id*, *star_count*, *media*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, *disable_notification=None*, *protect_content=None*, *reply_parameters=None*, *reply_markup=None*, *business_connection_id=None*, *payload=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_thread_id=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *send_paid_media()*

async sendPhoto(*chat_id*, *photo*, *caption=None*, *disable_notification=None*, *reply_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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_markup=None*, *explanation=None*, *explanation_parse_mode=None*, *open_period=None*, *close_date=None*, *explanation_entities=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *question_parse_mode=None*, *question_entities=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *allows_revoting=None*, *allow_adding_options=None*, *hide_results_until_closes=None*, *correct_option_ids=None*, *description=None*, *description_parse_mode=None*, *description_entities=None*, *shuffle_options=None*, *members_only=None*, *country_codes=None*, *explanation_media=None*, *media=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=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_markup=None*, *protect_content=None*, *message_thread_id=None*, *emoji=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *read_timeout=None*, *write_timeout=None*, *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_markup=None*, *foursquare_type=None*, *google_place_id=None*, *google_place_type=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=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_markup=None*, *width=None*, *height=None*, *parse_mode=None*, *supports_streaming=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *cover=None*, *start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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_markup=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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*, *caption=None*, *parse_mode=None*, *disable_notification=None*, *reply_markup=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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

thumbnail will be ignored for small files, for which Telegram can easily generate thumbnails. However, this behaviour is undocumented and might be changed by Telegram.

See also

Working with Files and Media

Shortcuts

- *telegram.Chat.send_animation()*
- *telegram.ChatFullInfo.send_animation()*
- *telegram.Message.reply_animation()*
- *telegram.User.send_animation()*

Changed in version 20.5: Removed deprecated argument `thumb`. Use `thumbnail` instead.

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 | *InputFile* | `bytes` | `pathlib.Path` | *telegram.Animation*) – Animation 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`), the file contents as bytes or the path of the file (as string or `pathlib.Path` object). In the latter case, the file contents will either be read as bytes or the file path will be passed to Telegram, depending on the *local_mode* setting. Lastly you can pass an existing *telegram.Animation* object to send.

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

- ***duration*** (`int` | `datetime.timedelta`, optional) – Duration of sent animation in seconds.

  Changed in version 21.11: `datetime.timedelta` objects are accepted in addition to plain `int` values.

- ***width*** (`int`, optional) – Animation width.
- ***height*** (`int`, optional) – Animation height.
- ***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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of `parse_mode`.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list.

- ***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.

  Added in version 13.10.

- ***message_thread_id*** (`int`, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.

- ***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.
- ***has_spoiler*** (`bool`, optional) – Pass `True` if the animation needs to be covered with a spoiler animation.

  Added in version 20.0.

- ***thumbnail*** (file object | `bytes` | `pathlib.Path` | `str`, 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`), the file contents as bytes or the path of the file (as string or `pathlib.Path` object). In the latter case, the file contents will either be read as bytes or the file path will be passed to Telegram, depending on the *local_mode* setting.

  Added in version 20.2.

- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.

  Added in version 20.8.

- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.1.

- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; for private chats only.

  Added in version 21.3.

- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.

- ***show_caption_above_media*** (bool, optional) – Pass True, if the caption must be shown above the message media.

  Added in version 21.3.

- ***suggested_post_parameters*** (*telegram.SuggestedPostParameters*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only. If the message is sent as a reply to another suggested post, then that suggested post is automatically declined.

  Added in version 22.4.

- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.

  Added in version 22.4.

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***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.

  Added 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*. By default, 20 seconds are used as write timeout.

  Changed in version 22.0: The default value changed 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. See *do_api_request()* for limitations.

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_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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.

See also

Working with Files and Media

Shortcuts

- *telegram.Chat.send_audio()*
- *telegram.ChatFullInfo.send_audio()*
- *telegram.Message.reply_audio()*
- *telegram.User.send_audio()*

Changed in version 20.5: Removed deprecated argument thumb. Use thumbnail instead.

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 | *InputFile* | bytes | pathlib.Path | *telegram.Audio*) – Audio file to send. Pass a file_id as String to send a file that exists on the Telegram

servers (recommended), pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`), the file contents as bytes or the path of the file (as string or `pathlib.Path` object). In the latter case, the file contents will either be read as bytes or the file path will be passed to Telegram, depending on the *local_mode* setting. Lastly you can pass an existing *telegram.Audio* object to send.

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

Changed in version 20.0: File paths as input is also accepted for bots *not* running in *local_mode*.

- ***caption*** (`str`, optional) – Audio caption, 0-*1024* characters after entities parsing.
- ***parse_mode*** (`str`, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of `parse_mode`.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list.
- ***duration*** (`int` | `datetime.timedelta`, optional) – Duration of sent audio in seconds.

  Changed in version 21.11: `datetime.timedelta` objects are accepted in addition to plain `int` values.
- ***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.

  Added in version 13.10.
- ***message_thread_id*** (`int`, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.
- ***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.
- ***thumbnail*** (file object | `bytes` | `pathlib.Path` | `str`, 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`), the file contents as bytes or the path of the file (as string or `pathlib.Path` object). In the latter case, the file contents will either be read as bytes or the file path will be passed to Telegram, depending on the *local_mode* setting.

  Added in version 20.2.
- ***reply_parameters*** (*ReplyParameters*, optional) – Description of the message to reply to.

Added in version 20.8.

- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.1.

- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; for private chats only.

  Added in version 21.3.

- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.

- ***suggested_post_parameters*** (*telegram.SuggestedPostParameters*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only. If the message is sent as a reply to another suggested post, then that suggested post is automatically declined.

  Added in version 22.4.

- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.

  Added in version 22.4.

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***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.

  Added 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*. By default, 20 seconds are used as write timeout.

  Changed in version 22.0: The default value changed 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. See *do_api_request()* for limitations.

Returns
On success, the sent Message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async send_chat_action(*chat_id*, *action*, *message_thread_id=None*, *business_connection_id=None*, ***, *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.

Shortcuts

- *telegram.Chat.send_action()*
- *telegram.Chat.send_chat_action()*
- *telegram.ChatFullInfo.send_action()*
- *telegram.ChatFullInfo.send_chat_action()*
- *telegram.Message.reply_chat_action()*
- *telegram.User.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*.
- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.
- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.1.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async send_checklist(*business_connection_id*, *chat_id*, *checklist*, *disable_notification=None*, *protect_content=None*, *message_effect_id=None*, *reply_parameters=None*, *reply_markup=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send a checklist on behalf of a connected business account.

Shortcuts

- *telegram.Chat.send_checklist()*
- *telegram.ChatFullInfo.send_checklist()*
- *telegram.Message.reply_checklist()*

Added in version 22.3.

Parameters

- ***business_connection_id*** (str) – Unique identifier of the business connection on behalf of which the message will be sent.
- ***chat_id*** (int) – Unique identifier for the target chat or username of the target bot in the format `@username`.
- ***checklist*** (*telegram.InputChecklist*) – The checklist to send.
- ***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.
- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; for private chats only.
- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – An object for an inline keyboard

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for
- ***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. See *do_api_request()* for limitations.

Returns
On success, the sent Message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async **send_contact**(*chat_id*, *phone_number=None*, *first_name=None*, *last_name=None*, *disable_notification=None*, *reply_markup=None*, *vcard=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=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*.

Shortcuts

- *telegram.Chat.send_contact()*
- *telegram.ChatFullInfo.send_contact()*
- *telegram.Message.reply_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.

  Added in version 13.10.
- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.
- ***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.
- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.

  Added in version 20.8.
- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.1.
- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; for private chats only.

  Added in version 21.3.
- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.
- ***suggested_post_parameters*** (*telegram.SuggestedPostParameters*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only. If the message is sent as a reply to another suggested post, then that suggested post is automatically declined.

  Added in version 22.4.
- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.

  Added in version 22.4.

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***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. See *do_api_request()* for limitations.

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_markup=None*, *emoji=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=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.

Shortcuts

- *telegram.Chat.send_dice()*
- *telegram.ChatFullInfo.send_dice()*
- *telegram.Message.reply_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_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.
- ***protect_content*** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  Added in version 13.10.
- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.
- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.

  Added in version 20.8.
- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.1.
- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; for private chats only.

  Added in version 21.3.
- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.
- ***suggested_post_parameters*** (*telegram.SuggestedPostParameters*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only. If the message is sent as a reply to another suggested post, then that suggested post is automatically declined.

  Added in version 22.4.
- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.

  Added in version 22.4.

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.
- ***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. See *do_api_request()* for limitations.

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_markup=None*, *parse_mode=None*, *disable_content_type_detection=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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.

See also

Working with Files and Media

Shortcuts

- *telegram.Chat.send_document()*
- *telegram.ChatFullInfo.send_document()*
- *telegram.Message.reply_document()*
- *telegram.User.send_document()*

Changed in version 20.5: Removed deprecated argument `thumb`. Use `thumbnail` instead.

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 | *InputFile* | `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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`), the file contents as bytes or the path of the file (as string or `pathlib.Path` object). In the latter case, the file contents will either be read as bytes or the file path will be passed to Telegram, depending on the *local_mode* setting. Lastly you can pass an existing *telegram.Document* object to send.

  Note

  Sending by URL will currently only work `GIF`, `PDF` & `ZIP` files.

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

  Changed in version 20.0: File paths as input is also accepted for bots *not* running in *local_mode*.

- ***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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of `parse_mode`.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list.

- ***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.

  Added in version 13.10.

- ***message_thread_id*** (`int`, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.

- ***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.
- ***thumbnail*** (file object | `bytes` | `pathlib.Path` | `str`, 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`), the file contents as bytes or the path of the file (as string or `pathlib.Path` object). In the latter case, the file contents will either be read as bytes or the file path will be passed to Telegram, depending on the *`local_mode`* setting.

Added in version 20.2.

- ***`reply_parameters`*** (*`telegram.ReplyParameters`*, optional) – Description of the message to reply to.

  Added in version 20.8.

- ***`business_connection_id`*** (`str`, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.1.

- ***`message_effect_id`*** (`str`, optional) – Unique identifier of the message effect to be added to the message; for private chats only.

  Added in version 21.3.

- ***`allow_paid_broadcast`*** (`bool`, optional) – Pass True to allow up to *`1000`* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.

- ***`suggested_post_parameters`*** (*`telegram.SuggestedPostParameters`*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only. If the message is sent as a reply to another suggested post, then that suggested post is automatically declined.

  Added in version 22.4.

- ***`direct_messages_topic_id`*** (`int`, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.

  Added in version 22.4.

Keyword Arguments

- ***`allow_sending_without_reply`*** (`bool`, optional) – Pass `True`, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *`reply_parameters`*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *`reply_parameters`* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***`reply_to_message_id`*** (`int`, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *`reply_parameters`*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *`reply_parameters`* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***`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*. By default, 20 seconds are used as write timeout.

  Changed in version 22.0: The default value changed 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. See *do_api_request()* for limitations.

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_markup=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send a game.

Shortcuts

- *telegram.Chat.send_game()*
- *telegram.ChatFullInfo.send_game()*
- *telegram.Message.reply_game()*
- *telegram.User.send_game()*

Parameters

- ***chat_id*** (int) – Unique identifier for the target chat or username of the target bot in the format @username. Games can't be sent to channel direct messages chats and channel chats.
- ***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.

  Added in version 13.10.
- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

Added in version 20.0.

- ***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.
- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.

  Added in version 20.8.
- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.1.
- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; for private chats only.

  Added in version 21.3.
- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.
- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.
- ***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. See *do_api_request()* for limitations.

Returns

On success, the sent Message is returned.

Return type

telegram.Message

Raises

telegram.error.TelegramError –

async **send_gift**(*gift_id*, *text=None*, *text_parse_mode=None*, *text_entities=None*, *pay_for_upgrade=None*, *chat_id=None*, *user_id=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Sends a gift to the given user or channel chat. The gift can't be converted to Telegram Stars by the receiver.

Shortcuts

- *telegram.Chat.send_gift()*
- *telegram.ChatFullInfo.send_gift()*
- *telegram.User.send_gift()*

Added in version 21.8.

Changed in version 22.1: Bot API 8.3 made *user_id* optional. In version 22.1, the methods signature was changed accordingly.

Parameters

- ***gift_id*** (`str` | *Gift*) – Identifier of the gift or a *Gift* object; limited gifts can't be sent to channel chats
- ***user_id*** (`int`, optional) – Required if *chat_id* is not specified. Unique identifier of the target user that will receive the gift.

  Changed in version 21.11: Now optional.
- ***chat_id*** (`int` | `str`, optional) – Required if *user_id* is not specified. Unique identifier for the target chat or username of the target channel (in the format `@channelusername`). It will receive the gift.

  Added in version 21.11.
- ***text*** (`str`, optional) – Text that will be shown along with the gift; 0- *128* characters
- ***text_parse_mode*** (`str`, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details. Entities other than *BOLD*, *ITALIC*, *UNDERLINE*, *STRIKETHROUGH*, *SPOILER*, *CUSTOM_EMOJI*, and *DATE_TIME* are ignored.
- ***text_entities*** (Sequence[*telegram.MessageEntity*], optional) – A list of special entities that appear in the gift text. It can be specified instead of *text_parse_mode*. Entities other than *BOLD*, *ITALIC*, *UNDERLINE*, *STRIKETHROUGH*, *SPOILER*, *CUSTOM_EMOJI*, and *DATE_TIME* are ignored.
- ***pay_for_upgrade*** (`bool`, optional) – Pass `True` to pay for the gift upgrade from the bot's balance, thereby making the upgrade free for the receiver.

  Added in version 21.10.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async send_invoice(*chat_id*, *title*, *description*, *payload*, *currency*, *prices*, *provider_token=None*, *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_markup=None*, *provider_data=None*, *send_phone_number_to_provider=None*, *send_email_to_provider=None*, *max_tip_amount=None*, *suggested_tip_amounts=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=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.

Shortcuts

- *telegram.Chat.send_invoice()*
- *telegram.ChatFullInfo.send_invoice()*
- *telegram.Message.reply_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 it for your internal processes.

- ***provider_token*** (str, optional) – Payments provider token, obtained via @BotFather. Pass an empty string for payments in Telegram Stars.

  Changed in version 21.11: Bot API 7.4 made this parameter is optional and this is now reflected in the function signature.

- ***currency*** (str) – Three-letter ISO 4217 currency code, see more on currencies. Pass XTR for payment in Telegram Stars.

- ***prices*** (Sequence[*telegram.LabeledPrice*]) – Price breakdown, a sequence of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.). Must contain exactly one item for payment in Telegram Stars.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list.

- ***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. Not supported for payment in Telegram Stars.

  Added in version 13.5.

- ***suggested_tip_amounts*** (Sequence[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*.

  Added in version 13.5.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list.

- ***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. Ignored for payments in Telegram Stars.

- ***need_phone_number*** (bool, optional) – Pass True, if you require the user's phone number to complete the order. Ignored for payments in Telegram Stars.

- ***need_email*** (bool, optional) – Pass True, if you require the user's email to complete the order. Ignored for payments in Telegram Stars.

- ***need_shipping_address*** (bool, optional) – Pass True, if you require the user's shipping address to complete the order. Ignored for payments in Telegram Stars.
- ***send_phone_number_to_provider*** (bool, optional) – Pass True, if user's phone number should be sent to provider. Ignored for payments in Telegram Stars.
- ***send_email_to_provider*** (bool, optional) – Pass True, if user's email address should be sent to provider. Ignored for payments in Telegram Stars.
- ***is_flexible*** (bool, optional) – Pass True, if the final price depends on the shipping method. Ignored for payments in Telegram Stars.
- ***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.

  Added in version 13.10.
- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.
- ***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.
- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.

  Added in version 20.8.
- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; for private chats only.

  Added in version 21.3.
- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.
- ***suggested_post_parameters*** (*telegram.SuggestedPostParameters*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only. If the message is sent as a reply to another suggested post, then that suggested post is automatically declined.

  Added in version 22.4.
- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.

  Added in version 22.4.

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.
- ***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. See *do_api_request()* for limitations.

Returns
On success, the sent Message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async send_live_photo(*chat_id*, *live_photo*, *photo*, *business_connection_id=None*, *message_thread_id=None*, *direct_messages_topic_id=None*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, *has_spoiler=None*, *disable_notification=None*, *protect_content=None*, *allow_paid_broadcast=None*, *message_effect_id=None*, *suggested_post_parameters=None*, *reply_parameters=None*, *reply_markup=None*, *, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send live photos.

See also

Working with Files and Media

Shortcuts

- *telegram.Chat.send_live_photo()*
- *telegram.ChatFullInfo.send_live_photo()*
- *telegram.Message.reply_live_photo()*
- *telegram.User.send_live_photo()*

Added in version 22.8.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- ***live_photo*** (str | file object | *InputFile* | bytes | pathlib.Path | *telegram.LivePhoto*) – Live photo video to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended). To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or pathlib.Path object) is supported as well. Sending live photos by a URL is currently unsupported. Lastly you can pass an existing *telegram.LivePhoto* object to send.

  Caution

  - The video must be at most 10MB in size.
  - The video duration must not exceed 10 seconds.
  - If you pass a *telegram.LivePhoto*, its

  photo field will not be considered, use *photo* to specify the photo to send.

- ***photo*** (str | file object | *InputFile* | bytes | pathlib.Path | *telegram.PhotoSize*) – The static photo to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended) . To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or pathlib.Path object) is supported as well. Sending live photos by a URL is currently unsupported. Lastly you can pass an existing *telegram.PhotoSize* object to send.
- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be sent.
- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.
- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.
- ***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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of `parse_mode`.
- ***show_caption_above_media*** (bool, optional) – Pass `True`, if the caption must be shown above the message media.
- ***has_spoiler*** (bool, optional) – Pass `True` if the video needs to be covered with a spoiler animation.
- ***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.

- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.
- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; for private chats only.
- ***suggested_post_parameters*** (*telegram.SuggestedPostParameters*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only. If the message is sent as a reply to another suggested post, then that suggested post is automatically declined.
- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.
- ***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

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for
- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for.
- ***filename*** (str, optional) – Custom file name for *photo*, 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 *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. See *do_api_request()* for limitations.

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_markup=None*, *live_period=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *, *allow_sending_without_reply=None*, *reply_to_message_id=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*.

Shortcuts

- *telegram.Chat.send_location()*
- *telegram.ChatFullInfo.send_location()*
- *telegram.Message.reply_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 | datetime.timedelta, optional) – Period in seconds for which the location will be updated, should be between *60* and *86400*, or *2147483647* for live locations that can be edited indefinitely.

  Changed in version 21.11: datetime.timedelta objects are accepted in addition to plain int values.
- ***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 *100000* 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.

  Added in version 13.10.

- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.

- ***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.

- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.

  Added in version 20.8.

- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.1.

- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; for private chats only.

  Added in version 21.3.

- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.

- ***suggested_post_parameters*** (*telegram.SuggestedPostParameters*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only. If the message is sent as a reply to another suggested post, then that suggested post is automatically declined.

  Added in version 22.4.

- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.

  Added in version 22.4.

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***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. See *do_api_request()* for limitations.

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*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*, *caption=None*, *parse_mode=None*, *caption_entities=None*)

Use this method to send a group of photos, videos, documents or audios as an album. Documents and audio files can be only grouped in an album with messages of the same type.

Note

If you supply a *caption* (along with either *parse_mode* or *caption_entities*), then items in *media* must have no captions, and vice versa.

See also

Working with Files and Media

Shortcuts

- *telegram.Chat.send_media_group()*
- *telegram.ChatFullInfo.send_media_group()*
- *telegram.Message.reply_media_group()*
- *telegram.User.send_media_group()*

Changed in version 20.0: Returns a tuple instead of a list.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- ***media*** (Sequence[*telegram.InputMediaAudio*, *telegram.InputMediaDocument*, *telegram.InputMediaPhoto*, *telegram.InputMediaVideo*, *telegram.InputMediaLivePhoto*]) – An array describing messages to be sent, must include *2*- *10* items.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list.
- ***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.

  Added in version 13.10.
- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.
- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.

  Added in version 20.8.
- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.1.
- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; for private chats only.

  Added in version 21.3.
- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.
- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the messages will be sent; required if the messages are sent to a direct messages chat.

  Added in version 22.4.

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.
- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

Changed in version 21.0: This argument is now a keyword-only argument.

- ***caption*** (str, optional) – Caption that will be added to the first element of *media*, so that it will be used as caption for the whole media group. Defaults to None.

  Added in version 20.0.

- ***parse_mode*** (str | None, optional) – Parse mode for *caption*. See the constants in *telegram.constants.ParseMode* for the available modes.

  Added in version 20.0.

- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – List of special entities for *caption*, which can be specified instead of *parse_mode*. Defaults to None.

  Added in version 20.0.

- ***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*. By default, 20 seconds are used as write timeout.

  Changed in version 22.0: The default value changed 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. See *do_api_request()* for limitations.

Returns
An array of the sent Messages.

Return type
tuple[*telegram.Message*]

Raises
telegram.error.TelegramError –

async send_message(*chat_id*, *text*, *parse_mode=None*, *entities=None*, *disable_notification=None*, *protect_content=None*, *reply_markup=None*, *message_thread_id=None*, *link_preview_options=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *disable_web_page_preview=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send text messages.

Shortcuts

- *telegram.Chat.send_message()*
- *telegram.ChatFullInfo.send_message()*

- *telegram.Message.reply_html()*
- *telegram.Message.reply_markdown_v2()*
- *telegram.Message.reply_markdown()*
- *telegram.Message.reply_text()*
- *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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in message text, which can be specified instead of *parse_mode*.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list.
- ***link_preview_options*** (*LinkPreviewOptions*, optional) – Link preview generation options for the message. Mutually exclusive with *disable_web_page_preview*.

  Added in version 20.8.
- ***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.

  Added in version 13.10.
- ***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.
- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.
- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.

  Added in version 20.8.
- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.1.
- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; for private chats only.

  Added in version 21.3.
- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

Added in version 21.7.

- ***suggested_post_parameters*** (*telegram.SuggestedPostParameters*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only. If the message is sent as a reply to another suggested post, then that suggested post is automatically declined.

  Added in version 22.4.

- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.

  Added in version 22.4.

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***disable_web_page_preview*** (bool, optional) – Disables link previews for links in this message. Convenience parameter for setting *link_preview_options*. Mutually exclusive with *link_preview_options*.

  Changed in version 20.8: Bot API 7.0 introduced *link_preview_options* replacing this argument. PTB will automatically convert this argument to that one, but for advanced options, please use *link_preview_options* directly.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***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. See *do_api_request()* for limitations.

Returns

On success, the sent message is returned.

Return type

telegram.Message

Raises

- **ValueError** – If both *disable_web_page_preview* and *link_preview_options* are passed.
- ***telegram.error.TelegramError*** – For other errors.

async send_message_draft(*chat_id*, *draft_id*, *text=None*, *message_thread_id=None*, *parse_mode=None*, *entities=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to stream a partial message to a user while the message is being generated. Note that the streamed draft is ephemeral and acts as a temporary 30-second preview - once the output is finalized, you must call *send_message()* with the complete message to persist it in the user's chat.

Shortcuts

- *telegram.Chat.send_message_draft()*
- *telegram.ChatFullInfo.send_message_draft()*
- *telegram.Message.reply_text_draft()*
- *telegram.User.send_message_draft()*

Added in version 22.6.

Changed in version 22.7: Now all bots can use this method.

Parameters

- ***chat_id*** (int) – Unique identifier for the target private chat.
- ***draft_id*** (int) – Unique identifier of the message draft; must be non-zero. Changes of drafts with the same identifier are animated.
- ***text*** (str, optional) – Text of the message to be sent, 0-*4096* characters after entities parsing. Pass an empty text to show a "Thinking…" placeholder.

  Changed in version 22.8: Bot API 10.0 now makes this an optional parameter.
- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread.
- ***parse_mode*** (str) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in message text, which can be specified instead of *parse_mode*.

  Accepts any collections.abc.Sequence as input instead of just a list.

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. See *do_api_request()* for limitations.

Returns
On success, `True` is returned.

Return type
`bool`

Raises
telegram.error.TelegramError –

async **send_paid_media**(*chat_id*, *star_count*, *media*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, *disable_notification=None*, *protect_content=None*, *reply_parameters=None*, *reply_markup=None*, *business_connection_id=None*, *payload=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_thread_id=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send paid media.

Shortcuts

- *telegram.Chat.send_paid_media()*
- *telegram.ChatFullInfo.send_paid_media()*
- *telegram.Message.reply_paid_media()*

Added in version 21.4.

Parameters

- ***chat_id*** (`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`). If the chat is a channel, all Telegram Star proceeds from this media will be credited to the chat's balance. Otherwise, they will be credited to the bot's balance.
- ***star_count*** (`int`) – The number of Telegram Stars that must be paid to buy access to the media; *1* - *25000*.
- ***media*** (Sequence[*telegram.InputPaidMedia*]) – A list describing the media to be sent; up to *10* items.
- ***payload*** (`str`, optional) – Bot-defined paid media payload, 0-*128* bytes. This will not be displayed to the user, use it for your internal processes.

  Added in version 21.6.
- ***caption*** (`str`, optional) – Caption of the media to be sent, 0-*1024* characters.
- ***parse_mode*** (`str`, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of `parse_mode`.
- ***show_caption_above_media*** (`bool`, optional) – Pass `True`, if the caption must be shown above the message media.
- ***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.

- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.
- ***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.
- ***business_connection_id*** (`str`, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.5.
- ***allow_paid_broadcast*** (`bool`, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.
- ***suggested_post_parameters*** (*telegram.SuggestedPostParameters*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only. If the message is sent as a reply to another suggested post, then that suggested post is automatically declined.

  Added in version 22.4.
- ***direct_messages_topic_id*** (`int`, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.

  Added in version 22.4.
- ***message_thread_id*** (`int`, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 22.4.

Keyword Arguments

- ***allow_sending_without_reply*** (`bool`, optional) – Pass `True`, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for
- ***reply_to_message_id*** (`int`, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for
- ***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. See *do_api_request()* for limitations.

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_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send photos.

See also

Working with Files and Media

Shortcuts

- *telegram.Chat.send_photo()*
- *telegram.ChatFullInfo.send_photo()*
- *telegram.Message.reply_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 | *InputFile* | `bytes` | `pathlib.Path` | *telegram.PhotoSize*) – Photo 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`), the file contents as bytes or the path of the file (as string or `pathlib.Path` object). In the latter case, the file contents will either be read as bytes or the file path will be passed to Telegram, depending on the *local_mode* setting. Lastly you can pass an existing *telegram.PhotoSize* object to send.

  Caution

  - The photo must be at most 10MB in size.
  - The photo's width and height must not exceed 10000 in total.
  - Width and height ratio must be at most 20.

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

  Changed in version 20.0: File paths as input is also accepted for bots *not* running in *local_mode*.

- ***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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of parse_mode.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list.
- ***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.

  Added in version 13.10.
- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.
- ***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.
- ***has_spoiler*** (bool, optional) – Pass True if the photo needs to be covered with a spoiler animation.

  Added in version 20.0.
- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.

  Added in version 20.8.
- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.1.
- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; for private chats only.

  Added in version 21.3.
- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.
- ***show_caption_above_media*** (bool, optional) – Pass True, if the caption must be shown above the message media.

  Added in version 21.3.
- ***suggested_post_parameters*** (*telegram.SuggestedPostParameters*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only. If the message is sent as a reply to another suggested post, then that suggested post is automatically declined.

  Added in version 22.4.
- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.

Added in version 22.4.

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***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.

  Added 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*. By default, 20 seconds are used as write timeout.

  Changed in version 22.0: The default value changed 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. See *do_api_request()* for limitations.

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_markup=None*, *explanation=None*, *explanation_parse_mode=None*, *open_period=None*, *close_date=None*, *explanation_entities=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *question_parse_mode=None*, *question_entities=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *allows_revoting=None*, *allow_adding_options=None*, *hide_results_until_closes=None*, *correct_option_ids=None*, *description=None*, *description_parse_mode=None*, *description_entities=None*, *shuffle_options=None*, *members_only=None*, *country_codes=None*, *explanation_media=None*, *media=None*, *, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to send a native poll.

Shortcuts

- *telegram.Chat.send_poll()*
- *telegram.ChatFullInfo.send_poll()*
- *telegram.Message.reply_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*** (Sequence[str | *telegram.InputPollOption*]) – Sequence of *1*- *12* answer options. Each option may either be a string with *1*- *100* characters or an *InputPollOption* object. Strings are converted to *InputPollOption* objects automatically.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list.

  Changed in version 21.2: Bot API 7.3 adds support for *InputPollOption* objects.
- ***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, defaults to `False`.
- ***correct_option_id*** (int, optional) – 0-based identifier of the correct answer option, required for polls in quiz mode.

  Deprecated since version 22.8: Bot API 9.6 replaces this with *correct_option_ids* instead.
- ***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*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in message text, which can be specified instead of *explanation_parse_mode*.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list.

- ***open_period*** (`int` | `datetime.timedelta`, optional) – Amount of time in seconds the poll will be active after creation, *5*- *2628000*. Can't be used together with *close_date*.

  Changed in version 21.11: `datetime.timedelta` objects are accepted in addition to plain `int` values.

- ***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 *2628000* 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.

  Added in version 13.10.

- ***message_thread_id*** (`int`, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.

- ***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.

- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.

  Added in version 20.8.

- ***business_connection_id*** (`str`, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.1.

- ***question_parse_mode*** (`str`, optional) – Mode for parsing entities in the question. See the constants in *telegram.constants.ParseMode* for the available modes. Currently, only custom emoji entities are allowed.

  Added in version 21.2.

- ***question_entities*** (Sequence[*telegram.Message*], optional) – Special entities that appear in the poll *question*. It can be specified instead of *question_parse_mode*.

  Added in version 21.2.

- ***message_effect_id*** (`str`, optional) – Unique identifier of the message effect to be added to the message; for private chats only.

  Added in version 21.3.

- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.

- ***allows_revoting*** (bool, optional) – True, if the poll allows to change the chosen answer options, defaults to False for quizzes and to True for regular polls

  Added in version 22.8.

- ***allow_adding_options*** (bool, optional) – True, if answer options can be added to the poll after creation; not supported for anonymous polls and quizzes

  Added in version 22.8.

- ***hide_results_until_closes*** (bool, optional) – True, if poll results must be shown only after the poll closes

  Added in version 22.8.

- ***correct_option_ids*** (Sequence[int], optional) – A list of monotonically increasing 0-based identifiers of the correct answer options, required for polls in quiz mode.

  Added in version 22.8.

- ***description*** (str, optional) – Description of the poll to be sent, 0-*1024* characters after entities parsing.

  Added in version 22.8.

- ***description_parse_mode*** (str, optional) – Mode for parsing entities in the poll description. See the constants in *telegram.constants.ParseMode*

  Added in version 22.8.

- ***description_entities*** (Sequence[*telegram.MessageEntity*], optional) – A JSON-serialized list of special entities that appear in the poll description, which can be specified instead of *description_parse_mode*

  Added in version 22.8.

- ***shuffle_options*** (bool, optional) – True, if the poll options must be shown in random order

  Added in version 22.8.

- ***members_only*** (bool, optional) – True, if voting is limited to users who have been members of the chat where the poll is being sent for more than *24* hours; for channel chats only

  Added in version 22.8.

- ***country_codes*** (Sequence[str], optional) – A list of 0-*12* two-letter ISO 3166-1 alpha-2 country codes indicating the countries from which users can vote in the poll; for channel chats only. Use "FT" as a country code to allow users with anonymous numbers to vote. If omitted or empty, then users from any country can participate in the poll.

  Added in version 22.8.

- ***explanation_media*** (*telegram.InputPollMedia*, optional) – Media added to the quiz explanation

  Added in version 22.8.

- ***media*** (*telegram.InputPollMedia*, optional) – Media added to the poll description.

  Added in version 22.8.

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***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. See *do_api_request()* for limitations.

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_markup=None*, *protect_content=None*, *message_thread_id=None*, *emoji=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

See also

Working with Files and Media

Shortcuts

- *telegram.Chat.send_sticker()*
- *telegram.ChatFullInfo.send_sticker()*
- *telegram.Message.reply_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 | *InputFile* | 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 file from the Internet, or upload a new one. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`), the file contents as bytes or the path of the file (as string or `pathlib.Path` object). In the latter case, the file contents will either be read as bytes or the file path will be passed to Telegram, depending on the *local_mode* setting. Video stickers can only be sent by a `file_id`. Video and animated stickers can't be sent via an HTTP URL.

  Lastly you can pass an existing *telegram.Sticker* object to send.

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

  Changed in version 20.0: File paths as input is also accepted for bots *not* running in *local_mode*.
- ***emoji*** (str, optional) – Emoji associated with the sticker; only for just uploaded stickers

  Added in version 20.2.
- ***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.

  Added in version 13.10.
- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.
- ***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.
- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.

  Added in version 20.8.
- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.1.

- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; for private chats only.

  Added in version 21.3.

- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.

- ***suggested_post_parameters*** (*telegram.SuggestedPostParameters*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only. If the message is sent as a reply to another suggested post, then that suggested post is automatically declined.

  Added in version 22.4.

- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.

  Added in version 22.4.

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***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*. By default, 20 seconds are used as write timeout.

  Changed in version 22.0: The default value changed 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. See *do_api_request()* for limitations.

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_markup=None*, *foursquare_type=None*, *google_place_id=None*, *google_place_type=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=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.

Shortcuts

- *telegram.Chat.send_venue()*
- *telegram.ChatFullInfo.send_venue()*
- *telegram.Message.reply_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.

  Added in version 13.10.

- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.

- ***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.

- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.

  Added in version 20.8.

- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.1.

- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; for private chats only.

  Added in version 21.3.

- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.

- ***suggested_post_parameters*** (*telegram.SuggestedPostParameters*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only. If the message is sent as a reply to another suggested post, then that suggested post is automatically declined.

  Added in version 22.4.

- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.

  Added in version 22.4.

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

Changed in version 21.0: This argument is now a keyword-only argument.

- ***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. See *do_api_request()* for limitations.

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_markup=None*, *width=None*, *height=None*, *parse_mode=None*, *supports_streaming=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *cover=None*, *start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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

thumbnail 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

Working with Files and Media

Shortcuts

- *telegram.Chat.send_video()*
- *telegram.ChatFullInfo.send_video()*

- *telegram.Message.reply_video()*
- *telegram.User.send_video()*

Changed in version 20.5: Removed deprecated argument `thumb`. Use `thumbnail` instead.

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 | *InputFile* | `bytes` | `pathlib.Path` | *telegram.Video*) – Video 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`), the file contents as bytes or the path of the file (as string or `pathlib.Path` object). In the latter case, the file contents will either be read as bytes or the file path will be passed to Telegram, depending on the *local_mode* setting. Lastly you can pass an existing *telegram.Video* object to send.

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

  Changed in version 20.0: File paths as input is also accepted for bots *not* running in *local_mode*.
- ***duration*** (`int` | `datetime.timedelta`, optional) – Duration of sent video in seconds.

  Changed in version 21.11: `datetime.timedelta` objects are accepted in addition to plain `int` values.
- ***width*** (`int`, optional) – Video width.
- ***height*** (`int`, optional) – Video height.
- ***cover*** (file object | `bytes` | `pathlib.Path` | `str`, optional) – Cover for the video in the message. 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well.

  Added in version 21.11.
- ***start_timestamp*** (`int`, optional) – Start timestamp for the video in the message.

  Added in version 21.11.
- ***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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of `parse_mode`.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list.
- ***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.

  Added in version 13.10.

- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.

- ***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.

- ***has_spoiler*** (bool, optional) – Pass True if the video needs to be covered with a spoiler animation.

  Added in version 20.0.

- ***thumbnail*** (file object | bytes | pathlib.Path | str, 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`), the file contents as bytes or the path of the file (as string or pathlib.Path object). In the latter case, the file contents will either be read as bytes or the file path will be passed to Telegram, depending on the *local_mode* setting.

  Added in version 20.2.

- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.

  Added in version 20.8.

- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.1.

- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; for private chats only.

  Added in version 21.3.

- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.

- ***show_caption_above_media*** (bool, optional) – Pass True, if the caption must be shown above the message media.

  Added in version 21.3.

- ***suggested_post_parameters*** (*telegram.SuggestedPostParameters*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only. If the message is sent as a reply to another suggested post, then that suggested post is automatically declined.

  Added in version 22.4.

- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.

Added in version 22.4.

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***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.

  Added 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*. By default, 20 seconds are used as write timeout.

  Changed in version 22.0: The default value changed 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. See *do_api_request()* for limitations.

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_markup=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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

thumbnail 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

Working with Files and Media

Shortcuts

- *telegram.Chat.send_video_note()*
- *telegram.ChatFullInfo.send_video_note()*
- *telegram.Message.reply_video_note()*
- *telegram.User.send_video_note()*

Changed in version 20.5: Removed deprecated argument `thumb`. Use `thumbnail` instead.

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 | *InputFile* | `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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`), the file contents as bytes or the path of the file (as string or `pathlib.Path` object). In the latter case, the file contents will either be read as bytes or the file path will be passed to Telegram, depending on the *local_mode* setting. Lastly 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.

  Changed in version 20.0: File paths as input is also accepted for bots *not* running in *local_mode*.
- ***duration*** (`int` | `datetime.timedelta`, optional) – Duration of sent video in seconds.

  Changed in version 21.11: `datetime.timedelta` objects are accepted in addition to plain `int` values.
- ***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.

  Added in version 13.10.

- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

  Added in version 20.0.

- ***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.

- ***thumbnail*** (file object | bytes | pathlib.Path | str, 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`), the file contents as bytes or the path of the file (as string or pathlib.Path object). In the latter case, the file contents will either be read as bytes or the file path will be passed to Telegram, depending on the *local_mode* setting.

  Added in version 20.2.

- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.

  Added in version 20.8.

- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.1.

- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; for private chats only.

  Added in version 21.3.

- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.

- ***suggested_post_parameters*** (*telegram.SuggestedPostParameters*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only. If the message is sent as a reply to another suggested post, then that suggested post is automatically declined.

  Added in version 22.4.

- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.

  Added in version 22.4.

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.

- ***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.

  Added 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*. By default, 20 seconds are used as write timeout.

  Changed in version 22.0: The default value changed 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. See *do_api_request()* for limitations.

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_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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 , or in .MP3 format, or in .M4A format (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

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

See also

Working with Files and Media

Shortcuts

- *telegram.Chat.send_voice()*
- *telegram.ChatFullInfo.send_voice()*
- *telegram.Message.reply_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 | *InputFile* | bytes | pathlib.Path | *telegram.Voice*) – Voice 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`), the file contents as bytes or the path of the file (as string or `pathlib.Path` object). In the latter case, the file contents will either be read as bytes or the file path will be passed to Telegram, depending on the *local_mode* setting. Lastly you can pass an existing *telegram.Voice* object to send.

  Changed in version 13.2: Accept bytes as input.

  Changed in version 20.0: File paths as input is also accepted for bots *not* running in *local_mode*.
- ***caption*** (str, optional) – Voice message caption, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of `parse_mode`.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list.
- ***duration*** (int | datetime.timedelta, optional) – Duration of the voice message in seconds.

  Changed in version 21.11: `datetime.timedelta` objects are accepted in addition to plain int values.
- ***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.

  Added in version 13.10.
- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread (topic) of a forum; for forum supergroups and private chats of bots with forum topic mode enabled only.

Added in version 20.0.

- ***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.
- ***reply_parameters*** (*telegram.ReplyParameters*, optional) – Description of the message to reply to.

  Added in version 20.8.
- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be sent.

  Added in version 21.1.
- ***message_effect_id*** (str, optional) – Unique identifier of the message effect to be added to the message; for private chats only.

  Added in version 21.3.
- ***allow_paid_broadcast*** (bool, optional) – Pass True to allow up to *1000* messages per second, ignoring broadcasting limits for a fee of 0.1 Telegram Stars per message. The relevant Stars will be withdrawn from the bot's balance.

  Added in version 21.7.
- ***suggested_post_parameters*** (*telegram.SuggestedPostParameters*, optional) – An object containing the parameters of the suggested post to send; for direct messages chats only. If the message is sent as a reply to another suggested post, then that suggested post is automatically declined.

  Added in version 22.4.
- ***direct_messages_topic_id*** (int, optional) – Identifier of the direct messages topic to which the message will be sent; required if the message is sent to a direct messages chat.

  Added in version 22.4.

Keyword Arguments

- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.
- ***reply_to_message_id*** (int, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*, which this is a convenience parameter for

  Changed in version 20.8: Bot API 7.0 introduced *reply_parameters* replacing this argument. PTB will automatically convert this argument to that one, but you should update your code to use the new argument.

  Changed in version 21.0: This argument is now a keyword-only argument.
- ***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.

  Added 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*. By default, 20 seconds are used as write timeout.

  Changed in version 22.0: The default value changed 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. See *do_api_request()* for limitations.

Returns
On success, the sent Message is returned.

Return type
telegram.Message

Raises
telegram.error.TelegramError –

async setBusinessAccountBio(*business_connection_id*, *bio=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_business_account_bio()*

async setBusinessAccountGiftSettings(*business_connection_id*, *show_gift_button*, *accepted_gift_types*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_business_account_gift_settings()*

async setBusinessAccountName(*business_connection_id*, *first_name*, *last_name=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_business_account_name()*

async setBusinessAccountProfilePhoto(*business_connection_id*, *photo*, *is_public=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_business_account_profile_photo()*

async setBusinessAccountUsername(*business_connection_id*, *username=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_business_account_username()*

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 setChatMemberTag(*chat_id*, *user_id*, *tag=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_chat_member_tag()*

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*, *use_independent_chat_permissions=None*, ***, *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=None*, *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 setCustomEmojiStickerSetThumbnail(*name*, *custom_emoji_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_custom_emoji_sticker_set_thumbnail()*

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 setManagedBotAccessSettings(*user_id*, *is_access_restricted*, *added_user_ids=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_managed_bot_access_settings()*

async setMessageReaction(*chat_id*, *message_id*, *reaction=None*, *is_big=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_message_reaction()*

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 setMyDescription(*description=None*, *language_code=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_my_description()*

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

Alias for *set_my_name()*

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

Alias for *set_my_profile_photo()*

async setMyShortDescription(*short_description=None*, *language_code=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_my_short_description()*

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 setStickerEmojiList(*sticker*, *emoji_list*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_sticker_emoji_list()*

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

Alias for *set_sticker_keywords()*

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

Alias for *set_sticker_mask_position()*

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 setStickerSetThumbnail(*name*, *user_id*, *format*, *thumbnail=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_sticker_set_thumbnail()*

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

Alias for *set_sticker_set_title()*

async setUserEmojiStatus(*user_id*, *emoji_status_custom_emoji_id=None*, *emoji_status_expiration_date=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *set_user_emoji_status()*

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_business_account_bio(*business_connection_id*, *bio=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Changes the bio of a managed business account. Requires the *can_edit_bio* business bot right.

Added in version 22.1.

Parameters

- ***business_connection_id*** (str) – Unique identifier of the business connection.
- ***bio*** (str, optional) – The new value of the bio for the business account; 0-*140* 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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async set_business_account_gift_settings(*business_connection_id*, *show_gift_button*, *accepted_gift_types*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Changes the privacy settings pertaining to incoming gifts in a managed business account. Requires the *can_change_gift_settings* business bot right.

Added in version 22.1.

Parameters

- ***business_connection_id*** (str) – Unique identifier of the business connection
- ***show_gift_button*** (bool) – Pass True, if a button for sending a gift to the user or by the business account must always be shown in the input field.
- ***accepted_gift_types*** (*telegram.AcceptedGiftTypes*) – Types of gifts accepted by the business account.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async set_business_account_name(*business_connection_id*, *first_name*, *last_name=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Changes the first and last name of a managed business account. Requires the *can_edit_name* business bot right.

Added in version 22.1.

Parameters

- **business_connection_id** (int | str) – Unique identifier of the business connection
- **first_name** (str) – New first name of the business account; *1*- *64* characters.
- **last_name** (str, optional) – New last name of the business account; 0-*64* 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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async set_business_account_profile_photo(*business_connection_id*, *photo*, *is_public=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Changes the profile photo of a managed business account. Requires the *can_edit_profile_photo* business bot right.

Added in version 22.1.

Parameters

- ***business_connection_id*** (str) – Unique identifier of the business connection.
- ***photo*** (*telegram.InputProfilePhoto*) – The new profile photo to set.
- ***is_public*** (bool, optional) – Pass True to set the public photo, which will be visible even if the main photo is hidden by the business account's privacy settings. An account can have only one public photo.

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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

async set_business_account_username(*business_connection_id*, *username=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Changes the username of a managed business account. Requires the *can_edit_username* business bot right.

Added in version 22.1.

Parameters

- ***business_connection_id*** (str) – Unique identifier of the business connection.
- ***username*** (str, optional) – New business account username; 0-*32* 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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

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.

Shortcuts

- *telegram.Chat.set_administrator_custom_title()*
- *telegram.ChatFullInfo.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. See *do_api_request()* for limitations.

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.

Shortcuts

- *telegram.Chat.set_description()*
- *telegram.ChatFullInfo.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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

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

Use this method to set a tag for a regular member in a group or a supergroup. The bot must be an administrator in the chat for this to work and must have the *can_manage_tags* administrator right.

Shortcuts

- *telegram.Chat.set_chat_member_tag()*
- *telegram.ChatFullInfo.set_chat_member_tag()*
- *telegram.User.set_chat_member_tag()*

Added in version 22.7.

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.
- ***tag*** (str, optional) – New tag for the member; 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. See *do_api_request()* for limitations.

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.get_menu_button()* *telegram.User.get_menu_button()*

Shortcuts

- *telegram.Chat.set_menu_button()*
- *telegram.ChatFullInfo.set_menu_button()*
- *telegram.User.set_menu_button()*

Added in version 20.0.

Parameters

- ***chat_id*** (int, optional) – Unique identifier for the target private chat. If not specified, the bot's default 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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async set_chat_permissions(*chat_id*, *permissions*, *use_independent_chat_permissions=None*, ***, *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.

Shortcuts

- *telegram.Chat.set_permissions()*
- *telegram.ChatFullInfo.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.
- ***use_independent_chat_permissions*** (bool, optional) – Pass True if chat permissions are set independently. Otherwise, the *can_send_other_messages* and *can_add_web_page_previews* permissions will imply the *can_send_messages*, *can_send_audios*, *can_send_documents*, *can_send_photos*, *can_send_videos*, *can_send_video_notes*, and *can_send_voice_notes* permissions; the *can_send_polls* permission will imply the *can_send_messages* permission.

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. See *do_api_request()* for limitations.

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=None*, *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.

Shortcuts

- *telegram.Chat.set_photo()*
- *telegram.ChatFullInfo.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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`), the file contents as bytes or the path of the file (as string or pathlib.Path object). In the latter case, the file contents will either be read as bytes or the file path will be passed to Telegram, depending on the *local_mode* setting.

  Changed in version 13.2: Accept bytes as input.

  Changed in version 20.0: File paths as input is also accepted for bots *not* running in *local_mode*.

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*. By default, `20` seconds are used as write timeout.

  Changed in version 22.0: The default value changed 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. See *do_api_request()* for limitations.

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.ChatFullInfo.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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

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.

Shortcuts

- *telegram.Chat.set_title()*
- *telegram.ChatFullInfo.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*- *128* 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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async set_custom_emoji_sticker_set_thumbnail(*name*, *custom_emoji_id=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 custom emoji sticker set.

Added in version 20.2.

Parameters

- ***name*** (str) – Sticker set name.
- ***custom_emoji_id*** (str, optional) – Custom emoji identifier of a sticker from the sticker set; pass an empty string to drop the thumbnail and use the first sticker as the thumbnail.

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. See *do_api_request()* for limitations.

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.

See also

telegram.Game.text

Shortcuts

- *telegram.CallbackQuery.set_game_score()*
- *telegram.Message.set_game_score()*

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, 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. See *do_api_request()* for limitations.

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_managed_bot_access_settings(*user_id*, *is_access_restricted*, *added_user_ids=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to change the access settings of a managed bot.

Shortcuts

telegram.User.set_managed_bot_access_settings()

Added in version 22.8.

Parameters

- ***user_id*** (int) – User identifier of the managed bot whose access settings will be changed.
- ***is_access_restricted*** (bool) – Pass True, if only selected users can access the bot. The bot's owner can always access it.
- ***added_user_ids*** (Sequence[int], optional) – A list of up to *10* identifiers of users who will have access to the bot in addition to its owner. Ignored if *is_access_restricted* is False.

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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

async set_message_reaction(*chat_id*, *message_id*, *reaction=None*, *is_big=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Use this method to change the chosen reactions on a message. Service messages of some types can't be reacted to. Automatically forwarded messages from a channel to its discussion group have the same available reactions as messages in the channel. Bots can't use paid reactions.

Shortcuts

- *telegram.Chat.set_message_reaction()*
- *telegram.ChatFullInfo.set_message_reaction()*
- *telegram.Message.set_reaction()*

Added in version 20.8.

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 target message. If the message belongs to a media group, the reaction is set to the first non-deleted message in the group instead.
- ***reaction*** (Sequence[*telegram.ReactionType* | str] | *telegram.ReactionType* | str, optional) – A list of reaction types to set on the message. Currently, as non-premium users, bots can set up to one reaction per message. A custom emoji reaction can be used if it is either already present on the message or explicitly allowed by chat administrators. Paid reactions can't be used by bots.

> **Tip**
>
> Passed str values will be converted to either *telegram.ReactionTypeEmoji* or *telegram.ReactionTypeCustomEmoji* depending on whether they are listed in *ReactionEmoji*.

- ***is_big*** (bool, optional) – Pass True to set the reaction with a big animation.

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. See *do_api_request()* for limitations.

Returns

bool On success, True is returned.

Raises

telegram.error.TelegramError –

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.

> **See also**
>
> *get_my_commands()*, *delete_my_commands()*

Parameters

- ***commands*** (Sequence[*BotCommand* | (str, str)]) – A sequence of bot commands to be set as the list of the bot's commands. At most *100* commands can be specified.

> **Note**

If you pass in a sequence of `tuple`, the order of elements in each `tuple` must correspond to the order of positional arguments to create a *BotCommand* instance.

Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list.

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

  Added 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.

  Added 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. See *do_api_request()* for limitations.

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 free to modify the list before adding the bot.

See also

get_my_default_administrator_rights()

Added 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. See *do_api_request()* for limitations.

Returns
Returns True on success.

Return type
bool

Raises
telegram.error.TelegramError –

async set_my_description(*description=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 bot's description, which is shown in the chat with the bot if the chat is empty.

Added in version 20.2.

Parameters

- ***description*** (str, optional) – New bot description; 0-*512* characters. Pass an empty string to remove the dedicated description for the given language.
- ***language_code*** (str, optional) – A two-letter ISO 639-1 language code. If empty, the description will be applied to all users for whose language there is no dedicated description.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async **set_my_name**(*name=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 bot's name.

Added in version 20.3.

Parameters

- ***name*** (str, optional) – New bot name; 0-*64* characters. Pass an empty string to remove the dedicated name for the given language.

  Caution

  If *language_code* is not specified, a *name* *must* be specified.

- ***language_code*** (str, optional) – A two-letter ISO 639-1 language code. If empty, the name will be applied to all users for whose language there is no dedicated name.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

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

Changes the profile photo of the bot.

Added in version 22.7.

Parameters
photo (*telegram.InputProfilePhoto*) – The new profile photo to 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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async set_my_short_description(*short_description=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 bot's short description, which is shown on the bot's profile page and is sent together with the link when users share the bot.

Added in version 20.2.

Parameters

- ***short_description*** (str, optional) – New short description for the bot; 0-*120* characters. Pass an empty string to remove the dedicated description for the given language.
- ***language_code*** (str, optional) – A two-letter ISO 639-1 language code. If empty, the description will be applied to all users for whose language there is no dedicated description.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

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*** (Sequence[*PassportElementError*]) – A Sequence describing the errors.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

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

Use this method to change the list of emoji assigned to a regular or custom emoji sticker. The sticker must belong to a sticker set created by the bot.

Added in version 20.2.

Parameters

- ***sticker*** (str | *Sticker*) – File identifier of the sticker or the sticker object.

  Changed in version 21.10: Accepts also *telegram.Sticker* instances.

- ***emoji_list*** (Sequence[str]) – A sequence of *1*- *20* emoji associated with 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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

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

Use this method to change search keywords assigned to a regular or custom emoji sticker. The sticker must belong to a sticker set created by the bot.

Added in version 20.2.

Parameters

- ***sticker*** (str | *Sticker*) – File identifier of the sticker or the sticker object.

  Changed in version 21.10: Accepts also *telegram.Sticker* instances.

- ***keywords*** (Sequence[str]) – A sequence of 0-*20* search keywords for the sticker with total length up to *64* 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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

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

Use this method to change the mask position of a mask sticker. The sticker must belong to a sticker set that was created by the bot.

Added in version 20.2.

Parameters

- ***sticker*** (str | *Sticker*) – File identifier of the sticker or the sticker object.

  Changed in version 21.10: Accepts also *telegram.Sticker* instances.

- ***mask_position*** (*telegram.MaskPosition*, optional) – A object with the position where the mask should be placed on faces. Omit the parameter to remove the mask position.

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. See *do_api_request()* for limitations.

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 | *Sticker*) – File identifier of the sticker or the sticker object.

  Changed in version 21.10: Accepts also *telegram.Sticker* instances.

- ***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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async set_sticker_set_thumbnail(*name*, *user_id*, *format*, *thumbnail=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 regular or mask sticker set. The format of the thumbnail file must match the format of the stickers in the set.

Added in version 20.2.

Changed in version 21.1: As per Bot API 7.2, the new argument *format* will be required, and thus the order of the arguments had to be changed.

Parameters

- ***name*** (str) – Sticker set name
- ***user_id*** (int) – User identifier of created sticker set owner.
- ***format*** (str) – Format of the added sticker, must be one of *'static'* for a .WEBP or .PNG image, *'animated'* for a .TGS animation, *'video'* for a .WEBM video.

  Added in version 21.1.
- ***thumbnail*** (str | file object | *InputFile* | bytes | pathlib.Path, optional) – A **.WEBP** or **.PNG** image with the thumbnail, must be up to *128* kilobytes in size and have width and height of exactly *100* px, or a **.TGS** animation with the thumbnail up to *32* kilobytes in size; see the docs for animated sticker technical requirements, or a .WEBM video with the thumbnail up to *32* kilobytes in size; see this for video sticker technical requirements.

  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. To upload a file, you can either pass a file object (e.g. open("filename", "rb")), the file contents as bytes or the path of the file (as string or pathlib.Path object). In the latter case, the file contents will either be read as bytes or the file path will be passed to Telegram, depending on the *local_mode* setting.

  Animated and video sticker set thumbnails can't be uploaded via HTTP URL. If omitted, then the thumbnail is dropped and the first sticker is used as the thumbnail.

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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

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

Use this method to set the title of a created sticker set.

Added in version 20.2.

Parameters

- ***name*** (str) – Sticker set name.

- ***title*** (str) – Sticker set title, *1- 64* 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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

async set_user_emoji_status(*user_id*, *emoji_status_custom_emoji_id=None*, *emoji_status_expiration_date=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Changes the emoji status for a given user that previously allowed the bot to manage their emoji status via the Mini App method requestEmojiStatusAccess .

Added in version 21.8.

Parameters

- ***user_id*** (int) – Unique identifier of the target user
- ***emoji_status_custom_emoji_id*** (str, optional) – Custom emoji identifier of the emoji status to set. Pass an empty string to remove the status.
- ***emoji_status_expiration_date*** (int | datetime.datetime, optional) – Expiration date of the emoji status, if any, as unix timestamp or datetime.datetime object. 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.

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. See *do_api_request()* for limitations.

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 (a request with response *HTTP status code <https://en.wikipedia.org/wiki/List_of_HTTP_status_codes>`_different from ``2XY``*), Telegram will repeat the request and 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

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.

See also

telegram.ext.Application.run_webhook(), *telegram.ext.Updater.start_webhook()*

Examples

Custom Webhook Bot

Parameters

- ***url*** (str) – HTTPS url to send updates to. Use an empty string to remove webhook integration.
- ***certificate*** (file object | bytes | pathlib.Path | str) – Upload your public key certificate so that the root certificate in use can be checked. See our self-signed guide for details. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or pathlib.Path object) is supported as well.
- ***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*** (Sequence[str], optional) – A sequence of 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 sequence to receive all updates except *telegram.Update.chat_member*, *telegram.Update.message_reaction* and *telegram.Update.message_reaction_count* (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 update may be received for a short period of time.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list.

- ***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.

  Added 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. See *do_api_request()* for limitations.

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()

Added in version 20.0.

async stopMessageLiveLocation(*chat_id=None*, *message_id=None*, *inline_message_id=None*, *reply_markup=None*, *business_connection_id=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*, *business_connection_id=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*, *business_connection_id=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.

Shortcuts

- *telegram.CallbackQuery.stop_message_live_location()*
- *telegram.Message.stop_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 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.
- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message to be edited was sent

  Added in version 21.4.

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. See *do_api_request()* for limitations.

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 stop_poll(*chat_id*, *message_id*, *reply_markup=None*, *business_connection_id=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.

Shortcuts

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.
- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message to be edited was sent

  Added in version 21.4.

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. See *do_api_request()* for limitations.

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(*recursive=True*)

See *telegram.TelegramObject.to_dict()*.

property token

Bot's unique authentication token.

Added in version 20.0.

Type
str

async transferBusinessAccountStars(*business_connection_id*, *star_count*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *transfer_business_account_stars()*

async transferGift(*business_connection_id*, *owned_gift_id*, *new_owner_chat_id*, *star_count=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *transfer_gift()*

async transfer_business_account_stars(*business_connection_id*, *star_count*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Transfers Telegram Stars from the business account balance to the bot's balance. Requires the *can_transfer_stars* business bot right.

Added in version 22.1.

Parameters

- ***business_connection_id*** (str) – Unique identifier of the business connection
- ***star_count*** (int) – Number of Telegram Stars to transfer; *1-10000*

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async transfer_gift(*business_connection_id*, *owned_gift_id*, *new_owner_chat_id*, *star_count=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Transfers an owned unique gift to another user. Requires the *can_transfer_and_upgrade_gifts* business bot right. Requires *can_transfer_stars* business bot right if the transfer is paid.

Shortcuts

- *telegram.Chat.transfer_gift()*
- *telegram.ChatFullInfo.transfer_gift()*

Added in version 22.1.

Parameters

- ***business_connection_id*** (str) – Unique identifier of the business connection
- ***owned_gift_id*** (str) – Unique identifier of the regular gift that should be transferred.
- ***new_owner_chat_id*** (int) – Unique identifier of the chat which will own the gift. The chat must be active in the last *86400* seconds.
- ***star_count*** (int, optional) – The amount of Telegram Stars that will be paid for the transfer from the business account balance. If positive, then the *can_transfer_stars* business bot right is required.

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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

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*.

Shortcuts

- *telegram.Chat.unban_member()*
- *telegram.ChatFullInfo.unban_member()*

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.
- ***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. See *do_api_request()* for limitations.

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.

Shortcuts

- *telegram.Chat.unban_chat()*
- *telegram.Chat.unban_sender_chat()*
- *telegram.ChatFullInfo.unban_chat()*
- *telegram.ChatFullInfo.unban_sender_chat()*

Added in version 13.9.

Parameters

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

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

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

Alias for *unhide_general_forum_topic()*

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

Use this method to unhide the 'General' topic in a forum supergroup chat. The bot must be an administrator in the chat for this to work and must have *can_manage_topics* administrator rights.

Shortcuts

- *telegram.Chat.unhide_general_forum_topic()*
- *telegram.ChatFullInfo.unhide_general_forum_topic()*

Added in version 20.0.

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. See *do_api_request()* for limitations.

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 unpinAllForumTopicMessages(*chat_id*, *message_thread_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *unpin_all_forum_topic_messages()*

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

Alias for *unpin_all_general_forum_topic_messages()*

async unpinChatMessage(*chat_id*, *message_id=None*, *business_connection_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.

Shortcuts

- *telegram.Chat.unpin_all_messages()*
- *telegram.ChatFullInfo.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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async unpin_all_forum_topic_messages(*chat_id*, *message_thread_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 forum topic in a forum supergroup chat or a private chat with a user. In the case of a supergroup chat the bot must be an administrator in the chat for this to work and must have the *can_pin_messages* administrator right in the supergroup.

Shortcuts

- *telegram.Chat.unpin_all_forum_topic_messages()*
- *telegram.ChatFullInfo.unpin_all_forum_topic_messages()*
- *telegram.Message.unpin_all_forum_topic_messages()*

Added in version 20.0.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername).
- ***message_thread_id*** (int) – Unique identifier for the target message thread of the forum topic.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async unpin_all_general_forum_topic_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 General forum topic. The bot must be an administrator in the chat for this to work and must have *can_pin_messages* administrator rights in the supergroup.

Shortcuts

- *telegram.Chat.unpin_all_general_forum_topic_messages()*
- *telegram.ChatFullInfo.unpin_all_general_forum_topic_messages()*

Added in version 20.5.

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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

async unpin_chat_message(*chat_id*, *message_id=None*, *business_connection_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.

Shortcuts

- *telegram.Chat.unpin_message()*
- *telegram.ChatFullInfo.unpin_message()*
- *telegram.Message.unpin()*
- *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 the message to unpin. Required if *business_connection_id* is specified. If not specified, the most recent pinned message (by sending date) will be unpinned.
- ***business_connection_id*** (str, optional) – Unique identifier of the business connection on behalf of which the message will be unpinned.

  Added in version 21.5.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

async upgradeGift(*business_connection_id*, *owned_gift_id*, *keep_original_details=None*, *star_count=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *upgrade_gift()*

async upgrade_gift(*business_connection_id*, *owned_gift_id*, *keep_original_details=None*, *star_count=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Upgrades a given regular gift to a unique gift. Requires the *can_transfer_and_upgrade_gifts* business bot right. Additionally requires the *can_transfer_stars* business bot right if the upgrade is paid.

Added in version 22.1.

Parameters

- ***business_connection_id*** (str) – Unique identifier of the business connection
- ***owned_gift_id*** (str) – Unique identifier of the regular gift that should be upgraded to a unique one.
- ***keep_original_details*** (bool, optional) – Pass True to keep the original gift text, sender and receiver in the upgraded gift
- ***star_count*** (int, optional) – The amount of Telegram Stars that will be paid for the upgrade from the business account balance. If `gift.prepaid_upgrade_star_count > 0`, then pass `0`, otherwise, the

can_transfer_stars business bot right is required and *telegram.Gift.upgrade_star_count* must be passed.

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. See *do_api_request()* for limitations.

Returns

On success, True is returned.

Return type

bool

Raises

telegram.error.TelegramError –

async uploadStickerFile(*user_id*, *sticker*, *sticker_format*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Alias for *upload_sticker_file()*

async upload_sticker_file(*user_id*, *sticker*, *sticker_format*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

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

Changed in version 20.5: Removed deprecated parameter `png_sticker`.

Parameters

- ***user_id*** (int) – User identifier of sticker file owner.
- ***sticker*** (str | file object | *InputFile* | bytes | pathlib.Path) – A file with the sticker in the `".WEBP"`, `".PNG"`, `".TGS"` or `".WEBM"` format. See here for technical requirements . To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`), the file contents as bytes or the path of the file (as string or pathlib.Path object). In the latter case, the file contents will either be read as bytes or the file path will be passed to Telegram, depending on the *local_mode* setting.

  Added in version 20.2.

- ***sticker_format*** (str) – Format of the sticker. Must be one of *telegram.constants.StickerFormat.STATIC*, *telegram.constants.StickerFormat.ANIMATED*, *telegram.constants.StickerFormat.VIDEO*.

  Added in version 20.2.

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*. By default, 20 seconds are used as write timeout.

  Changed in version 22.0: The default value changed 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. See *do_api_request()* for limitations.

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

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

Alias for *verify_chat()*

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

Alias for *verify_user()*

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

Verifies a chat on behalf of the organization which is represented by the bot.

Shortcuts

- *telegram.Chat.verify()*
- *telegram.ChatFullInfo.verify()*

Added in version 21.10.

Parameters

- ***chat_id*** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- ***custom_description*** (str, optional) – Custom description for the verification; 0-*70* characters. Must be empty if the organization isn't allowed to provide a custom verification description.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

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

Verifies a user on behalf of the organization which is represented by the bot.

Shortcuts

telegram.User.verify()

Added in version 21.10.

Parameters

- ***user_id*** (int) – Unique identifier of the target user.
- ***custom_description*** (str, optional) – Custom description for the verification; 0-*70* characters. Must be empty if the organization isn't allowed to provide a custom verification description.

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. See *do_api_request()* for limitations.

Returns
On success, True is returned.

Return type
bool

Raises
telegram.error.TelegramError –

Available Types

AcceptedGiftTypes

class telegram.**AcceptedGiftTypes**(*unlimited_gifts*, *limited_gifts*, *unique_gifts*, *premium_subscription*, *gifts_from_channels*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object describes the types of gifts that can be gifted to a user or a chat.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal if their *unlimited_gifts*, *limited_gifts*, *unique_gifts*, *premium_subscription* and *gifts_from_channels* are equal.

Use In

telegram.Bot.set_business_account_gift_settings()

Available In

telegram.ChatFullInfo.accepted_gift_types

Added in version 22.1.

Changed in version 22.6: *gifts_from_channels* is now considered for equality checks.

Parameters

- ***unlimited_gifts*** (bool) – True, if unlimited regular gifts are accepted.
- ***limited_gifts*** (bool) – True, if limited regular gifts are accepted.
- ***unique_gifts*** (bool) – True, if unique gifts or gifts that can be upgraded to unique for free are accepted.
- ***premium_subscription*** (bool) – True, if a Telegram Premium subscription is accepted.
- ***gifts_from_channels*** (bool) – True, if transfers of unique gifts from channels are accepted

  Added in version 22.6.

unlimited_gifts

True, if unlimited regular gifts are accepted.

Type
bool

limited_gifts

True, if limited regular gifts are accepted.

Type
bool

unique_gifts

True, if unique gifts or gifts that can be upgraded to unique for free are accepted.

Type
bool

premium_subscription

True, if a Telegram Premium subscription is accepted.

Type
bool

gifts_from_channels

True, if transfers of unique gifts from channels are accepted

Added in version 22.6.

Type
bool

Animation

class telegram.**Animation**(*file_id*, *file_unique_id*, *width*, *height*, *duration*, *file_name=None*, *mime_type=None*, *file_size=None*, *thumbnail=None*, *, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.get_file()*
- *telegram.Bot.send_animation()*

Available In

- *telegram.ExternalReplyInfo.animation*
- *telegram.Game.animation*
- *telegram.InputMediaAnimation.media*
- *telegram.Message.animation*
- *telegram.Message.effective_attachment*
- *telegram.PollMedia.animation*

Changed in version 20.5: Removed the deprecated argument and attribute thumb.

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 the sender.
- ***height*** (int) – Video height as defined by the sender.
- ***duration*** (int | datetime.timedelta, optional) – Duration of the video in seconds as defined by the sender.

  Changed in version v22.2: datetime.timedelta objects are accepted in addition to plain int values.
- ***file_name*** (str, optional) – Original animation filename as defined by the sender.
- ***mime_type*** (str, optional) – MIME type of the file as defined by the sender.

- ***file_size*** (int, optional) – File size in bytes.
- ***thumbnail*** (*telegram.PhotoSize*, optional) – Animation thumbnail as defined by sender.

  Added in version 20.2.

file_id

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

Type
int

height

Video height as defined by the sender.

Type
int

duration

Duration of the video in seconds as defined by the sender.

Deprecated since version v22.2: In a future major version this attribute will be of type datetime.timedelta. You can opt-in early by setting *PTB_TIMEDELTA=true* or PTB_TIMEDELTA=1 as an environment variable.

Type
int | datetime.timedelta

file_name

Optional. Original animation filename as defined by the sender.

Type
str

mime_type

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

Type
str

file_size

Optional. File size in bytes.

Type
int

thumbnail

Optional. Animation thumbnail as defined by sender.

Added in version 20.2.

Type
telegram.PhotoSize

classmethod de_json(*data*, *bot=None*)

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 –

Audio

class telegram.**Audio**(*file_id*, *file_unique_id*, *duration*, *performer=None*, *title=None*, *mime_type=None*, *file_size=None*, *file_name=None*, *thumbnail=None*, ***, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.get_file()*
- *telegram.Bot.send_audio()*

Available In

- *telegram.ChatFullInfo.first_profile_audio*
- *telegram.ExternalReplyInfo.audio*
- *telegram.InputMediaAudio.media*
- *telegram.Message.audio*
- *telegram.Message.effective_attachment*
- *telegram.PollMedia.audio*
- *telegram.UserProfileAudios.audios*

Changed in version 20.5: Removed the deprecated argument and attribute `thumb`.

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` | `datetime.timedelta`) – Duration of the audio in seconds as defined by the sender.

  Changed in version v22.2: `datetime.timedelta` objects are accepted in addition to plain `int` values.

- ***performer*** (str, optional) – Performer of the audio as defined by the sender or by audio tags.
- ***title*** (str, optional) – Title of the audio as defined by the sender or by audio tags.
- ***file_name*** (str, optional) – Original filename as defined by the sender.
- ***mime_type*** (str, optional) – MIME type of the file as defined by the sender.
- ***file_size*** (int, optional) – File size in bytes.
- ***thumbnail*** (*telegram.PhotoSize*, optional) – Thumbnail of the album cover to which the music file belongs.

  Added in version 20.2.

file_id

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

Deprecated since version v22.2: In a future major version this attribute will be of type datetime.timedelta. You can opt-in early by setting *PTB_TIMEDELTA=true* or PTB_TIMEDELTA=1 as an environment variable.

Type
int | datetime.timedelta

performer

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

Type
str

title

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

Type
str

file_name

Optional. Original filename as defined by the sender.

Type
str

mime_type

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

Type
str

file_size

Optional. File size in bytes.

Type
int

thumbnail

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

Added in version 20.2.

Type
telegram.PhotoSize

classmethod de_json(*data*, *bot=None*)

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 –

Birthdate

class telegram.**Birthdate**(*day*, *month*, *year=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object describes the birthdate of a user.

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

Available In

telegram.ChatFullInfo.birthdate

Added in version 21.1.

Parameters

- ***day*** (int) – Day of the user's birth; 1-31.
- ***month*** (int) – Month of the user's birth; 1-12.
- ***year*** (int, optional) – Year of the user's birth.

day

Day of the user's birth; 1-31.

Type
int

month

Month of the user's birth; 1-12.

Type
int

year

Optional. Year of the user's birth.

Type
int

to_date(*year=None*)

Return the birthdate as a date object.

Changed in version 21.2: Now returns a `datetime.date` object instead of a `datetime.datetime` object, as was originally intended.

Parameters

year (`int`, optional) – The year to use. Required, if the *year* was not present.

Returns

The birthdate as a date object.

Return type

`datetime.date`

BotAccessSettings

class telegram.**BotAccessSettings**(*is_access_restricted*, *added_users=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object describes the access settings of a bot.

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

Returned In

telegram.Bot.get_managed_bot_access_settings()

Added in version 22.8.

Parameters

- ***is_access_restricted*** (`bool`) – `True`, if only selected users can access the bot. The bot's owner can always access it.
- ***added_users*** (Sequence[*telegram.User*], optional) – The list of other users who have access to the bot if the access is restricted.

is_access_restricted

`True`, if only selected users can access the bot. The bot's owner can always access it.

Type

`bool`

added_users

Optional. The list of other users who have access to the bot if the access is restricted.

Type

Sequence[*telegram.User*]

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

BotCommand

class telegram.**BotCommand**(*command*, *description*, ***, *api_kwargs=None*)

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.

Use In

telegram.Bot.set_my_commands()

Returned In

telegram.Bot.get_my_commands()

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; *1*- *32* characters. Can contain only lowercase English letters, digits and underscores.

Type

str

description

Description of the command; *1*- *256* characters.

Type

str

MAX_COMMAND = 32

telegram.constants.BotCommandLimit.MAX_COMMAND

Added in version 20.0.

MAX_DESCRIPTION = 256

telegram.constants.BotCommandLimit.MAX_DESCRIPTION

Added in version 20.0.

MIN_COMMAND = 1

telegram.constants.BotCommandLimit.MIN_COMMAND

Added in version 20.0.

MIN_DESCRIPTION = 1

telegram.constants.BotCommandLimit.MIN_DESCRIPTION

Added in version 20.0.

BotCommandScope

class telegram.**BotCommandScope**(*type*, *, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.delete_my_commands()*
- *telegram.Bot.get_my_commands()*
- *telegram.Bot.set_my_commands()*

Added 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=None*)

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*, optional) – The bot associated with this object. Defaults to None, in which case shortcut methods will not be available.

  Changed in version 21.4: *bot* is now optional and defaults to None

Returns
The Telegram object.

BotCommandScopeAllChatAdministrators

class telegram.**BotCommandScopeAllChatAdministrators**(*, *api_kwargs=None*)

Bases: *telegram.BotCommandScope*

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

Use In

- *telegram.Bot.delete_my_commands()*
- *telegram.Bot.get_my_commands()*
- *telegram.Bot.set_my_commands()*

Added in version 13.7.

type
Scope type *'all_chat_administrators'*.

Type
str

BotCommandScopeAllGroupChats

class telegram.**BotCommandScopeAllGroupChats**(*, *api_kwargs=None*)

Bases: *telegram.BotCommandScope*

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

Use In

- *telegram.Bot.delete_my_commands()*
- *telegram.Bot.get_my_commands()*
- *telegram.Bot.set_my_commands()*

Added in version 13.7.

type
Scope type *'all_group_chats'*.

Type
str

BotCommandScopeAllPrivateChats

class telegram.**BotCommandScopeAllPrivateChats**(*, *api_kwargs=None*)

Bases: *telegram.BotCommandScope*

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

Use In

- *telegram.Bot.delete_my_commands()*
- *telegram.Bot.get_my_commands()*
- *telegram.Bot.set_my_commands()*

Added in version 13.7.

type

Scope type *'all_private_chats'*.

Type
str

BotCommandScopeChat

class telegram.**BotCommandScopeChat**(*chat_id*, *, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.delete_my_commands()*
- *telegram.Bot.get_my_commands()*
- *telegram.Bot.set_my_commands()*

Added 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

BotCommandScopeChatAdministrators

class telegram.**BotCommandScopeChatAdministrators**(*chat_id*, *, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.delete_my_commands()*
- *telegram.Bot.get_my_commands()*
- *telegram.Bot.set_my_commands()*

Added 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

BotCommandScopeChatMember

class telegram.**BotCommandScopeChatMember**(*chat_id*, *user_id*, ***, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.delete_my_commands()*
- *telegram.Bot.get_my_commands()*
- *telegram.Bot.set_my_commands()*

Added 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

BotCommandScopeDefault

class telegram.**BotCommandScopeDefault**(*, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.delete_my_commands()*
- *telegram.Bot.get_my_commands()*
- *telegram.Bot.set_my_commands()*

Added in version 13.7.

type

Scope type *'default'*.

Type
str

BotDescription

class telegram.**BotDescription**(*description*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents the bot's description.

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

Returned In

telegram.Bot.get_my_description()

Added in version 20.2.

Parameters
description (str) – The bot's description.

description

The bot's description.

Type
str

BotName

class telegram.**BotName**(*name*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents the bot's name.

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

Returned In

telegram.Bot.get_my_name()

Added in version 20.3.

Parameters

name (str) – The bot's name.

name

The bot's name.

Type

str

MAX_LENGTH = 64

telegram.constants.BotNameLimit.MAX_NAME_LENGTH

BotShortDescription

class telegram.**BotShortDescription**(*short_description*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents the bot's short description.

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

Returned In

telegram.Bot.get_my_short_description()

Added in version 20.2.

Parameters

short_description (str) – The bot's short description.

short_description

The bot's short description.

Type

str

BusinessBotRights

class telegram.**BusinessBotRights**(*can_reply=None*, *can_read_messages=None*, *can_delete_sent_messages=None*, *can_delete_all_messages=None*, *can_edit_name=None*, *can_edit_bio=None*, *can_edit_profile_photo=None*, *can_edit_username=None*, *can_change_gift_settings=None*, *can_view_gifts_and_stars=None*, *can_convert_gifts_to_stars=None*, *can_transfer_and_upgrade_gifts=None*, *can_transfer_stars=None*, *can_manage_stories=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents the rights of a business bot.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if all their attributes are equal.

Available In

telegram.BusinessConnection.rights

Added in version 22.1.

Parameters

- ***can_reply*** (bool, optional) – True, if the bot can send and edit messages in the private chats that had incoming messages in the last 24 hours.
- ***can_read_messages*** (bool, optional) – True, if the bot can mark incoming private messages as read.
- ***can_delete_sent_messages*** (bool, optional) – True, if the bot can delete messages sent by the bot.
- ***can_delete_all_messages*** (bool, optional) – True, if the bot can delete all private messages in managed chats.
- ***can_edit_name*** (bool, optional) – True, if the bot can edit the first and last name of the business account.
- ***can_edit_bio*** (bool, optional) – True, if the bot can edit the bio of the business account.
- ***can_edit_profile_photo*** (bool, optional) – True, if the bot can edit the profile photo of the business account.
- ***can_edit_username*** (bool, optional) – True, if the bot can edit the username of the business account.
- ***can_change_gift_settings*** (bool, optional) – True, if the bot can change the privacy settings pertaining to gifts for the business account.
- ***can_view_gifts_and_stars*** (bool, optional) – True, if the bot can view gifts and the amount of Telegram Stars owned by the business account.
- ***can_convert_gifts_to_stars*** (bool, optional) – True, if the bot can convert regular gifts owned by the business account to Telegram Stars.
- ***can_transfer_and_upgrade_gifts*** (bool, optional) – True, if the bot can transfer and upgrade gifts owned by the business account.
- ***can_transfer_stars*** (bool, optional) – True, if the bot can transfer Telegram Stars received by the business account to its own account, or use them to upgrade and transfer gifts.
- ***can_manage_stories*** (bool, optional) – True, if the bot can post, edit and delete stories on behalf of the business account.

`can_reply`

Optional. True, if the bot can send and edit messages in the private chats that had incoming messages in the last 24 hours.

Type
bool

`can_read_messages`

Optional. True, if the bot can mark incoming private messages as read.

Type
bool

`can_delete_sent_messages`

Optional. True, if the bot can delete messages sent by the bot.

Type
bool

`can_delete_all_messages`

Optional. True, if the bot can delete all private messages in managed chats.

Type
bool

`can_edit_name`

Optional. True, if the bot can edit the first and last name of the business account.

Type
bool

`can_edit_bio`

Optional. True, if the bot can edit the bio of the business account.

Type
bool

`can_edit_profile_photo`

Optional. True, if the bot can edit the profile photo of the business account.

Type
bool

`can_edit_username`

Optional. True, if the bot can edit the username of the business account.

Type
bool

`can_change_gift_settings`

Optional. True, if the bot can change the privacy settings pertaining to gifts for the business account.

Type
bool

`can_view_gifts_and_stars`

Optional. True, if the bot can view gifts and the amount of Telegram Stars owned by the business account.

Type
bool

`can_convert_gifts_to_stars`

Optional. True, if the bot can convert regular gifts owned by the business account to Telegram Stars.

Type
bool

can_transfer_and_upgrade_gifts

Optional. True, if the bot can transfer and upgrade gifts owned by the business account.

Type
bool

can_transfer_stars

Optional. True, if the bot can transfer Telegram Stars received by the business account to its own account, or use them to upgrade and transfer gifts.

Type
bool

can_manage_stories

Optional. True, if the bot can post, edit and delete stories on behalf of the business account.

Type
bool

BusinessConnection

class telegram.**BusinessConnection**(*id*, *user*, *user_chat_id*, *date*, *is_enabled*, *rights=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes the connection of the bot with a business account.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal if their *id*, *user*, *user_chat_id*, *date*, *rights*, and *is_enabled* are equal.

Available In

telegram.Update.business_connection

Returned In

telegram.Bot.get_business_connection()

Added in version 21.1.

Changed in version 22.1: Equality comparison now considers *rights* instead of `can_reply`.

Removed in version 22.3: Removed argument and attribute `can_reply` deprecated by API 9.0.

Parameters

- ***id*** (`str`) – Unique identifier of the business connection.
- ***user*** (*telegram.User*) – Business account user that created the business connection.
- ***user_chat_id*** (`int`) – Identifier of a private chat with the user who created the business connection.
- ***date*** (`datetime.datetime`) – Date the connection was established in Unix time.
- ***is_enabled*** (`bool`) – True, if the connection is active.
- ***rights*** (*BusinessBotRights*, optional) – Rights of the business bot.

  Added in version 22.1.

id

Unique identifier of the business connection.

Type
str

user

Business account user that created the business connection.

Type
telegram.User

user_chat_id

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

Type
int

date

Date the connection was established in Unix time.

Type
datetime.datetime

is_enabled

True, if the connection is active.

Type
bool

rights

Optional. Rights of the business bot.

Added in version 22.1.

Type
BusinessBotRights

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

BusinessIntro

class telegram.**BusinessIntro**(*title=None*, *message=None*, *sticker=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object contains information about the start page settings of a Telegram Business account.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *title*, *message* and *sticker* are equal.

Available In

telegram.ChatFullInfo.business_intro

Added in version 21.1.

Parameters

- ***title*** (str, optional) – Title text of the business intro.
- ***message*** (str, optional) – Message text of the business intro.
- ***sticker*** (*telegram.Sticker*, optional) – Sticker of the business intro.

title

Optional. Title text of the business intro.

Type
str

message

Optional. Message text of the business intro.

Type
str

sticker

Optional. Sticker of the business intro.

Type
telegram.Sticker

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

BusinessLocation

class telegram.**BusinessLocation**(*address*, *location=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object contains information about the location of a Telegram Business account.

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

Available In

telegram.ChatFullInfo.business_location

Added in version 21.1.

Parameters

- ***address*** (str) – Address of the business.
- ***location*** (*telegram.Location*, optional) – Location of the business.

address

Address of the business.

Type
str

location

Optional. Location of the business.

Type
telegram.Location

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

BusinessOpeningHours

class telegram.**BusinessOpeningHours**(*time_zone_name*, *opening_hours*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object describes the opening hours of a business.

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

Available In

telegram.ChatFullInfo.business_opening_hours

Added in version 21.1.

Parameters

- ***time_zone_name*** (str) – Unique name of the time zone for which the opening hours are defined.
- ***opening_hours*** (Sequence[*telegram.BusinessOpeningHoursInterval*]) – List of time intervals describing business opening hours.

time_zone_name

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

Type
str

opening_hours

List of time intervals describing business opening hours.

Type
Sequence[*telegram.BusinessOpeningHoursInterval*]

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

get_opening_hours_for_day(*date*, *time_zone=None*)

Returns the opening hours intervals for a specific day as datetime objects.

Added in version 22.5.

Parameters

- ***date*** (datetime.date) – The date to get opening hours for.
- ***time_zone*** (datetime.tzinfo | str, optional) – Timezone to use for the returned datetime objects. If not specified, then *time_zone_name* be used.

Returns
A tuple of datetime pairs representing opening and closing times for the specified day. Each pair consists of `(opening_time, closing_time)`. Returns an empty tuple if there are no opening hours for the given day.

Return type
tuple[tuple[datetime.datetime, datetime.datetime], ...]

is_open(*datetime*)

Check if the business is open at the specified datetime.

Added in version 22.5.

Parameters
datetime (datetime.datetime) – The datetime to check. If the object is timezone-naive, it is assumed to be in the timezone specified by *time_zone_name*.

Returns
True if the business is open at the specified time, False otherwise.

Return type
bool

BusinessOpeningHoursInterval

class telegram.**BusinessOpeningHoursInterval**(*opening_minute*, *closing_minute*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object describes an interval of time during which a business is open.

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

Available In

telegram.BusinessOpeningHours.opening_hours

Added in version 21.1.

Examples

A day has (24 * 60 =) 1440 minutes, a week has (7 * 1440 =) 10080 minutes. Starting the minute's sequence from Monday, example values of *opening_minute*, *closing_minute* will map to the following day times:

- **Monday - 8am to 8:30pm:**
  - opening_minute = 480 *8 * 60*
  - closing_minute = 1230 *20 * 60 + 30*
- **Tuesday - 24 hours:**
  - opening_minute = 1440 *24 * 60*
  - closing_minute = 2879 *2 * 24 * 60 - 1*
- **Sunday - 12am - 11:58pm:**
  - opening_minute = 8640 *6 * 24 * 60*
  - closing_minute = 10078 *7 * 24 * 60 - 2*

Parameters

- ***opening_minute*** (int) – The minute's sequence number in a week, starting on Monday, marking the start of the time interval during which the business is open; 0 - 7 * 24 * 60.
- ***closing_minute*** (int) – The minute's sequence number in a week, starting on Monday, marking the end of the time interval during which the business is open; 0 - 8 * 24 * 60

opening_minute

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

Type
int

closing_minute

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

Type
int

property closing_time

Convenience attribute. A tuple parsed from *closing_minute*. It contains the *weekday*, *hour* and *minute* in the same ranges as datetime.datetime.weekday, datetime.datetime.hour and datetime.datetime.minute

Return type
tuple[int, int, int]

property opening_time

Convenience attribute. A tuple parsed from *opening_minute*. It contains the *weekday*, *hour* and *minute* in the same ranges as datetime.datetime.weekday, datetime.datetime.hour and datetime.datetime.minute

Return type
tuple[int, int, int]

BusinessMessagesDeleted

class telegram.**BusinessMessagesDeleted**(*business_connection_id*, *chat*, *message_ids*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal if their *business_connection_id*, *message_ids*, and *chat* are equal.

Available In

telegram.Update.deleted_business_messages

Added in version 21.1.

Parameters

- ***business_connection_id*** (str) – Unique identifier of the business connection.
- ***chat*** (*telegram.Chat*) – Information about a chat in the business account. The bot may not have access to the chat or the corresponding user.
- ***message_ids*** (Sequence[int]) – A list of identifiers of the deleted messages in the chat of the business account.

business_connection_id

Unique identifier of the business connection.

Type
str

chat

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

Type
telegram.Chat

message_ids

A list of identifiers of the deleted messages in the chat of the business account.

Type
tuple[int]

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

CallbackQuery

class telegram.**CallbackQuery**(*id*, *from_user*, *chat_instance*, *message=None*, *data=None*, *inline_message_id=None*, *game_short_name=None*, ***, *api_kwargs=None*)

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.callback_data_cache*, *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.

  Added in version 13.6.

Available In

telegram.Update.callback_query

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.MaybeInaccessibleMessage*, optional) – Message sent by the bot with the callback button that originated the query.

  Changed in version 20.8: Accept objects of type *telegram.MaybeInaccessibleMessage* since Bot API 7.0.

- ***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.

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. Useful for high scores in games.

Type
str

message

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

Changed in version 20.8: Objects may be of type *telegram.MaybeInaccessibleMessage* since Bot API 7.0.

Type
telegram.MaybeInaccessibleMessage

data

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

> **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, serves as the unique identifier for the game.

Type
str

MAX_ANSWER_TEXT_LENGTH = 200

telegram.constants.CallbackQueryLimit.ANSWER_CALLBACK_QUERY_TEXT_LENGTH

Added 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_markup=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *show_caption_above_media=None*, *allow_paid_broadcast=None*, *video_start_timestamp=None*, *suggested_post_parameters=None*, *message_effect_id=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=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,
    direct_messages_topic_id=update.message.direct_messages_topic.topic_id,
    *args,
    **kwargs
)
```

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

Changed in version 20.8: Raises `TypeError` if *message* is not accessible.

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

Return type
telegram.MessageId

Raises
TypeError –

classmethod de_json(*data*, *bot=None*)

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()*.

Changed in version 20.8: Raises `TypeError` if *message* is not accessible.

Returns
On success, `True` is returned.

Return type
`bool`

Raises
`TypeError` –

async edit_message_caption(*caption=None*, *reply_markup=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=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()*.

Changed in version 20.8: Raises `TypeError` if *message* is not accessible.

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

Return type
telegram.Message

Raises
`TypeError` –

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

Shortcut for:

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

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

Added in version 22.3.

Returns
On success, the edited Message is returned.

Return type
telegram.Message

Raises
`TypeError` –

async edit_message_live_location(*latitude=None*, *longitude=None*, *reply_markup=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, *live_period=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()*.

Changed in version 20.8: Raises TypeError if *message* is not accessible.

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

Return type
telegram.Message

Raises
TypeError –

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()*.

Changed in version 20.8: Raises TypeError if *message* is not accessible.

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
TypeError –

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()*.

Changed in version 20.8: Raises TypeError if *message* is not accessible.

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

Return type
telegram.Message

Raises
TypeError –

async edit_message_text(*text*, *parse_mode=None*, *reply_markup=None*, *entities=None*, *link_preview_options=None*, ***, *disable_web_page_preview=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()*.

Changed in version 20.8: Raises TypeError if *message* is not accessible.

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

Return type
telegram.Message

Raises
TypeError –

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()*.

Changed in version 20.8: Raises TypeError if *message* is not accessible.

Returns
tuple[*telegram.GameHighScore*]

Raises
TypeError –

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()*.

Changed in version 20.8: Raises TypeError if *message* is not accessible.

Returns
On success, True is returned.

Return type
bool

Raises
TypeError –

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()*.

Changed in version 20.8: Raises TypeError if *message* is not accessible.

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

Return type
telegram.Message

Raises
TypeError –

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()*.

Changed in version 20.8: Raises TypeError if *message* is not accessible.

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

Return type
telegram.Message

Raises
TypeError –

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()*.

Changed in version 20.8: Raises TypeError if *message* is not accessible.

Returns
On success, True is returned.

Return type
bool

Raises
TypeError –

Chat

class telegram.**Chat**(*id*, *type*, *title=None*, *username=None*, *first_name=None*, *last_name=None*, *is_forum=None*, *is_direct_messages=None*, *, *api_kwargs=None*)

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.

Available In

- *telegram.AffiliateInfo.affiliate_chat*
- *telegram.BusinessMessagesDeleted.chat*
- *telegram.ChatBoostRemoved.chat*
- *telegram.ChatBoostUpdated.chat*

- *telegram.ChatFullInfo.parent_chat*
- *telegram.ChatFullInfo.personal_chat*
- *telegram.ChatJoinRequest.chat*
- *telegram.ChatMemberUpdated.chat*
- *telegram.ChecklistTask.completed_by_chat*
- *telegram.ExternalReplyInfo.chat*
- *telegram.Gift.publisher_chat*
- *telegram.Giveaway.chats*
- *telegram.GiveawayWinners.chat*
- *telegram.InaccessibleMessage.chat*
- *telegram.MaybeInaccessibleMessage.chat*
- *telegram.Message.chat*
- *telegram.Message.guest_bot_caller_chat*
- *telegram.Message.sender_chat*
- *telegram.MessageOriginChannel.chat*
- *telegram.MessageOriginChat.sender_chat*
- *telegram.MessageReactionCountUpdated.chat*
- *telegram.MessageReactionUpdated.actor_chat*
- *telegram.MessageReactionUpdated.chat*
- *telegram.PollAnswer.voter_chat*
- *telegram.PollOption.added_by_chat*
- *telegram.Story.chat*
- *telegram.TransactionPartnerChat.chat*
- *telegram.UniqueGift.publisher_chat*
- *telegram.Update.effective_chat*
- *telegram.Update.effective_sender*

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.

Changed in version 20.0: Removed the attribute `all_members_are_administrators`. As long as Telegram provides this field for backwards compatibility, it is available through *api_kwargs*.

Changed in version 21.3: As per Bot API 7.3, most of the arguments and attributes of this class have now moved to *telegram.ChatFullInfo*.

Parameters

- ***id*** (`int`) – Unique identifier for this chat.
- ***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.
- ***is_forum*** (bool, optional) – True, if the supergroup chat is a forum (has topics enabled).

  Added in version 20.0.
- ***is_direct_messages*** (bool, optional) – True, if the chat is the direct messages chat of a channel.

  Added in version 22.4.

id

Unique identifier for this chat.

Type
int

type

Type of chat, can be either *PRIVATE*, *GROUP*, *SUPERGROUP* or *CHANNEL*.

Type
str

title

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

Type
str

username

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

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

is_forum

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

Added in version 20.0.

Type
bool

is_direct_messages

Optional. True, if the chat is the direct messages chat of a channel.

Added in version 22.4.

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

Added 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()*.

Added in version 13.8.

Returns
On success, `True` is returned.

Return type
`bool`

async approve_suggested_post(*message_id*, *send_date=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.approve_suggested_post(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 22.4.

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()*.

Added 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()*.

Added in version 13.9.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.close_forum_topic(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.close_general_forum_topic(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 20.0.

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_markup=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *show_caption_above_media=None*, *allow_paid_broadcast=None*, *video_start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_effect_id=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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()*.

See also

send_copy(), *send_copies()*, *copy_messages()*.

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

Return type
telegram.MessageId

async copy_messages(*chat_id*, *message_ids*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *remove_caption=None*, *direct_messages_topic_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.copy_messages(from_chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

See also

copy_message(), *send_copy()*, *send_copies()*.

Added in version 20.8.

Returns
On success, a tuple of *MessageId* of the sent messages is returned.

Return type
tuple[*telegram.MessageId*]

async create_forum_topic(*name*, *icon_color=None*, *icon_custom_emoji_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.create_forum_topic(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 20.0.

Returns
telegram.ForumTopic

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()*.

Added 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

async create_subscription_invite_link(*subscription_period*, *subscription_price*, *name=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.create_chat_subscription_invite_link(
    chat_id=update.effective_chat.id, *args, **kwargs
)
```

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

Added in version 21.5.

Returns
telegram.ChatInviteLink

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()*.

Added in version 13.8.

Returns
On success, `True` is returned.

Return type
`bool`

async decline_suggested_post(*message_id*, *comment=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.decline_suggested_post(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 22.4.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.delete_all_message_reactions(
    chat_id=update.effective_chat.id,
    *args,
    **kwargs
)
```

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

Added in version 22.8.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.delete_forum_topic(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.delete_message(update.effective_chat.id, *args, **kwargs)
```

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

Added in version 20.8.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.delete_messages(update.effective_chat.id, *args, **kwargs)
```

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

Added in version 20.8.

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()*.

Added in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

async delete_reaction(*message_id*, *user_id=None*, *actor_chat_id=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.delete_message_reaction(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 22.8.

Returns
On success, `True` is returned.

Return type
bool

async edit_forum_topic(*message_thread_id*, *name=None*, *icon_custom_emoji_id=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.edit_forum_topic(chat_id=update.effective_chat.id, *args, **kwargs)
```

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

Added in version 20.0.

Returns
On success, True is returned.

Return type
bool

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

Shortcut for:

```
await bot.edit_general_forum_topic(
   chat_id=update.effective_chat.id, *args, **kwargs
)
```

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

Added 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()*.

Added 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 edit_subscription_invite_link(*invite_link*, *name=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.edit_chat_subscription_invite_link(
    chat_id=update.effective_chat.id, *args, **kwargs
)
```

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

Added in version 21.5.

Returns
telegram.ChatInviteLink

property effective_name

Convenience property. Gives *title* if not None, else *full_name* if not None.

Added in version 20.1.

Type
str

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()*.

Added 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*, *message_thread_id=None*, *video_start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_effect_id=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(), *forward_messages_from()*, *forward_messages_to()*

Added in version 20.0.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async **forward_messages_from**(*from_chat_id*, *message_ids*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *direct_messages_topic_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.forward_messages(chat_id=update.effective_chat.id, *args, **kwargs)
```

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

See also

forward_to(), *forward_from()*, *forward_messages_to()*.

Added in version 20.8.

Returns
On success, a tuple of *MessageId* of sent messages is returned.

Return type
tuple[*telegram.MessageId*]

async **forward_messages_to**(*chat_id*, *message_ids*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *direct_messages_topic_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.forward_messages(from_chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

See also

forward_from(), *forward_to()*, *forward_messages_from()*.

Added in version 20.8.

Returns
On success, a tuple of *MessageId* of sent messages is returned.

Return type
tuple[*telegram.MessageId*]

async **forward_to**(*chat_id*, *message_id*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *video_start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_effect_id=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()*, *forward_messages_from()*, *forward_messages_to()*

Added 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.

Added in version 13.2.

Type
str

async get_administrators(*return_bots=None*, ***, *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 tuple 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
tuple[*telegram.ChatMember*]

async get_gifts(*exclude_unsaved=None*, *exclude_saved=None*, *exclude_unlimited=None*, *exclude_limited_upgradable=None*, *exclude_limited_non_upgradable=None*, *exclude_from_blockchain=None*, *exclude_unique=None*, *sort_by_price=None*, *offset=None*, *limit=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.get_chat_gifts(chat_id=update.effective_chat.id)
```

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

Added in version 22.6.

Returns
On success, returns the gifts owned by the chat.

Return type
telegram.OwnedGifts

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.get_chat_menu_button()*.

Caution

Can only work, if the chat is a private chat.

See also

set_menu_button()

Added in version 20.0.

Returns
On success, the current menu button is returned.

Return type
telegram.MenuButton

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

Shortcut for:

```
await bot.get_user_chat_boosts(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 20.8.

Returns
On success, returns the boosts applied in the chat.

Return type
telegram.UserChatBoosts

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

Shortcut for:

```
await bot.hide_general_forum_topic(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

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`

mention_html(*name=None*)

Added in version 20.0.

Parameters
name (`str`) – The name used as a link for the chat. Defaults to *full_name*.

Returns
The inline mention for the chat as HTML.

Return type
`str`

Raises
TypeError – If the chat is a private chat and neither the *name* nor the *first_name* is set, then throw an `TypeError`. If the chat is a public chat and neither the *name* nor the *title* is set, then throw an `TypeError`. If chat is a private group chat, then throw an `TypeError`.

mention_markdown(*name=None*)

> **Note**
>
> *'Markdown'* is a legacy mode, retained by Telegram for backward compatibility. You should use *mention_markdown_v2()* instead.

Added in version 20.0.

Parameters
name (str) – The name used as a link for the chat. Defaults to *full_name*.

Returns
The inline mention for the chat as markdown (version 1).

Return type
str

Raises
TypeError – If the chat is a private chat and neither the *name* nor the *first_name* is set, then throw an TypeError. If the chat is a public chat and neither the *name* nor the *title* is set, then throw an TypeError. If chat is a private group chat, then throw an TypeError.

mention_markdown_v2(*name=None*)

Added in version 20.0.

Parameters
name (str) – The name used as a link for the chat. Defaults to *full_name*.

Returns
The inline mention for the chat as markdown (version 2).

Return type
str

Raises
TypeError – If the chat is a private chat and neither the *name* nor the *first_name* is set, then throw an TypeError. If the chat is a public chat and neither the *name* nor the *title* is set, then throw an TypeError. If chat is a private group chat, then throw an TypeError.

async pin_message(*message_id*, *disable_notification=None*, *business_connection_id=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*, *can_manage_topics=None*, *can_post_stories=None*, *can_edit_stories=None*, *can_delete_stories=None*, *can_manage_direct_messages=None*, *can_manage_tags=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()*.

Added 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.

Changed in version 20.6: The arguments *can_post_stories*, *can_edit_stories* and *can_delete_stories* were added.

Returns
On success, `True` is returned.

Return type
`bool`

async read_business_message(*business_connection_id*, *message_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.read_business_message(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 22.1.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.remove_chat_verification(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 21.10.

Returns
On success, `True` is returned.

Return type
bool

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

Shortcut for:

```
await bot.reopen_forum_topic(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 20.0.

Returns
On success, True is returned.

Return type
bool

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

Shortcut for:

```
await bot.reopen_general_forum_topic(
   chat_id=update.effective_chat.id, *args, **kwargs
)
```

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

Added in version 20.0.

Returns
On success, True is returned.

Return type
bool

async **repost_story**(*business_connection_id*, *from_story_id*, *active_period*, *post_to_chat_page=None*, *protect_content=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
 await bot.repost_story(
    from_chat_id=update.effective_chat.id,
    *args, **kwargs
)
```

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

Added in version 22.6.

Returns
On success, *Story* is returned.

Return type
Story

async **restrict_member**(*user_id*, *permissions*, *until_date=None*, *use_independent_chat_permissions=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()*.

Added in version 13.2.

Added in version 20.1: Added *use_independent_chat_permissions*.

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()*.

Added in version 13.4.

Returns
telegram.ChatInviteLink

async send_action(*action*, *message_thread_id=None*, *business_connection_id=None*, *, *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_animation(*animation*, *duration=None*, *width=None*, *height=None*, *caption=None*, *parse_mode=None*, *disable_notification=None*, *reply_markup=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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*, *message_thread_id=None*, *business_connection_id=None*, ***, *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_checklist**(*business_connection_id*, *checklist*, *disable_notification=None*, *reply_markup=None*, *protect_content=None*, *reply_parameters=None*, *message_effect_id=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_checklist(chat_id=update.effective_chat.id, *args, **kwargs)
```

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

Added in version 22.3.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_contact(*phone_number=None*, *first_name=None*, *last_name=None*, *disable_notification=None*, *reply_markup=None*, *vcard=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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_copies(*from_chat_id*, *message_ids*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *remove_caption=None*, *direct_messages_topic_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.copy_messages(chat_id=update.effective_chat.id, *args, **kwargs)
```

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

See also

copy_message(), *send_copy()*, *copy_messages()*.

Added in version 20.8.

Returns
On success, a tuple of *MessageId* of the sent messages is returned.

Return type
tuple[*telegram.MessageId*]

async send_copy(*from_chat_id*, *message_id*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *disable_notification=None*, *reply_markup=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *show_caption_above_media=None*, *allow_paid_broadcast=None*, *video_start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_effect_id=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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()*.

> **See also**
>
> *copy_message()*, *send_copies()*, *copy_messages()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_dice(*disable_notification=None*, *reply_markup=None*, *emoji=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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_markup=None*, *parse_mode=None*, *disable_content_type_detection=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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_markup=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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_gift(*gift_id*, *text=None*, *text_parse_mode=None*, *text_entities=None*, *pay_for_upgrade=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_gift(user_id=update.effective_chat.id, *args, **kwargs )
```

or:

```
await bot.send_gift(chat_id=update.effective_chat.id, *args, **kwargs )
```

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

Caution

Will only work if the chat is a private or channel chat, see *type*.

Added in version 21.8.

Changed in version 21.11: Added support for channel chats.

Returns
On success, `True` is returned.

Return type
`bool`

async send_invoice(*title*, *description*, *payload*, *currency*, *prices*, *provider_token=None*, *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_markup=None*, *provider_data=None*, *send_phone_number_to_provider=None*, *send_email_to_provider=None*, *max_tip_amount=None*, *suggested_tip_amounts=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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_live_photo(*live_photo*, *photo*, *business_connection_id=None*, *message_thread_id=None*, *direct_messages_topic_id=None*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, *has_spoiler=None*, *disable_notification=None*, *protect_content=None*, *allow_paid_broadcast=None*, *message_effect_id=None*, *suggested_post_parameters=None*, *reply_parameters=None*, *reply_markup=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_live_photo(update.effective_chat.id, *args, **kwargs)
```

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

Added in version 22.8.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_location(*latitude=None*, *longitude=None*, *disable_notification=None*, *reply_markup=None*, *live_period=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*, *caption=None*, *parse_mode=None*, *caption_entities=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, a tuple of *Message* instances that were sent is returned.

Return type
tuple[*telegram.Message*]

async send_message(*text*, *parse_mode=None*, *disable_notification=None*, *reply_markup=None*, *entities=None*, *protect_content=None*, *message_thread_id=None*, *link_preview_options=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *disable_web_page_preview=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_message_draft(*draft_id*, *text=None*, *message_thread_id=None*, *parse_mode=None*, *entities=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_message_draft(update.effective_chat.id, *args, **kwargs)
```

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

Changed in version 22.8: Bot API 10.0 makes the `text` argument optional.

Returns
On success, `True` is returned.

Return type
`bool`

async send_paid_media(*star_count*, *media*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, *disable_notification=None*, *protect_content=None*, *reply_parameters=None*, *reply_markup=None*, *business_connection_id=None*, *payload=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_thread_id=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_paid_media(chat_id=update.effective_chat.id, *args, **kwargs)
```

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

Added in version 21.4.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_photo(*photo*, *caption=None*, *disable_notification=None*, *reply_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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_markup=None*, *explanation=None*, *explanation_parse_mode=None*, *open_period=None*, *close_date=None*, *explanation_entities=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *question_parse_mode=None*, *question_entities=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *shuffle_options=None*, *allows_revoting=None*, *correct_option_ids=None*, *allow_adding_options=None*, *hide_results_until_closes=None*, *description=None*, *description_parse_mode=None*, *description_entities=None*, *members_only=None*, *country_codes=None*, *explanation_media=None*, *media=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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_markup=None*, *protect_content=None*, *message_thread_id=None*, *emoji=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *read_timeout=None*, *write_timeout=None*, *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_markup=None*, *foursquare_type=None*, *google_place_id=None*, *google_place_type=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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_markup=None*, *width=None*, *height=None*, *parse_mode=None*, *supports_streaming=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *cover=None*, *start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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_markup=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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_chat_member_tag(*user_id*, *tag=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.set_chat_member_tag(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 22.7.

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()*.

Added 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()

Added in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

async set_message_reaction(*message_id*, *reaction=None*, *is_big=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.set_message_reaction(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 20.8.

Returns
bool On success, True is returned.

async set_permissions(*permissions*, *use_independent_chat_permissions=None*, *, *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()*.

Added in version 20.1: Added *use_independent_chat_permissions*.

Returns
On success, True is returned.

Return type
bool

async set_photo(*photo*, *, *read_timeout=None*, *write_timeout=None*, *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()*.

Added 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()*.

Added in version 20.0.

Returns
On success, True is returned.

Return type
bool

async transfer_gift(*business_connection_id*, *owned_gift_id*, *star_count=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.transfer_gift(new_owner_chat_id=update.effective_chat.id, *args,␣
↪**kwargs )
```

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

Added in version 22.1.

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()*.

Added 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()*.

Added in version 13.9.

Returns
On success, `True` is returned.

Return type
bool

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

Shortcut for:

```
await bot.unhide_general_forum_topic (
   chat_id=update.effective_chat.id, *args, **kwargs
)
```

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

Added in version 20.0.

Returns
On success, True is returned.

Return type
bool

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

Shortcut for:

```
await bot.unpin_all_forum_topic_messages(chat_id=update.effective_chat.id,
   *args, **kwargs)
```

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

Added in version 20.0.

Returns
On success, True is returned.

Return type
bool

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

Shortcut for:

```
await bot.unpin_all_general_forum_topic_messages(chat_id=update.effective_
↪chat.id,
   *args, **kwargs)
```

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

Added in version 20.5.

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*, *business_connection_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`

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

Shortcut for:

```
await bot.verify_chat(chat_id=update.effective_chat.id, *args, **kwargs)
```

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

Added in version 21.10.

Returns
On success, `True` is returned.

Return type
`bool`

ChatAdministratorRights

Added in version 20.0.

class telegram.**ChatAdministratorRights**(*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_stories*, *can_edit_stories*, *can_delete_stories*, *can_post_messages=None*, *can_edit_messages=None*, *can_pin_messages=None*, *can_manage_topics=None*, *can_manage_direct_messages=None*, *can_manage_tags=None*, *, *api_kwargs=None*)

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*, *can_manage_topics*, *can_post_stories*, *can_delete_stories*, *can_edit_stories*, *can_manage_direct_messages* and *can_manage_tags* are equal.

Use In

telegram.Bot.set_my_default_administrator_rights()

Available In

- *telegram.KeyboardButtonRequestChat.bot_administrator_rights*
- *telegram.KeyboardButtonRequestChat.user_administrator_rights*

Returned In

telegram.Bot.get_my_default_administrator_rights()

Added in version 20.0.

Changed in version 20.0: *can_manage_topics* is considered as well when comparing objects of this type in terms of equality.

Changed in version 20.6: *can_post_stories*, *can_edit_stories*, and *can_delete_stories* are considered as well when comparing objects of this type in terms of equality.

Changed in version 21.1: As of this version, *can_post_stories*, *can_edit_stories*, and *can_delete_stories* is now required. Thus, the order of arguments had to be changed.

Changed in version 22.4: *can_manage_direct_messages* is considered as well when comparing objects of this type in terms of equality.

Changed in version 22.7: *can_manage_tags* is considered as well when comparing objects of this type in terms of equality.

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, get boost list, see hidden supergroup and channel members, report spam messages and ignore slow mode. Implied by any other administrator privilege.
- ***can_delete_messages*** (bool) – True, if the administrator can delete messages of other users.
- ***can_manage_video_chats*** (bool) – True, if the administrator can manage video chats.
- ***can_restrict_members*** (bool) – True, if the administrator can restrict, ban or unban chat members, or access supergroup statistics.
- ***can_promote_members*** (bool) – True, if the administrator can add new administrators with a subset of their own privileges or demote administrators that they have promoted, directly or indirectly (promoted by administrators that were appointed by the user).
- ***can_change_info*** (bool) – True, if the user is allowed to change the chat title , photo and other settings.

- *can_invite_users* (bool) – True, if the user is allowed to invite new users to the chat.
- *can_post_messages* (bool, optional) – True, if the administrator can post messages in the channel, or access channel statistics; for channels only.
- *can_edit_messages* (bool, optional) – True, if the administrator can edit messages of other users and can pin messages; for channels only.
- *can_pin_messages* (bool, optional) – True, if the user is allowed to pin messages; for groups and supergroups only.
- *can_post_stories* (bool) – True, if the administrator can post stories to the chat.

  Added in version 20.6.

  Changed in version 21.0: As of this version, this argument is now required. In accordance with our stability policy, the signature will be kept as optional for now, though they are mandatory and an error will be raised if you don't pass it.
- *can_edit_stories* (bool) – True, if the administrator can edit stories posted by other users, post stories to the chat page, pin chat stories, and access the chat's story archive

  Added in version 20.6.

  Changed in version 21.0: As of this version, this argument is now required. In accordance with our stability policy, the signature will be kept as optional for now, though they are mandatory and an error will be raised if you don't pass it.
- *can_delete_stories* (bool) – True, if the administrator can delete stories posted by other users.

  Added in version 20.6.

  Changed in version 21.0: As of this version, this argument is now required. In accordance with our stability policy, the signature will be kept as optional for now, though they are mandatory and an error will be raised if you don't pass it.
- *can_manage_topics* (bool, optional) – True, if the user is allowed to create, rename, close, and reopen forum topics; for supergroups only.

  Added in version 20.0.
- *can_manage_direct_messages* (bool, optional) – True, if the administrator can manage direct messages of the channel and decline suggested posts; for channels only.

  Added in version 22.4.
- *can_manage_tags* (bool, optional) – True, if the administrator can edit the tags of regular members; for groups and supergroups only. If omitted defaults to the value of *can_pin_messages*.

  Added in version 22.7.

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, get boost list, see hidden supergroup and channel members, report spam messages 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, or access supergroup statistics.

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, or access channel statistics; for channels only.

Type
bool

can_edit_messages

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

Type
bool

can_pin_messages

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

Type
bool

can_post_stories

True, if the administrator can post stories to the chat.

Added in version 20.6.

Changed in version 21.0: As of this version, this argument is now required. In accordance with our stability policy, the signature will be kept as optional for now, though they are mandatory and an error will be raised if you don't pass it.

Type
bool

can_edit_stories

True, if the administrator can edit stories posted by other users, post stories to the chat page, pin chat stories, and access the chat's story archive

Added in version 20.6.

Changed in version 21.0: As of this version, this argument is now required. In accordance with our stability policy, the signature will be kept as optional for now, though they are mandatory and an error will be raised if you don't pass it.

Type
bool

can_delete_stories

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

Added in version 20.6.

Changed in version 21.0: As of this version, this argument is now required. In accordance with our stability policy, the signature will be kept as optional for now, though they are mandatory and an error will be raised if you don't pass it.

Type
bool

can_manage_topics

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

Added in version 20.0.

Type
bool

can_manage_direct_messages

Optional. True, if the administrator can manage direct messages of the channel and decline suggested posts; for channels only.

Added in version 22.4.

Type
bool

can_manage_tags

Optional. True, if the administrator can edit the tags of regular members; for groups and supergroups only. If omitted defaults to the value of *can_pin_messages*.

Added in version 22.7.

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()*.

Added in version 20.0.

classmethod no_rights()

This method returns the *ChatAdministratorRights* object with all attributes set to False.

Added in version 20.0.

ChatBackground

Added in version 21.2.

class telegram.**ChatBackground**(*type*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a chat background.

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

Available In

telegram.Message.chat_background_set

Added in version 21.2.

Parameters

type (*telegram.BackgroundType*) – Type of the background.

type

Type of the background.

Type

telegram.BackgroundType

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

Checklist

class telegram.**Checklist**(*title*, *tasks*, *title_entities=None*, *others_can_add_tasks=None*, *others_can_mark_tasks_as_done=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a checklist.

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

Available In

- *telegram.ExternalReplyInfo.checklist*
- *telegram.Message.checklist*

Added in version 22.3.

Parameters

- ***title*** (str) – Title of the checklist.
- ***title_entities*** (Sequence[*telegram.MessageEntity*], optional) – Special entities that appear in the checklist title.
- ***tasks*** (Sequence[*telegram.ChecklistTask*]) – List of tasks in the checklist.
- ***others_can_add_tasks*** (bool, optional) – True if users other than the creator of the list can add tasks to the list
- ***others_can_mark_tasks_as_done*** (bool, optional) – True if users other than the creator of the list can mark tasks as done or not done

title

Title of the checklist.

Type
str

title_entities

Optional. Special entities that appear in the checklist title.

Type
Tuple[*telegram.MessageEntity*]

tasks

List of tasks in the checklist.

Type
Tuple[*telegram.ChecklistTask*]

others_can_add_tasks

Optional. `True` if users other than the creator of the list can add tasks to the list

Type
bool

others_can_mark_tasks_as_done

Optional. `True` if users other than the creator of the list can mark tasks as done or not done

Type
bool

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

parse_entities(*types=None*)

Returns a `dict` that maps *telegram.MessageEntity* to `str`. It contains entities from this checklist's title 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 *title_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 `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_entity(*entity*)

Returns the text in *title* from a given *telegram.MessageEntity* of *title_entities*.

> **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 *title* with the offset and length.)

Parameters

entity (*telegram.MessageEntity*) – The entity to extract the text from. It must be an entity that belongs to *title_entities*.

Returns

The text of the given entity.

Return type

str

ChecklistTask

class telegram.**ChecklistTask**(*id*, *text*, *text_entities=None*, *completed_by_user=None*, *completion_date=None*, *completed_by_chat=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a task in a checklist.

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

> **Available In**
>
> - *telegram.Checklist.tasks*
> - *telegram.ChecklistTasksAdded.tasks*

Added in version 22.3.

Parameters

- ***id*** (int) – Unique identifier of the task.
- ***text*** (str) – Text of the task.
- ***text_entities*** (Sequence[*telegram.MessageEntity*], optional) – Special entities that appear in the task text.
- ***completed_by_user*** (*telegram.User*, optional) – User that completed the task; omitted if the task wasn't completed
- ***completed_by_chat*** (*telegram.Chat*, optional) – Chat that completed the task; omitted if the task wasn't completed by a chat

  Added in version 22.6.
- ***completion_date*** (datetime.datetime, optional) – Point in time when the task was completed; *ZERO_DATE* if the task wasn't completed

  The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

id

Unique identifier of the task.

Type

int

text

Text of the task.

Type
str

text_entities

Optional. Special entities that appear in the task text.

Type
Tuple[*telegram.MessageEntity*]

completed_by_user

Optional. User that completed the task; omitted if the task wasn't completed

Type
telegram.User

completed_by_chat

Optional. Chat that completed the task; omitted if the task wasn't completed by a chat

Added in version 22.6.

Type
telegram.Chat

completion_date

Optional. Point in time when the task was completed; *ZERO_DATE* if the task wasn't completed

The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

parse_entities(*types=None*)

Returns a dict that maps *telegram.MessageEntity* to str. It contains entities from this checklist task 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_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_entity(*entity*)

Returns the text in *text* from a given *telegram.MessageEntity* of *text_entities*.

> **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 `ChecklistTask.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 *text_entities*.

Returns

The text of the given entity.

Return type

str

ChecklistTasksAdded

class telegram.**ChecklistTasksAdded**(*tasks*, *checklist_message=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a service message about tasks added to a checklist.

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

> **Available In**
>
> *telegram.Message.checklist_tasks_added*

Added in version 22.3.

Parameters

- ***checklist_message*** (*telegram.Message*, optional) – Message containing the checklist to which tasks were added. Note that the ~:class:*telegram.Message* object in this field will not contain the *reply_to_message* field even if it itself is a reply.
- ***tasks*** (Sequence[*telegram.ChecklistTask*]) – List of tasks added to the checklist

checklist_message

Optional. Message containing the checklist to which tasks were added. Note that the ~:class:*telegram.Message* object in this field will not contain the *reply_to_message* field even if it itself is a reply.

Type

telegram.Message

tasks

List of tasks added to the checklist

Type

Tuple[*telegram.ChecklistTask*]

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

ChecklistTasksDone

class telegram.**ChecklistTasksDone**(*checklist_message=None*, *marked_as_done_task_ids=None*, *marked_as_not_done_task_ids=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a service message about checklist tasks marked as done or not done.

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

Available In

telegram.Message.checklist_tasks_done

Added in version 22.3.

Parameters

- ***checklist_message*** (*telegram.Message*, optional) – Message containing the checklist whose tasks were marked as done or not done. Note that the ~:class:*telegram.Message* object in this field will not contain the *reply_to_message* field even if it itself is a reply.
- ***marked_as_done_task_ids*** (Sequence[int], optional) – Identifiers of the tasks that were marked as done
- ***marked_as_not_done_task_ids*** (Sequence[int], optional) – Identifiers of the tasks that were marked as not done

checklist_message

Optional. Message containing the checklist whose tasks were marked as done or not done. Note that the ~:class:*telegram.Message* object in this field will not contain the *reply_to_message* field even if it itself is a reply.

Type
telegram.Message

marked_as_done_task_ids

Optional. Identifiers of the tasks that were marked as done

Type
Tuple[int]

marked_as_not_done_task_ids

Optional. Identifiers of the tasks that were marked as not done

Type
Tuple[int]

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

CopyTextButton

class telegram.**CopyTextButton**(*text*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents an inline keyboard button that copies specified text to the clipboard.

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

Available In

telegram.InlineKeyboardButton.copy_text

Added in version 21.7.

Parameters

text (str) – The text to be copied to the clipboard; *1*- *256* characters

text

The text to be copied to the clipboard; *1*- *256* characters

Type

str

BackgroundType

Added in version 21.2.

class telegram.**BackgroundType**(*type*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Base class for Telegram BackgroundType Objects. It can be one of:

- *telegram.BackgroundTypeFill*
- *telegram.BackgroundTypeWallpaper*
- *telegram.BackgroundTypePattern*
- *telegram.BackgroundTypeChatTheme*.

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

Available In

telegram.ChatBackground.type

Added in version 21.2.

Parameters

type (str) – Type of the background. Can be one of: *FILL*, *WALLPAPER PATTERN* or *CHAT_THEME*.

type

Type of the background. Can be one of: *FILL*, *WALLPAPER PATTERN* or *CHAT_THEME*.

Type

str

CHAT_THEME = 'chat_theme'

telegram.constants.BackgroundTypeType.CHAT_THEME

FILL = 'fill'

telegram.constants.BackgroundTypeType.FILL

PATTERN = 'pattern'

telegram.constants.BackgroundTypeType.PATTERN

WALLPAPER = 'wallpaper'

telegram.constants.BackgroundTypeType.WALLPAPER

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

BackgroundTypeFill

Added in version 21.2.

class telegram.**BackgroundTypeFill**(*fill*, *dark_theme_dimming*, *, *api_kwargs=None*)

Bases: *telegram.BackgroundType*

The background is automatically filled based on the selected colors.

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

Available In

telegram.ChatBackground.type

Added in version 21.2.

Parameters

- ***fill*** (*telegram.BackgroundFill*) – The background fill.
- ***dark_theme_dimming*** (int) – Dimming of the background in dark themes, as a percentage; 0-*100*.

type

Type of the background. Always *FILL*.

Type
str

fill

The background fill.

Type
telegram.BackgroundFill

dark_theme_dimming

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

Type
int

BackgroundTypeWallpaper

Added in version 21.2.

class telegram.**BackgroundTypeWallpaper**(*document*, *dark_theme_dimming*, *is_blurred=None*, *is_moving=None*, *, *api_kwargs=None*)

Bases: *telegram.BackgroundType*

The background is a wallpaper in the *JPEG* format.

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

Available In

telegram.ChatBackground.type

Added in version 21.2.

Parameters

- ***document*** (*telegram.Document*) – Document with the wallpaper
- ***dark_theme_dimming*** (int) – Dimming of the background in dark themes, as a percentage; 0-*100*.
- ***is_blurred*** (bool, optional) – True, if the wallpaper is downscaled to fit in a 450x450 square and then box-blurred with radius 12
- ***is_moving*** (bool, optional) – True, if the background moves slightly when the device is tilted

type

Type of the background. Always *WALLPAPER*.

Type
str

document

Document with the wallpaper

Type
telegram.Document

dark_theme_dimming

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

Type
int

is_blurred

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

Type
bool

is_moving

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

Type
bool

BackgroundTypePattern

Added in version 21.2.

class telegram.**BackgroundTypePattern**(*document*, *fill*, *intensity*, *is_inverted=None*, *is_moving=None*, ***, *api_kwargs=None*)

Bases: *telegram.BackgroundType*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *document* and *fill* and *intensity* are equal.

Available In

telegram.ChatBackground.type

Added in version 21.2.

Parameters

- ***document*** (*telegram.Document*) – Document with the pattern.
- ***fill*** (*telegram.BackgroundFill*) – The background fill that is combined with the pattern.
- ***intensity*** (int) – Intensity of the pattern when it is shown above the filled background; 0-*100*.
- ***is_inverted*** (int, optional) – True, if the background fill must be applied only to the pattern itself. All other pixels are black in this case. For dark themes only.
- ***is_moving*** (bool, optional) – True, if the background moves slightly when the device is tilted.

type

Type of the background. Always *PATTERN*.

Type
str

document

Document with the pattern.

Type
telegram.Document

fill

The background fill that is combined with the pattern.

Type
telegram.BackgroundFill

intensity

Intensity of the pattern when it is shown above the filled background; 0-*100*.

Type
int

is_inverted

Optional. True, if the background fill must be applied only to the pattern itself. All other pixels are black in this case. For dark themes only.

Type
int

is_moving

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

Type
bool

BackgroundTypeChatTheme

Added in version 21.2.

class telegram.**BackgroundTypeChatTheme**(*theme_name*, *, *api_kwargs=None*)

Bases: *telegram.BackgroundType*

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

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

Available In

telegram.ChatBackground.type

Added in version 21.2.

Parameters

theme_name (str) – Name of the chat theme, which is usually an emoji.

type

Type of the background. Always *CHAT_THEME*.

Type

str

theme_name

Name of the chat theme, which is usually an emoji.

Type

str

BackgroundFill

Added in version 21.2.

class telegram.**BackgroundFill**(*type*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Base class for Telegram BackgroundFill Objects. It can be one of:

- *telegram.BackgroundFillSolid*
- *telegram.BackgroundFillGradient*
- *telegram.BackgroundFillFreeformGradient*

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

Available In

- *telegram.BackgroundTypeFill.fill*
- *telegram.BackgroundTypePattern.fill*

Added in version 21.2.

Parameters

type (str) – Type of the background fill. Can be one of: *SOLID*, *GRADIENT* or *FREEFORM_GRADIENT*.

type

Type of the background fill. Can be one of: *SOLID*, *GRADIENT* or *FREEFORM_GRADIENT*.

Type
str

FREEFORM_GRADIENT = 'freeform_gradient'
telegram.constants.BackgroundFillType.FREEFORM_GRADIENT

GRADIENT = 'gradient'
telegram.constants.BackgroundFillType.GRADIENT

SOLID = 'solid'
telegram.constants.BackgroundFillType.SOLID

classmethod de_json(*data*, *bot=None*)
See *telegram.TelegramObject.de_json()*.

BackgroundFillSolid

Added in version 21.2.

class telegram.**BackgroundFillSolid**(*color*, ***, *api_kwargs=None*)
Bases: *telegram.BackgroundFill*

The background is filled using the selected color.

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

Available In
- *telegram.BackgroundTypeFill.fill*
- *telegram.BackgroundTypePattern.fill*

Added in version 21.2.

Parameters
color (int) – The color of the background fill in the *RGB24* format.

type
Type of the background fill. Always *SOLID*.

Type
str

color
The color of the background fill in the *RGB24* format.

Type
int

BackgroundFillGradient

Added in version 21.2.

class telegram.**BackgroundFillGradient**(*top_color*, *bottom_color*, *rotation_angle*, ***, *api_kwargs=None*)
Bases: *telegram.BackgroundFill*

The background is a gradient fill.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *top_color*, *bottom_color* and *rotation_angle* are equal.

Available In

- *telegram.BackgroundTypeFill.fill*
- *telegram.BackgroundTypePattern.fill*

Added in version 21.2.

Parameters

- ***top_color*** (int) – Top color of the gradient in the *RGB24* format.
- ***bottom_color*** (int) – Bottom color of the gradient in the *RGB24* format.
- ***rotation_angle*** (int) – Clockwise rotation angle of the background fill in degrees; 0-*359*.

type

Type of the background fill. Always *GRADIENT*.

Type
str

top_color

Top color of the gradient in the *RGB24* format.

Type
int

bottom_color

Bottom color of the gradient in the *RGB24* format.

Type
int

rotation_angle

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

Type
int

BackgroundFillFreeformGradient

Added in version 21.2.

class telegram.**BackgroundFillFreeformGradient**(*colors*, *, *api_kwargs=None*)

Bases: *telegram.BackgroundFill*

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

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

Available In

- *telegram.BackgroundTypeFill.fill*
- *telegram.BackgroundTypePattern.fill*

Added in version 21.2.

Parameters

colors (Sequence[int]) – A list of the 3 or 4 base colors that are used to generate the freeform gradient in the *RGB24* format

type

Type of the background fill. Always *FREEFORM_GRADIENT*.

Type

str

colors

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

Type

Sequence[int]

ChatBoost

Added in version 20.8.

class telegram.**ChatBoost**(*boost_id*, *add_date*, *expiration_date*, *source*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object contains information about a chat boost.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *boost_id*, *add_date*, *expiration_date*, and *source* are equal.

Available In

- *telegram.ChatBoostUpdated.boost*
- *telegram.UserChatBoosts.boosts*

Added in version 20.8.

Parameters

- ***boost_id*** (str) – Unique identifier of the boost.
- ***add_date*** (datetime.datetime) – Point in time when the chat was boosted.
- ***expiration_date*** (datetime.datetime) – Point in time when the boost will automatically expire, unless the booster's Telegram Premium subscription is prolonged.
- ***source*** (*telegram.ChatBoostSource*) – Source of the added boost.

boost_id

Unique identifier of the boost.

Type

str

add_date

Point in time when the chat was boosted. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type

datetime.datetime

expiration_date

Point in time when the boost will automatically expire, unless the booster's Telegram Premium subscription is prolonged. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

source
Source of the added boost.

Type
telegram.ChatBoostSource

classmethod de_json(*data*, *bot=None*)
See *telegram.TelegramObject.de_json()*.

ChatBoostAdded

class telegram.**ChatBoostAdded**(*boost_count*, ***, *api_kwargs=None*)
Bases: *telegram.TelegramObject*

This object represents a service message about a user boosting a chat.

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

Available In
telegram.Message.boost_added

Added in version 21.0.

Parameters
boost_count (int) – Number of boosts added by the user.

boost_count
Number of boosts added by the user.

Type
int

ChatBoostRemoved

Added in version 20.8.

class telegram.**ChatBoostRemoved**(*chat*, *boost_id*, *remove_date*, *source*, ***, *api_kwargs=None*)
Bases: *telegram.TelegramObject*

This object represents a boost removed from a chat.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *chat*, *boost_id*, *remove_date*, and *source* are equal.

Available In
telegram.Update.removed_chat_boost

Parameters
- ***chat*** (*telegram.Chat*) – Chat which was boosted.
- ***boost_id*** (str) – Unique identifier of the boost.
- ***remove_date*** (datetime.datetime) – Point in time when the boost was removed.
- ***source*** (*telegram.ChatBoostSource*) – Source of the removed boost.

chat

Chat which was boosted.

Type
telegram.Chat

boost_id

Unique identifier of the boost.

Type
str

remove_date

Point in time when the boost was removed. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

source

Source of the removed boost.

Type
telegram.ChatBoostSource

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

ChatBoostSource

Added in version 20.8.

class telegram.**ChatBoostSource**(*source*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Base class for Telegram ChatBoostSource objects. It can be one of:

- *telegram.ChatBoostSourcePremium*
- *telegram.ChatBoostSourceGiftCode*
- *telegram.ChatBoostSourceGiveaway*

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

Available In

- *telegram.ChatBoost.source*
- *telegram.ChatBoostRemoved.source*

Added in version 20.8.

Parameters
source (str) – The source of the chat boost. Can be one of: *PREMIUM*, *GIFT_CODE*, or *GIVEAWAY*.

source

The source of the chat boost. Can be one of: *PREMIUM*, *GIFT_CODE*, or *GIVEAWAY*.

Type
str

GIFT_CODE = 'gift_code'

telegram.constants.ChatBoostSources.GIFT_CODE

GIVEAWAY = 'giveaway'

telegram.constants.ChatBoostSources.GIVEAWAY

PREMIUM = 'premium'

telegram.constants.ChatBoostSources.PREMIUM

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

ChatBoostSourceGiftCode

Added in version 20.8.

class telegram.**ChatBoostSourceGiftCode**(*user*, ***, *api_kwargs=None*)

Bases: *telegram.ChatBoostSource*

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

Available In

- *telegram.ChatBoost.source*
- *telegram.ChatBoostRemoved.source*

Added in version 20.8.

Parameters

user (*telegram.User*) – User for which the gift code was created.

source

The source of the chat boost. Always *GIFT_CODE*.

Type

str

user

User for which the gift code was created.

Type

telegram.User

ChatBoostSourceGiveaway

Added in version 20.8.

class telegram.**ChatBoostSourceGiveaway**(*giveaway_message_id*, *user=None*, *is_unclaimed=None*, *prize_star_count=None*, ***, *api_kwargs=None*)

Bases: *telegram.ChatBoostSource*

The boost was obtained by the creation of a Telegram Premium giveaway or a Telegram Star. This boosts the chat 4 times for the duration of the corresponding Telegram Premium subscription for Telegram Premium giveaways and *prize_star_count* / 500 times for one year for Telegram Star giveaways.

Available In

- *telegram.ChatBoost.source*

- *telegram.ChatBoostRemoved.source*

Added in version 20.8.

Parameters

- ***giveaway_message_id*** (int) – Identifier of a message in the chat with the giveaway; the message could have been deleted already. May be 0 if the message isn't sent yet.
- ***user*** (*telegram.User*, optional) – User that won the prize in the giveaway if any; for Telegram Premium giveaways only.
- ***prize_star_count*** (int, optional) – The number of Telegram Stars to be split between giveaway winners; for Telegram Star giveaways only.

  Added in version 21.6.
- ***is_unclaimed*** (bool, optional) – True, if the giveaway was completed, but there was no user to win the prize.

source

Source of the boost. Always *GIVEAWAY*.

Type
str

giveaway_message_id

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

Type
int

user

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

Type
telegram.User

prize_star_count

Optional. The number of Telegram Stars to be split between giveaway winners; for Telegram Star giveaways only.

Added in version 21.6.

Type
int

is_unclaimed

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

Type
bool

ChatBoostSourcePremium

Added in version 20.8.

class telegram.**ChatBoostSourcePremium**(*user*, *, *api_kwargs=None*)

Bases: *telegram.ChatBoostSource*

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

Available In

- *telegram.ChatBoost.source*
- *telegram.ChatBoostRemoved.source*

Added in version 20.8.

Parameters

user (*telegram.User*) – User that boosted the chat.

source

The source of the chat boost. Always *PREMIUM*.

Type

str

user

User that boosted the chat.

Type

telegram.User

ChatBoostUpdated

Added in version 20.8.

class telegram.**ChatBoostUpdated**(*chat*, *boost*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

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

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

Available In

telegram.Update.chat_boost

Added in version 20.8.

Parameters

- ***chat*** (*telegram.Chat*) – Chat which was boosted.
- ***boost*** (*telegram.ChatBoost*) – Information about the chat boost.

chat

Chat which was boosted.

Type

telegram.Chat

boost

Information about the chat boost.

Type

telegram.ChatBoost

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

ChatFullInfo

class telegram.**ChatFullInfo**(*id*, *type*, *accent_color_id*, *max_reaction_count*, *accepted_gift_types*, *title=None*, *username=None*, *first_name=None*, *last_name=None*, *is_forum=None*, *photo=None*, *active_usernames=None*, *birthdate=None*, *business_intro=None*, *business_location=None*, *business_opening_hours=None*, *personal_chat=None*, *available_reactions=None*, *background_custom_emoji_id=None*, *profile_accent_color_id=None*, *profile_background_custom_emoji_id=None*, *emoji_status_custom_emoji_id=None*, *emoji_status_expiration_date=None*, *bio=None*, *has_private_forwards=None*, *has_restricted_voice_and_video_messages=None*, *join_to_send_messages=None*, *join_by_request=None*, *description=None*, *invite_link=None*, *pinned_message=None*, *permissions=None*, *slow_mode_delay=None*, *unrestrict_boost_count=None*, *message_auto_delete_time=None*, *has_aggressive_anti_spam_enabled=None*, *has_hidden_members=None*, *has_protected_content=None*, *has_visible_history=None*, *sticker_set_name=None*, *can_set_sticker_set=None*, *custom_emoji_sticker_set_name=None*, *linked_chat_id=None*, *location=None*, *can_send_paid_media=None*, *is_direct_messages=None*, *parent_chat=None*, *rating=None*, *unique_gift_colors=None*, *paid_message_star_count=None*, *first_profile_audio=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object contains full information about 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.

Returned In

telegram.Bot.get_chat()

Added in version 21.2.

Changed in version 21.3: Explicit support for all shortcut methods known from *telegram.Chat* on this object. Previously those were only available because this class inherited from *telegram.Chat*.

Removed in version 22.3: Removed argument and attribute `can_send_gift` deprecated by API 9.0.

Parameters

- ***id*** (`int`) – Unique identifier for this chat.
- ***type*** (`str`) – Type of chat, can be either *PRIVATE*, *GROUP*, *SUPERGROUP* or *CHANNEL*.
- ***accent_color_id*** (`int`, optional) – Identifier of the *accent color* for the chat name and backgrounds of the chat photo, reply header, and link preview. See accent colors for more details.

  Added in version 20.8.
- ***max_reaction_count*** (`int`) – The maximum number of reactions that can be set on a message in the chat.

  Added in version 21.2.
- ***accepted_gift_types*** (*telegram.AcceptedGiftTypes*) – Information about types of gifts that are accepted by the chat or by the corresponding user for private chats.

Added in version 22.1.

- ***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.
- ***is_forum*** (bool, optional) – True, if the supergroup chat is a forum (has topics enabled).

  Added in version 20.0.
- ***photo*** (*telegram.ChatPhoto*, optional) – Chat photo.
- ***active_usernames*** (Sequence[str], optional) – If set, the list of all active chat usernames; for private chats, supergroups and channels.

  Added in version 20.0.
- ***birthdate*** (*telegram.Birthdate*, optional) – For private chats, the date of birth of the user.

  Added in version 21.1.
- ***business_intro*** (*telegram.BusinessIntro*, optional) – For private chats with business accounts, the intro of the business.

  Added in version 21.1.
- ***business_location*** (*telegram.BusinessLocation*, optional) – For private chats with business accounts, the location of the business.

  Added in version 21.1.
- ***business_opening_hours*** (*telegram.BusinessOpeningHours*, optional) – For private chats with business accounts, the opening hours of the business.

  Added in version 21.1.
- ***personal_chat*** (*telegram.Chat*, optional) – For private chats, the personal channel of the user.

  Added in version 21.1.
- ***available_reactions*** (Sequence[*telegram.ReactionType*], optional) – List of available reactions allowed in the chat. If omitted, then all of *telegram.constants.ReactionEmoji* are allowed.

  Added in version 20.8.
- ***background_custom_emoji_id*** (str, optional) – Custom emoji identifier of emoji chosen by the chat for the reply header and link preview background.

  Added in version 20.8.
- ***profile_accent_color_id*** (int, optional) – Identifier of the *accent color* for the chat's profile background. See profile accent colors for more details.

  Added in version 20.8.
- ***profile_background_custom_emoji_id*** (str, optional) – Custom emoji identifier of the emoji chosen by the chat for its profile background.

  Added in version 20.8.
- ***emoji_status_custom_emoji_id*** (str, optional) – Custom emoji identifier of emoji status of the chat or the other party in a private chat.

  Added in version 20.0.

- ***emoji_status_expiration_date*** (datetime.datetime, optional) – Expiration date of emoji status of the chat or the other party in a private chat, as a datetime object, if any.

  The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

  Added in version 20.5.

- ***bio*** (str, optional) – Bio of the other party in a private 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.

  Added in version 13.9.

- ***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.

  Added in version 20.0.

- ***join_to_send_messages*** (bool, optional) – True, if users need to join the supergroup before they can send messages.

  Added in version 20.0.

- ***join_by_request*** (bool, optional) – True, if all users directly joining the supergroup without using an invite link need to be approved by supergroup administrators.

  Added in version 20.0.

- ***description*** (str, optional) – Description, for groups, supergroups and channel chats.

- ***invite_link*** (str, optional) – Primary invite link, for groups, supergroups and channel.

- ***pinned_message*** (*telegram.Message*, optional) – The most recent pinned message (by sending date).

- ***permissions*** (*telegram.ChatPermissions*) – Optional. Default chat member permissions, for groups and supergroups.

- ***slow_mode_delay*** (int | datetime.timedelta, optional) – For supergroups, the minimum allowed delay between consecutive messages sent by each unprivileged user.

  Changed in version v22.2: datetime.timedelta objects are accepted in addition to plain int values.

- ***unrestrict_boost_count*** (int, optional) – For supergroups, the minimum number of boosts that a non-administrator user needs to add in order to ignore slow mode and chat permissions.

  Added in version 21.0.

- ***message_auto_delete_time*** (int | datetime.timedelta, optional) – The time after which all messages sent to the chat will be automatically deleted; in seconds.

  Added in version 13.4.

  Changed in version v22.2: datetime.timedelta objects are accepted in addition to plain int values.

- ***has_aggressive_anti_spam_enabled*** (bool, optional) – True, if aggressive anti-spam checks are enabled in the supergroup. The field is only available to chat administrators.

  Added in version 20.0.

- ***has_hidden_members*** (bool, optional) – True, if non-administrators can only get the list of bots and administrators in the chat.

  Added in version 20.0.

- ***has_protected_content*** (bool, optional) – True, if messages from the chat can't be forwarded to other chats.

  Added in version 13.9.

- ***has_visible_history*** (bool, optional) – True, if new chat members will have access to old messages; available only to chat administrators.

  Added in version 20.8.

- ***sticker_set_name*** (str, optional) – For supergroups, name of group sticker set.

- ***can_set_sticker_set*** (bool, optional) – True, if the bot can change group the sticker set.

- ***custom_emoji_sticker_set_name*** (str, optional) – For supergroups, the name of the group's custom emoji sticker set. Custom emoji from this set can be used by all users and bots in the group.

  Added in version 21.0.

- ***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.

- ***location*** (*telegram.ChatLocation*, optional) – For supergroups, the location to which the supergroup is connected.

- ***can_send_paid_media*** (bool, optional) – True, if paid media messages can be sent or forwarded to the channel chat. The field is available only for channel chats.

  Added in version 21.4.

- ***is_direct_messages*** (bool, optional) – True, if the chat is the direct messages chat of a channel.

  Added in version 22.4.

- ***parent_chat*** (*telegram.Chat*, optional) – Information about the corresponding channel chat; for direct messages chats only.

  Added in version 22.4.

- ***rating*** (*telegram.UserRating*, optional) – For private chats, the rating of the user if any.

  Added in version 22.6.

- ***unique_gift_colors*** (*telegram.UniqueGiftColors*, optional) – The color scheme based on a unique gift that must be used for the chat's name, message replies and link previews

  Added in version 22.6.

- ***paid_message_star_count*** (int, optional) – The number of Telegram Stars a general user has to pay to send a message to the chat

  Added in version 22.6.

- ***first_profile_audio*** (*telegram.Audio*, optional) – For private chats, the first audio added to the profile of the user.

  Added in version 22.7.

id

Unique identifier for this chat.

Type
int

type

Type of chat, can be either *PRIVATE*, *GROUP*, *SUPERGROUP* or *CHANNEL*.

Type
str

accent_color_id

Optional. Identifier of the *accent color* for the chat name and backgrounds of the chat photo, reply header, and link preview. See accent colors for more details.

Added in version 20.8.

Type
int

max_reaction_count

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

Added in version 21.2.

Type
int

accepted_gift_types

Information about types of gifts that are accepted by the chat or by the corresponding user for private chats.

Added in version 22.1.

Type
telegram.AcceptedGiftTypes

title

Title, for supergroups, channels and group chats.

Type
str, optional

username

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

Type
str, optional

first_name

First name of the other party in a private chat.

Type
str, optional

last_name

Last name of the other party in a private chat.

Type
str, optional

is_forum

True, if the supergroup chat is a forum (has topics enabled).

Added in version 20.0.

Type
bool, optional

photo

Optional. Chat photo.

Type
telegram.ChatPhoto

active_usernames

Optional. If set, the list of all active chat usernames; for private chats, supergroups and channels.

This list is empty if the chat has no active usernames or this chat instance was not obtained via *get_chat()*.

Added in version 20.0.

Type
tuple[str]

birthdate

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

Added in version 21.1.

Type
telegram.Birthdate

business_intro

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

Added in version 21.1.

Type
telegram.BusinessIntro

business_location

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

Added in version 21.1.

Type
telegram.BusinessLocation

business_opening_hours

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

Added in version 21.1.

Type
telegram.BusinessOpeningHours

personal_chat

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

Added in version 21.1.

Type
telegram.Chat

available_reactions

Optional. List of available reactions allowed in the chat. If omitted, then all of *telegram.constants.ReactionEmoji* are allowed.

Added in version 20.8.

Type
tuple[*telegram.ReactionType*]

background_custom_emoji_id

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

Added in version 20.8.

Type
str

profile_accent_color_id

Optional. Identifier of the *accent color* for the chat's profile background. See profile accent colors for more details.

Added in version 20.8.

Type
int

profile_background_custom_emoji_id

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

Added in version 20.8.

Type
str

emoji_status_custom_emoji_id

Optional. Custom emoji identifier of emoji status of the chat or the other party in a private chat.

Added in version 20.0.

Type
str

emoji_status_expiration_date

Optional. Expiration date of emoji status of the chat or the other party in a private chat, as a datetime object, if any.

The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Added in version 20.5.

Type
datetime.datetime

bio

Optional. Bio of the other party in a private 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.

Added in version 13.9.

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.

Added in version 20.0.

Type
bool

join_to_send_messages

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

Added in version 20.0.

Type
bool

join_by_request

Optional. True, if all users directly joining the supergroup without using an invite link need to be approved by supergroup administrators.

Added in version 20.0.

Type
bool

description

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

Type
str

invite_link

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

Type
str

pinned_message

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

Type
telegram.Message

permissions

Optional. Default chat member permissions, for groups and supergroups.

Type
telegram.ChatPermissions

slow_mode_delay

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

Deprecated since version v22.2: In a future major version this attribute will be of type datetime.timedelta. You can opt-in early by setting *PTB_TIMEDELTA=true* or PTB_TIMEDELTA=1 as an environment variable.

Type
int | datetime.timedelta

unrestrict_boost_count

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

Added in version 21.0.

Type
int

message_auto_delete_time

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

Added in version 13.4.

Deprecated since version v22.2: In a future major version this attribute will be of type `datetime.timedelta`. You can opt-in early by setting *PTB_TIMEDELTA=true* or `PTB_TIMEDELTA=1` as an environment variable.

Type
`int | datetime.timedelta`

has_aggressive_anti_spam_enabled

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

Added in version 20.0.

Type
`bool`

has_hidden_members

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

Added in version 20.0.

Type
`bool`

has_protected_content

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

Added in version 13.9.

Type
`bool`

has_visible_history

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

Added in version 20.8.

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`

custom_emoji_sticker_set_name

Optional. For supergroups, the name of the group's custom emoji sticker set. Custom emoji from this set can be used by all users and bots in the group.

Added in version 21.0.

Type
`str`

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.

Type
int

location

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

Type
telegram.ChatLocation

can_send_paid_media

Optional. True, if paid media messages can be sent or forwarded to the channel chat. The field is available only for channel chats.

Added in version 21.4.

Type
bool

is_direct_messages

Optional. True, if the chat is the direct messages chat of a channel.

Added in version 22.4.

Type
bool

parent_chat

Optional. Information about the corresponding channel chat; for direct messages chats only.

Added in version 22.4.

Type
telegram.Chat

rating

Optional. For private chats, the rating of the user if any.

Added in version 22.6.

Type
telegram.UserRating

unique_gift_colors

Optional. The color scheme based on a unique gift that must be used for the chat's name, message replies and link previews

Added in version 22.6.

Type
telegram.UniqueGiftColors

paid_message_star_count

Optional. The number of Telegram Stars a general user have to pay to send a message to the chat

Added in version 22.6.

Type
int

first_profile_audio

Optional. For private chats, the first audio added to the profile of the user.

Added in version 22.7.

Type
telegram.Audio

CHANNEL = 'channel'

telegram.constants.ChatType.CHANNEL

GROUP = 'group'

telegram.constants.ChatType.GROUP

PRIVATE = 'private'

telegram.constants.ChatType.PRIVATE

SENDER = 'sender'

telegram.constants.ChatType.SENDER

Added 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()*.

Added in version 13.8.

Returns
On success, `True` is returned.

Return type
`bool`

async approve_suggested_post(*message_id*, *send_date=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.approve_suggested_post(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 22.4.

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()*.

Added 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()*.

Added in version 13.9.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.close_forum_topic(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.close_general_forum_topic(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 20.0.

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_markup=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *show_caption_above_media=None*, *allow_paid_broadcast=None*, *video_start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_effect_id=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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()*.

See also

send_copy(), *send_copies()*, *copy_messages()*.

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

Return type
telegram.MessageId

async copy_messages(*chat_id*, *message_ids*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *remove_caption=None*, *direct_messages_topic_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.copy_messages(from_chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

See also

copy_message(), *send_copy()*, *send_copies()*.

Added in version 20.8.

Returns
On success, a tuple of *MessageId* of the sent messages is returned.

Return type
tuple[*telegram.MessageId*]

async create_forum_topic(*name*, *icon_color=None*, *icon_custom_emoji_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.create_forum_topic(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 20.0.

Returns
telegram.ForumTopic

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()*.

Added 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

async create_subscription_invite_link(*subscription_period*, *subscription_price*, *name=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.create_chat_subscription_invite_link(
    chat_id=update.effective_chat.id, *args, **kwargs
)
```

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

Added in version 21.5.

Returns
telegram.ChatInviteLink

classmethod de_json(*data*, *bot=None*)

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()*.

Added in version 13.8.

Returns
On success, `True` is returned.

Return type
`bool`

async decline_suggested_post(*message_id*, *comment=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.decline_suggested_post(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 22.4.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.delete_all_message_reactions(
    chat_id=update.effective_chat.id,
    *args,
    **kwargs
)
```

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

Added in version 22.8.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.delete_forum_topic(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.delete_message(update.effective_chat.id, *args, **kwargs)
```

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

Added in version 20.8.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.delete_messages(update.effective_chat.id, *args, **kwargs)
```

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

Added in version 20.8.

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()*.

Added in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

async delete_reaction(*message_id*, *user_id=None*, *actor_chat_id=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.delete_message_reaction(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 22.8.

Returns
On success, `True` is returned.

Return type
`bool`

async edit_forum_topic(*message_thread_id*, *name=None*, *icon_custom_emoji_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.edit_forum_topic(chat_id=update.effective_chat.id, *args, **kwargs)
```

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

Added in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.edit_general_forum_topic(
    chat_id=update.effective_chat.id, *args, **kwargs
)
```

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

Added 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()*.

Added 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 edit_subscription_invite_link(*invite_link*, *name=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.edit_chat_subscription_invite_link(
    chat_id=update.effective_chat.id, *args, **kwargs
)
```

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

Added in version 21.5.

Returns
telegram.ChatInviteLink

property effective_name

Convenience property. Gives *title* if not None, else *full_name* if not None.

Added in version 20.1.

Type
str

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()*.

Added 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*, *message_thread_id=None*, *video_start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_effect_id=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(), *forward_messages_from()*, *forward_messages_to()*

Added in version 20.0.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async forward_messages_from(*from_chat_id*, *message_ids*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *direct_messages_topic_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.forward_messages(chat_id=update.effective_chat.id, *args, **kwargs)
```

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

See also

forward_to(), *forward_from()*, *forward_messages_to()*.

Added in version 20.8.

Returns
On success, a tuple of *MessageId* of sent messages is returned.

Return type
tuple[*telegram.MessageId*]

async forward_messages_to(*chat_id*, *message_ids*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *direct_messages_topic_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.forward_messages(from_chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

See also

forward_from(), *forward_to()*, *forward_messages_from()*.

Added in version 20.8.

Returns
On success, a tuple of *MessageId* of sent messages is returned.

Return type
tuple[*telegram.MessageId*]

async forward_to(*chat_id*, *message_id*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *video_start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_effect_id=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(), *forward_messages_from()*, *forward_messages_to()*

Added 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.

Added in version 13.2.

Type
str

async get_administrators(*return_bots=None*, ***, *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 tuple 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
tuple[*telegram.ChatMember*]

async get_gifts(*exclude_unsaved=None*, *exclude_saved=None*, *exclude_unlimited=None*, *exclude_limited_upgradable=None*, *exclude_limited_non_upgradable=None*, *exclude_from_blockchain=None*, *exclude_unique=None*, *sort_by_price=None*, *offset=None*, *limit=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.get_chat_gifts(chat_id=update.effective_chat.id)
```

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

Added in version 22.6.

Returns
On success, returns the gifts owned by the chat.

Return type
telegram.OwnedGifts

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.get_chat_menu_button()*.

Caution

Can only work, if the chat is a private chat.

See also

set_menu_button()

Added in version 20.0.

Returns
On success, the current menu button is returned.

Return type
telegram.MenuButton

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

Shortcut for:

```
await bot.get_user_chat_boosts(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 20.8.

Returns
On success, returns the boosts applied in the chat.

Return type
telegram.UserChatBoosts

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

Shortcut for:

```
await bot.hide_general_forum_topic(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

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`

mention_html(*name=None*)

Added in version 20.0.

Parameters
name (`str`) – The name used as a link for the chat. Defaults to *full_name*.

Returns
The inline mention for the chat as HTML.

Return type
`str`

Raises
TypeError – If the chat is a private chat and neither the *name* nor the *first_name* is set, then throw an `TypeError`. If the chat is a public chat and neither the *name* nor the *title* is set, then throw an `TypeError`. If chat is a private group chat, then throw an `TypeError`.

mention_markdown(*name=None*)

> **Note**
>
> *'Markdown'* is a legacy mode, retained by Telegram for backward compatibility. You should use *mention_markdown_v2()* instead.

Added in version 20.0.

Parameters
name (str) – The name used as a link for the chat. Defaults to *full_name*.

Returns
The inline mention for the chat as markdown (version 1).

Return type
str

Raises
TypeError – If the chat is a private chat and neither the *name* nor the *first_name* is set, then throw an TypeError. If the chat is a public chat and neither the *name* nor the *title* is set, then throw an TypeError. If chat is a private group chat, then throw an TypeError.

mention_markdown_v2(*name=None*)

Added in version 20.0.

Parameters
name (str) – The name used as a link for the chat. Defaults to *full_name*.

Returns
The inline mention for the chat as markdown (version 2).

Return type
str

Raises
TypeError – If the chat is a private chat and neither the *name* nor the *first_name* is set, then throw an TypeError. If the chat is a public chat and neither the *name* nor the *title* is set, then throw an TypeError. If chat is a private group chat, then throw an TypeError.

async pin_message(*message_id*, *disable_notification=None*, *business_connection_id=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*, *can_manage_topics=None*, *can_post_stories=None*, *can_edit_stories=None*, *can_delete_stories=None*, *can_manage_direct_messages=None*, *can_manage_tags=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()*.

Added 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.

Changed in version 20.6: The arguments *can_post_stories*, *can_edit_stories* and *can_delete_stories* were added.

Returns
On success, `True` is returned.

Return type
`bool`

async read_business_message(*business_connection_id*, *message_id*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.read_business_message(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 22.1.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.remove_chat_verification(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 21.10.

Returns
On success, `True` is returned.

Return type
bool

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

Shortcut for:

```
await bot.reopen_forum_topic(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 20.0.

Returns
On success, `True` is returned.

Return type
bool

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

Shortcut for:

```
await bot.reopen_general_forum_topic(
   chat_id=update.effective_chat.id, *args, **kwargs
)
```

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

Added in version 20.0.

Returns
On success, `True` is returned.

Return type
bool

async **repost_story**(*business_connection_id*, *from_story_id*, *active_period*, *post_to_chat_page=None*, *protect_content=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
 await bot.repost_story(
    from_chat_id=update.effective_chat.id,
    *args, **kwargs
)
```

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

Added in version 22.6.

Returns
On success, *Story* is returned.

Return type
Story

async **restrict_member**(*user_id*, *permissions*, *until_date=None*, *use_independent_chat_permissions=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()*.

Added in version 13.2.

Added in version 20.1: Added *use_independent_chat_permissions*.

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()*.

Added in version 13.4.

Returns
telegram.ChatInviteLink

async send_action(*action*, *message_thread_id=None*, *business_connection_id=None*, *, *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_animation(*animation*, *duration=None*, *width=None*, *height=None*, *caption=None*, *parse_mode=None*, *disable_notification=None*, *reply_markup=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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*, *message_thread_id=None*, *business_connection_id=None*, ***, *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_checklist(*business_connection_id*, *checklist*, *disable_notification=None*, *reply_markup=None*, *protect_content=None*, *reply_parameters=None*, *message_effect_id=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_checklist(chat_id=update.effective_chat.id, *args, **kwargs)
```

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

Added in version 22.3.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_contact(*phone_number=None*, *first_name=None*, *last_name=None*, *disable_notification=None*, *reply_markup=None*, *vcard=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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_copies(*from_chat_id*, *message_ids*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *remove_caption=None*, *direct_messages_topic_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.copy_messages(chat_id=update.effective_chat.id, *args, **kwargs)
```

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

See also

copy_message(), *send_copy()*, *copy_messages()*.

Added in version 20.8.

Returns
On success, a tuple of *MessageId* of the sent messages is returned.

Return type
tuple[*telegram.MessageId*]

async send_copy(*from_chat_id*, *message_id*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *disable_notification=None*, *reply_markup=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *show_caption_above_media=None*, *allow_paid_broadcast=None*, *video_start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_effect_id=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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()*.

> **See also**
>
> *copy_message()*, *send_copies()*, *copy_messages()*.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_dice(*disable_notification=None*, *reply_markup=None*, *emoji=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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_markup=None*, *parse_mode=None*, *disable_content_type_detection=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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_markup=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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_gift(*gift_id*, *text=None*, *text_parse_mode=None*, *text_entities=None*, *pay_for_upgrade=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_gift(user_id=update.effective_chat.id, *args, **kwargs )
```

or:

```
await bot.send_gift(chat_id=update.effective_chat.id, *args, **kwargs )
```

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

Caution

Will only work if the chat is a private or channel chat, see *type*.

Added in version 21.8.

Changed in version 21.11: Added support for channel chats.

Returns
On success, `True` is returned.

Return type
`bool`

async send_invoice(*title*, *description*, *payload*, *currency*, *prices*, *provider_token=None*, *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_markup=None*, *provider_data=None*, *send_phone_number_to_provider=None*, *send_email_to_provider=None*, *max_tip_amount=None*, *suggested_tip_amounts=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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_live_photo**(*live_photo*, *photo*, *business_connection_id=None*, *message_thread_id=None*, *direct_messages_topic_id=None*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, *has_spoiler=None*, *disable_notification=None*, *protect_content=None*, *allow_paid_broadcast=None*, *message_effect_id=None*, *suggested_post_parameters=None*, *reply_parameters=None*, *reply_markup=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_live_photo(update.effective_chat.id, *args, **kwargs)
```

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

Added in version 22.8.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async **send_location**(*latitude=None*, *longitude=None*, *disable_notification=None*, *reply_markup=None*, *live_period=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*, *caption=None*, *parse_mode=None*, *caption_entities=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, a tuple of *Message* instances that were sent is returned.

Return type
tuple[*telegram.Message*]

async send_message(*text*, *parse_mode=None*, *disable_notification=None*, *reply_markup=None*, *entities=None*, *protect_content=None*, *message_thread_id=None*, *link_preview_options=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *disable_web_page_preview=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_message_draft(*draft_id*, *text=None*, *message_thread_id=None*, *parse_mode=None*, *entities=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_message_draft(update.effective_chat.id, *args, **kwargs)
```

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

Changed in version 22.8: Bot API 10.0 makes the `text` argument optional.

Returns
On success, `True` is returned.

Return type
`bool`

async send_paid_media(*star_count*, *media*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, *disable_notification=None*, *protect_content=None*, *reply_parameters=None*, *reply_markup=None*, *business_connection_id=None*, *payload=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_thread_id=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_paid_media(chat_id=update.effective_chat.id, *args, **kwargs)
```

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

Added in version 21.4.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_photo(*photo*, *caption=None*, *disable_notification=None*, *reply_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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_markup=None*, *explanation=None*, *explanation_parse_mode=None*, *open_period=None*, *close_date=None*, *explanation_entities=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *question_parse_mode=None*, *question_entities=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *shuffle_options=None*, *allows_revoting=None*, *correct_option_ids=None*, *allow_adding_options=None*, *hide_results_until_closes=None*, *description=None*, *description_parse_mode=None*, *description_entities=None*, *members_only=None*, *country_codes=None*, *explanation_media=None*, *media=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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_markup=None*, *protect_content=None*, *message_thread_id=None*, *emoji=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *read_timeout=None*, *write_timeout=None*, *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_markup=None*, *foursquare_type=None*, *google_place_id=None*, *google_place_type=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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_markup=None*, *width=None*, *height=None*, *parse_mode=None*, *supports_streaming=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *cover=None*, *start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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_markup=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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_chat_member_tag(*user_id*, *tag=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.set_chat_member_tag(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 22.7.

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()*.

Added 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()

Added in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

async set_message_reaction(*message_id*, *reaction=None*, *is_big=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.set_message_reaction(chat_id=update.effective_chat.id, *args,␣
↪**kwargs)
```

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

Added in version 20.8.

Returns
bool On success, True is returned.

async set_permissions(*permissions*, *use_independent_chat_permissions=None*, *, *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()*.

Added in version 20.1: Added *use_independent_chat_permissions*.

Returns
On success, True is returned.

Return type
bool

async set_photo(*photo*, *, *read_timeout=None*, *write_timeout=None*, *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()*.

Added 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()*.

Added in version 20.0.

Returns
On success, True is returned.

Return type
bool

async transfer_gift(*business_connection_id*, *owned_gift_id*, *star_count=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.transfer_gift(new_owner_chat_id=update.effective_chat.id, *args,␣
↪**kwargs )
```

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

Added in version 22.1.

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()*.

Added 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()*.

Added in version 13.9.

Returns
On success, `True` is returned.

Return type
bool

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

Shortcut for:

```
await bot.unhide_general_forum_topic (
   chat_id=update.effective_chat.id, *args, **kwargs
)
```

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

Added in version 20.0.

Returns
On success, True is returned.

Return type
bool

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

Shortcut for:

```
await bot.unpin_all_forum_topic_messages(chat_id=update.effective_chat.id,
   *args, **kwargs)
```

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

Added in version 20.0.

Returns
On success, True is returned.

Return type
bool

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

Shortcut for:

```
await bot.unpin_all_general_forum_topic_messages(chat_id=update.effective_
↪chat.id,
   *args, **kwargs)
```

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

Added in version 20.5.

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*, *business_connection_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`

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

Shortcut for:

```
await bot.verify_chat(chat_id=update.effective_chat.id, *args, **kwargs)
```

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

Added in version 21.10.

Returns
On success, `True` is returned.

Return type
`bool`

ChatInviteLink

class telegram.**ChatInviteLink**(*invite_link*, *creator*, *creates_join_request*, *is_primary*, *is_revoked*, *expire_date=None*, *member_limit=None*, *name=None*, *pending_join_request_count=None*, *subscription_period=None*, *subscription_price=None*, *, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.edit_chat_invite_link()*
- *telegram.Bot.edit_chat_subscription_invite_link()*
- *telegram.Bot.revoke_chat_invite_link()*

Available In

- *telegram.ChatJoinRequest.invite_link*
- *telegram.ChatMemberUpdated.invite_link*

Returned In

- *telegram.Bot.create_chat_invite_link()*
- *telegram.Bot.create_chat_subscription_invite_link()*
- *telegram.Bot.edit_chat_invite_link()*
- *telegram.Bot.edit_chat_subscription_invite_link()*
- *telegram.Bot.revoke_chat_invite_link()*

Added 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.

  Added 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.

  Changed in version 20.3: The default timezone of the bot is used for localization, 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.

  Added in version 13.8.
- ***pending_join_request_count*** (int, optional) – Number of pending join requests created using this link.

Added in version 13.8.

- ***subscription_period*** (int | datetime.timedelta, optional) – The number of seconds the subscription will be active for before the next payment.

  Added in version 21.5.

  Changed in version v22.2: datetime.timedelta objects are accepted in addition to plain int values.

- ***subscription_price*** (int, optional) – The amount of Telegram Stars a user must pay initially and after each subsequent subscription period to be a member of the chat using the link.

  Added in version 21.5.

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.

Added 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.

Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

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. 0-*32* characters.

Added in version 13.8.

Type
str

pending_join_request_count

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

Added in version 13.8.

Type
int

subscription_period

Optional. The number of seconds the subscription will be active for before the next payment.

Added in version 21.5.

Deprecated since version v22.2: In a future major version this attribute will be of type `datetime.timedelta`. You can opt-in early by setting *PTB_TIMEDELTA=true* or `PTB_TIMEDELTA=1` as an environment variable.

Type
int | datetime.timedelta

subscription_price

Optional. The amount of Telegram Stars a user must pay initially and after each subsequent subscription period to be a member of the chat using the link.

Added in version 21.5.

Type
int

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

ChatJoinRequest

class telegram.**ChatJoinRequest**(*chat*, *from_user*, *date*, *user_chat_id*, *bio=None*, *invite_link=None*, ***, *api_kwargs=None*)

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.
- Telegram does not guarantee that *from_user.id* coincides with the `chat_id` of the user. Please use *user_chat_id* to contact the user in response to their join request.

Available In

telegram.Update.chat_join_request

Added in version 13.8.

Changed in version 20.1: In Bot API 6.5 the argument *user_chat_id* was added, which changes the position of the optional arguments *bio* and *invite_link*.

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.

  Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***user_chat_id*** (int) – Identifier of a private chat with the user who sent the join request. This number may have more than 32 significant bits and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a 64-bit integer or double-precision float type are safe for storing this identifier. The bot can use this identifier for 5 minutes to send messages until the join request is processed, assuming no other administrator contacted the user.

  Added in version 20.1.
- ***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.

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.

Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

user_chat_id

Identifier of a private chat with the user who sent the join request. This number may have more than 32 significant bits and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a 64-bit integer or double-precision float type are safe for storing this identifier. The bot can use this identifier for 5 minutes to send messages until the join request is processed, assuming no other administrator contacted the user.

Added in version 20.1.

Type
int

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.

> **Note**
>
> When a user joins a *public* group via an invite link, this attribute may not be present. However, this behavior is undocument and may be subject to change. See this GitHub thread for some discussion.

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=None*)

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

ChatLocation

class telegram.**ChatLocation**(*location*, *address*, *, *api_kwargs=None*)

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.

Available In

telegram.ChatFullInfo.location

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.

location

The location to which the supergroup is connected. Can't be a live location.

Type
telegram.Location

address

Location address; *1*- *64* characters, as defined by the chat owner.

Type
str

MAX_ADDRESS = 64

telegram.constants.LocationLimit.MAX_CHAT_LOCATION_ADDRESS

Added in version 20.0.

MIN_ADDRESS = 1

telegram.constants.LocationLimit.MIN_CHAT_LOCATION_ADDRESS

Added in version 20.0.

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

ChatMember

class telegram.**ChatMember**(*user*, *status*, *, *api_kwargs=None*)

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.

Examples

Chat Member Bot

Available In

- *telegram.ChatMemberUpdated.new_chat_member*
- *telegram.ChatMemberUpdated.old_chat_member*

Returned In

- *telegram.Bot.get_chat_administrators()*
- *telegram.Bot.get_chat_member()*

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. Can be *ADMINISTRATOR*, *OWNER*, *BANNED*, *LEFT*, *MEMBER* or *RESTRICTED*.

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=None*)

See *telegram.TelegramObject.de_json()*.

ChatMemberAdministrator

class telegram.**ChatMemberAdministrator**(*user*, *can_be_edited*, *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_stories*, *can_edit_stories*, *can_delete_stories*, *can_post_messages=None*, *can_edit_messages=None*, *can_pin_messages=None*, *can_manage_topics=None*, *custom_title=None*, *can_manage_direct_messages=None*, *can_manage_tags=None*, ***, *api_kwargs=None*)

Bases: *telegram.ChatMember*

Represents a chat member that has some additional privileges.

Available In

- *telegram.ChatMemberUpdated.new_chat_member*
- *telegram.ChatMemberUpdated.old_chat_member*

Returned In

- *telegram.Bot.get_chat_administrators()*
- *telegram.Bot.get_chat_member()*

Added 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.
- The argument *can_manage_topics* was added, which changes the position of the optional argument *custom_title*.

Changed in version 21.1: As of this version, *can_post_stories*, *can_edit_stories*, and *can_delete_stories* is now required. Thus, the order of arguments had to be changed.

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, get boost list, see hidden supergroup and channel members, report spam messages and ignore slow mode. Implied by any other administrator privilege.
- *can_delete_messages* (bool) – True, if the administrator can delete messages of other users.
- *can_manage_video_chats* (bool) – True, if the administrator can manage video chats.

  Added 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 messages in the channel, or access channel statistics; for channels only.
- *can_edit_messages* (bool, optional) – True, if the administrator can edit messages of other users and can pin messages; for channels only.
- *can_pin_messages* (bool, optional) – True, if the user is allowed to pin messages; for groups and supergroups only.
- *can_post_stories* (bool) – True, if the administrator can post stories to the chat.

  Added in version 20.6.

  Changed in version 21.0: As of this version, this argument is now required. In accordance with our stability policy, the signature will be kept as optional for now, though they are mandatory and an error will be raised if you don't pass it.
- *can_edit_stories* (bool) – True, if the administrator can edit stories posted by other users, post stories to the chat page, pin chat stories, and access the chat's story archive

  Added in version 20.6.

  Changed in version 21.0: As of this version, this argument is now required. In accordance with our stability policy, the signature will be kept as optional for now, though they are mandatory and an error will be raised if you don't pass it.
- *can_delete_stories* (bool) – True, if the administrator can delete stories posted by other users.

  Added in version 20.6.

  Changed in version 21.0: As of this version, this argument is now required. In accordance with our stability policy, the signature will be kept as optional for now, though they are mandatory and an error will be raised if you don't pass it.
- *can_manage_topics* (bool, optional) – True, if the user is allowed to create, rename, close, and reopen forum topics; for supergroups only.

  Added in version 20.0.
- *custom_title* (str, optional) – Custom title for this user.
- *can_manage_direct_messages* (bool, optional) – True, if the administrator can manage direct messages of the channel and decline suggested posts; for channels only.

  Added in version 22.4.

- *can_manage_tags* (bool, optional) – True, if the administrator can edit the tags of regular members; for groups and supergroups only. If omitted defaults to the value of *can_pin_messages*.

  Added in version 22.7.

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, get boost list, see hidden supergroup and channel members, report spam messages 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.

Added in version 20.0.

Type
bool

can_restrict_members

True, if the administrator can restrict, ban or unban chat members, or access supergroup statistics.

Type
bool

can_promote_members

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

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 messages in the channel or access channel statistics; for channels only.

Type
bool

can_edit_messages

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

Type
bool

can_pin_messages

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

Type
bool

can_post_stories

True, if the administrator can post stories to the chat.

Added in version 20.6.

Changed in version 21.0: As of this version, this argument is now required. In accordance with our stability policy, the signature will be kept as optional for now, though they are mandatory and an error will be raised if you don't pass it.

Type
bool

can_edit_stories

True, if the administrator can edit stories posted by other users, post stories to the chat page, pin chat stories, and access the chat's story archive

Added in version 20.6.

Changed in version 21.0: As of this version, this argument is now required. In accordance with our stability policy, the signature will be kept as optional for now, though they are mandatory and an error will be raised if you don't pass it.

Type
bool

can_delete_stories

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

Added in version 20.6.

Changed in version 21.0: As of this version, this argument is now required. In accordance with our stability policy, the signature will be kept as optional for now, though they are mandatory and an error will be raised if you don't pass it.

Type
bool

can_manage_topics

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

Added in version 20.0.

Type
bool

custom_title

Optional. Custom title for this user.

Type
str

can_manage_direct_messages

Optional. True, if the administrator can manage direct messages of the channel and decline suggested posts; for channels only.

Added in version 22.4.

Type
bool

can_manage_tags

Optional. True, if the administrator can edit the tags of regular members; for groups and supergroups only. If omitted defaults to the value of *can_pin_messages*.

Added in version 22.7.

Type
bool

ChatMemberBanned

class telegram.**ChatMemberBanned**(*user*, *until_date*, *, *api_kwargs=None*)

Bases: *telegram.ChatMember*

Represents a chat member that was banned in the chat and can't return to the chat or view chat messages.

Available In

- *telegram.ChatMemberUpdated.new_chat_member*
- *telegram.ChatMemberUpdated.old_chat_member*

Returned In

- *telegram.Bot.get_chat_administrators()*
- *telegram.Bot.get_chat_member()*

Added 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.

Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

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.

Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

ChatMemberLeft

class telegram.**ChatMemberLeft**(*user*, *, *api_kwargs=None*)

Bases: *telegram.ChatMember*

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

Available In

- *telegram.ChatMemberUpdated.new_chat_member*
- *telegram.ChatMemberUpdated.old_chat_member*

Returned In

- *telegram.Bot.get_chat_administrators()*
- *telegram.Bot.get_chat_member()*

Added 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

ChatMemberMember

class telegram.**ChatMemberMember**(*user*, *until_date=None*, *tag=None*, *, *api_kwargs=None*)

Bases: *telegram.ChatMember*

Represents a chat member that has no additional privileges or restrictions.

Available In

- *telegram.ChatMemberUpdated.new_chat_member*
- *telegram.ChatMemberUpdated.old_chat_member*

Returned In

- *telegram.Bot.get_chat_administrators()*
- *telegram.Bot.get_chat_member()*

Added in version 13.7.

Parameters

- ***user*** (*telegram.User*) – Information about the user.
- ***until_date*** (datetime.datetime, optional) – Date when the user's subscription will expire.

  Added in version 21.5.
- ***tag*** (str, optional) – Tag of the member.

  Added in version 22.7.

status

The member's status in the chat, always *'member'*.

Type
str

user

Information about the user.

Type
telegram.User

until_date

Optional. Date when the user's subscription will expire.

Added in version 21.5.

Type
datetime.datetime

tag

Optional. Tag of the member.

Added in version 22.7.

Type
str

ChatMemberOwner

class telegram.**ChatMemberOwner**(*user*, *is_anonymous*, *custom_title=None*, *, *api_kwargs=None*)

Bases: *telegram.ChatMember*

Represents a chat member that owns the chat and has all administrator privileges.

Available In

- *telegram.ChatMemberUpdated.new_chat_member*
- *telegram.ChatMemberUpdated.old_chat_member*

Returned In

- *telegram.Bot.get_chat_administrators()*
- *telegram.Bot.get_chat_member()*

Added 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

ChatMemberRestricted

class telegram.**ChatMemberRestricted**(*user*, *is_member*, *can_change_info*, *can_invite_users*, *can_pin_messages*, *can_send_messages*, *can_send_polls*, *can_send_other_messages*, *can_add_web_page_previews*, *can_manage_topics*, *until_date*, *can_send_audios*, *can_send_documents*, *can_send_photos*, *can_send_videos*, *can_send_video_notes*, *can_send_voice_notes*, *can_edit_tag*, *tag=None*, *can_react_to_messages=None*, *, *api_kwargs=None*)

Bases: *telegram.ChatMember*

Represents a chat member that is under certain restrictions in the chat. Supergroups only.

Available In

- *telegram.ChatMemberUpdated.new_chat_member*
- *telegram.ChatMemberUpdated.old_chat_member*

Returned In

- *telegram.Bot.get_chat_administrators()*
- *telegram.Bot.get_chat_member()*

Added in version 13.7.

Changed in version 20.0: All arguments were made positional and their order was changed. The argument can_manage_topics was added.

Changed in version 20.5: Removed deprecated argument and attribute `can_send_media_messages`.

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, invoices, locations and venues.
- ***can_send_polls*** (`bool`) – `True`, if the user is allowed to send polls.
- ***can_send_other_messages*** (`bool`) – `True`, if the user is allowed to send animations, games, stickers and use inline bots.
- ***can_add_web_page_previews*** (`bool`) – `True`, if the user is allowed to add web page previews to their messages.
- ***can_manage_topics*** (`bool`) – `True`, if the user is allowed to create forum topics.

  Added in version 20.0.
- ***until_date*** (`datetime.datetime`) – Date when restrictions will be lifted for this user.

  Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***can_send_audios*** (`bool`) – `True`, if the user is allowed to send audios.

  Added in version 20.1.
- ***can_send_documents*** (`bool`) – `True`, if the user is allowed to send documents.

  Added in version 20.1.

- ***can_send_photos*** (bool) – True, if the user is allowed to send photos.

  Added in version 20.1.

- ***can_send_videos*** (bool) – True, if the user is allowed to send videos.

  Added in version 20.1.

- ***can_send_video_notes*** (bool) – True, if the user is allowed to send video notes.

  Added in version 20.1.

- ***can_send_voice_notes*** (bool) – True, if the user is allowed to send voice notes.

  Added in version 20.1.

- ***can_edit_tag*** (bool) – True, if the user is allowed to edit their own tag. If omitted, defaults to the value of *can_pin_messages*.

  Added in version 22.7.

- ***can_react_to_messages*** (bool) – True, if the user is allowed to react to messages.

  Added in version 22.8.

- ***tag*** (str, optional) – Tag of the member.

  Added in version 22.7.

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_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

can_manage_topics

True, if the user is allowed to create forum topics.

Added in version 20.0.

Type
bool

until_date

Date when restrictions will be lifted for this user.

Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

can_send_audios

True, if the user is allowed to send audios.

Added in version 20.1.

Type
bool

can_send_documents

True, if the user is allowed to send documents.

Added in version 20.1.

Type
bool

can_send_photos

True, if the user is allowed to send photos.

Added in version 20.1.

Type
bool

can_send_videos

True, if the user is allowed to send videos.

Added in version 20.1.

Type
bool

can_send_video_notes

True, if the user is allowed to send video notes.

Added in version 20.1.

Type
bool

can_send_voice_notes

True, if the user is allowed to send voice notes.

Added in version 20.1.

Type
bool

can_edit_tag

True, if the user is allowed to edit their own tag. If omitted, defaults to the value of *can_pin_messages*.

Added in version 22.7.

Type
bool

can_react_to_messages

True, if the user is allowed to react to messages.

Added in version 22.8.

Type
bool

tag

Optional. Tag of the member.

Added in version 22.7.

Type
str

ChatMemberUpdated

class telegram.**ChatMemberUpdated**(*chat*, *from_user*, *date*, *old_chat_member*, *new_chat_member*, *invite_link=None*, *via_chat_folder_invite_link=None*, *via_join_request=None*, ***, *api_kwargs=None*)

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.

Available In

- *telegram.Update.chat_member*
- *telegram.Update.my_chat_member*

Added in version 13.4.

Note

In Python `from` is a reserved word. Use *`from_user`* instead.

Examples

Chat Member Bot

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.

  Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***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.
- ***via_chat_folder_invite_link*** (bool, optional) – True, if the user joined the chat via a chat folder invite link

  Added in version 20.3.
- ***via_join_request*** (bool, optional) – True, if the user joined the chat after sending a direct join request without using an invite link and being approved by an administrator

  Added in version 21.2.

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.

Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

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. For joining by invite link events only.

Type
telegram.ChatInviteLink

via_chat_folder_invite_link

Optional. `True`, if the user joined the chat via a chat folder invite link

Added in version 20.3.

Type
`bool`

via_join_request

Optional. `True`, if the user joined the chat after sending a direct join request without using an invite link and being approved by an administrator

Added in version 21.2.

Type
`bool`

classmethod de_json(*data*, *bot=None*)

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.

Added 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`]]

ChatOwnerChanged

class telegram.**ChatOwnerChanged**(*new_owner*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a service message about an ownership change in the chat.

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

Available In

telegram.Message.chat_owner_changed

Added in version 22.7.

Parameters

new_owner (*telegram.User*) – The new owner of the chat

new_owner

The new owner of the chat

Type

telegram.User

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

ChatOwnerLeft

class telegram.**ChatOwnerLeft**(*new_owner=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a service message about the chat owner leaving the chat.

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

Available In

telegram.Message.chat_owner_left

Added in version 22.7.

Parameters

new_owner (*telegram.User*, optional) – The user who will become the new owner of the chat if the previous owner does not return to the chat

new_owner

Optional. The user who will become the new owner of the chat if the previous owner does not return to the chat

Type

telegram.User

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

ChatPermissions

class telegram.**ChatPermissions**(*can_send_messages=None*, *can_send_polls=None*, *can_send_other_messages=None*, *can_add_web_page_previews=None*, *can_change_info=None*, *can_invite_users=None*, *can_pin_messages=None*, *can_manage_topics=None*, *can_send_audios=None*, *can_send_documents=None*, *can_send_photos=None*, *can_send_videos=None*, *can_send_video_notes=None*, *can_send_voice_notes=None*, *can_edit_tag=None*, *can_react_to_messages=None*, *, *api_kwargs=None*)

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_polls*, *can_send_other_messages*, *can_add_web_page_previews*, *can_change_info*, *can_invite_users*, *can_pin_messages*, *can_send_audios*, *can_send_documents*, *can_send_photos*, *can_send_videos*, *can_send_video_notes*, *can_send_voice_notes*, *can_manage_topics*, *can_edit_tag*, and *can_react_to_messages* are equal.

Use In

- *telegram.Bot.restrict_chat_member()*
- *telegram.Bot.set_chat_permissions()*

Available In

telegram.ChatFullInfo.permissions

Changed in version 20.0: *can_manage_topics* is considered as well when comparing objects of this type in terms of equality.

Changed in version 20.5:

- *can_send_audios*, *can_send_documents*, *can_send_photos*, *can_send_videos*, *can_send_video_notes* and *can_send_voice_notes* are considered as well when comparing objects of this type in terms of equality.
- Removed deprecated argument and attribute `can_send_media_messages`.

Changed in version 22.7: *can_edit_tag* is considered as well when comparing objects of this type in terms of equality.

Changed in version 22.8: *can_react_to_messages* is considered as well when comparing objects of this type in terms of equality.

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 behavior 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_polls*** (bool, optional) – True, if the user is allowed to send polls.
- ***can_send_other_messages*** (bool, optional) – True, if the user is allowed to send animations, games, stickers and use inline bots.
- ***can_add_web_page_previews*** (bool, optional) – True, if the user is allowed to add web page previews to their 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_manage_topics*** (bool, optional) – True, if the user is allowed to create forum topics. If omitted defaults to the value of *can_pin_messages*.

  Added in version 20.0.
- ***can_send_audios*** (bool) – True, if the user is allowed to send audios.

  Added in version 20.1.
- ***can_send_documents*** (bool) – True, if the user is allowed to send documents.

  Added in version 20.1.
- ***can_send_photos*** (bool) – True, if the user is allowed to send photos.

  Added in version 20.1.
- ***can_send_videos*** (bool) – True, if the user is allowed to send videos.

  Added in version 20.1.
- ***can_send_video_notes*** (bool) – True, if the user is allowed to send video notes.

  Added in version 20.1.
- ***can_send_voice_notes*** (bool) – True, if the user is allowed to send voice notes.

  Added in version 20.1.
- ***can_edit_tag*** (bool, optional) – True, if the user is allowed to edit their own tag.

  Added in version 22.7.
- ***can_react_to_messages*** (bool, optional) – True, if the user is allowed to react to messages. If omitted, defaults to the value of *can_send_messages*.

  Added in version 22.8.

can_send_messages

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

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.

Type
bool

can_add_web_page_previews

Optional. True, if the user is allowed to add web page previews to their 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

can_manage_topics

Optional. True, if the user is allowed to create forum topics. If omitted defaults to the value of *can_pin_messages*.

Added in version 20.0.

Type
bool

can_send_audios

True, if the user is allowed to send audios.

Added in version 20.1.

Type
bool

can_send_documents

True, if the user is allowed to send documents.

Added in version 20.1.

Type
bool

can_send_photos

True, if the user is allowed to send photos.

Added in version 20.1.

Type
bool

can_send_videos

True, if the user is allowed to send videos.

Added in version 20.1.

Type
bool

can_send_video_notes

True, if the user is allowed to send video notes.

Added in version 20.1.

Type
bool

can_send_voice_notes

True, if the user is allowed to send voice notes.

Added in version 20.1.

Type
bool

can_edit_tag

Optional. True, if the user is allowed to edit their own tag.

Added in version 22.7.

Type
bool

can_react_to_messages

Optional. True, if the user is allowed to react to messages. If omitted, defaults to the value of *can_send_messages*.

Added in version 22.8.

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()*.

Added in version 20.0.

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

classmethod no_permissions()

This method returns an *ChatPermissions* instance with all attributes set to False.

Added in version 20.0.

ChatPhoto

class telegram.**ChatPhoto**(*small_file_id*, *small_file_unique_id*, *big_file_id*, *big_file_unique_id*, ***, *api_kwargs=None*)

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.

Use In

telegram.Bot.get_file()

Available In

telegram.ChatFullInfo.photo

Parameters

- ***small_file_id*** (str) – File identifier of small (*160* x *160*) 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 (*160* x *160*) chat photo, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.
- ***big_file_id*** (str) – File identifier of big (*640* x *640*) 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 (*640* x *640*) 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.

small_file_id

File identifier of small (*160* x *160*) 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 (*160* x *160*) 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 (*640* x *640*) 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 (*640* x *640*) 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

SIZE_BIG = 640

telegram.constants.ChatPhotoSize.BIG

Added in version 20.0.

SIZE_SMALL = 160

telegram.constants.ChatPhotoSize.SMALL

Added in version 20.0.

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 (*640* x *640*) 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 (*160* x *160*) chat photo

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

Returns
telegram.File

Raises
telegram.error.TelegramError –

ChatShared

class telegram.**ChatShared**(*request_id*, *chat_id*, *title=None*, *username=None*, *photo=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object contains information about the chat whose identifier was shared with the bot using a *telegram.KeyboardButtonRequestChat* button.

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

Available In

telegram.Message.chat_shared

Added in version 20.1.

Parameters

- ***request_id*** (int) – Identifier of the request.
- ***chat_id*** (int) – Identifier of the shared user. 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.
- ***title*** (str, optional) – Title of the chat, if the title was requested by the bot.

  Added in version 21.1.
- ***username*** (str, optional) – Username of the chat, if the username was requested by the bot and available.

  Added in version 21.1.
- ***photo*** (Sequence[*telegram.PhotoSize*], optional) – Available sizes of the chat photo, if the photo was requested by the bot

  Added in version 21.1.

request_id

Identifier of the request.

Type
int

chat_id

Identifier of the shared user. 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
int

title

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

Added in version 21.1.

Type
str

username

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

Added in version 21.1.

Type
str

photo

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

Added in version 21.1.

Type
tuple[*telegram.PhotoSize*]

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

property link

Convenience property. If *username* is available, returns a t.me link of the chat.

Added in version 22.4.

Type
str

Contact

class telegram.**Contact**(*phone_number*, *first_name*, *last_name=None*, *user_id=None*, *vcard=None*, ***, *api_kwargs=None*)

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.

Use In

telegram.Bot.send_contact()

Available In

- *telegram.ExternalReplyInfo.contact*

- *telegram.Message.contact*
- *telegram.Message.effective_attachment*

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.

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

Dice

class telegram.**Dice**(*value*, *emoji*, *, *api_kwargs=None*)

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 in our wiki. However, this behaviour is undocumented and might be changed by Telegram.

Available In

- *telegram.ExternalReplyInfo.dice*
- *telegram.Message.dice*
- *telegram.Message.effective_attachment*

Parameters

- ***value*** (int) – Value of the dice. *1-6* for '', '' and '' base emoji, *1-5* for '' and '' base emoji, *1-64* for '' base emoji.
- ***emoji*** (str) – Emoji on which the dice throw animation is based.

value

Value of the dice. *1-6* for '', '' and '' base emoji, *1-5* for '' and '' base emoji, *1-64* for '' base emoji.

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

Added in version 13.4.

DARTS = ''

telegram.constants.DiceEmoji.DARTS

DICE = ''

telegram.constants.DiceEmoji.DICE

FOOTBALL = ''

telegram.constants.DiceEmoji.FOOTBALL

MAX_VALUE_BASKETBALL = 5

telegram.constants.DiceLimit.MAX_VALUE_BASKETBALL

Added in version 20.0.

MAX_VALUE_BOWLING = 6

telegram.constants.DiceLimit.MAX_VALUE_BOWLING

Added in version 20.0.

MAX_VALUE_DARTS = 6

telegram.constants.DiceLimit.MAX_VALUE_DARTS

Added in version 20.0.

MAX_VALUE_DICE = 6

telegram.constants.DiceLimit.MAX_VALUE_DICE

Added in version 20.0.

MAX_VALUE_FOOTBALL = 5

telegram.constants.DiceLimit.MAX_VALUE_FOOTBALL

Added in version 20.0.

MAX_VALUE_SLOT_MACHINE = 64

telegram.constants.DiceLimit.MAX_VALUE_SLOT_MACHINE

Added in version 20.0.

MIN_VALUE = 1

telegram.constants.DiceLimit.MIN_VALUE

Added in version 20.0.

SLOT_MACHINE = '🎰'

telegram.constants.DiceEmoji.SLOT_MACHINE

DirectMessagePriceChanged

class telegram.**DirectMessagePriceChanged**(*are_direct_messages_enabled*, *direct_message_star_count=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a service message about a change in the price of direct messages sent to a channel chat.

Available In

telegram.Message.direct_message_price_changed

Added in version 22.3.

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

Parameters

- ***are_direct_messages_enabled*** (`bool`) – `True`, if direct messages are enabled for the channel chat; `False` otherwise.
- ***direct_message_star_count*** (`int`, optional) – The new number of Telegram Stars that must be paid by users for each direct message sent to the channel. Does not apply to users who have been exempted by administrators. Defaults to `0`.

are_direct_messages_enabled

`True`, if direct messages are enabled for the channel chat; `False` otherwise.

Type
`bool`

direct_message_star_count

Optional. The new number of Telegram Stars that must be paid by users for each direct message sent to the channel. Does not apply to users who have been exempted by administrators. Defaults to `0`.

Type
`int`

DirectMessagesTopic

class telegram.**DirectMessagesTopic**(*topic_id*, *user=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This class represents a topic for direct messages in a chat.

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

Available In

telegram.Message.direct_messages_topic

Added in version 22.4.

Parameters

- ***topic_id*** (`int`) – Unique identifier of the topic. This number may have more than 32 significant bits and some programming languages may have difficulty/silent defects in interpreting it. But it has at most 52 significant bits, so a 64-bit integer or double-precision float type are safe for storing this identifier.
- ***user*** (*telegram.User*, optional) – Information about the user that created the topic.

  Hint

  According to Telegram, this field is always present as of Bot API 9.2.

topic_id

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

Type
`int`

user

Optional. Information about the user that created the topic.

Hint

According to Telegram, this field is always present as of Bot API 9.2.

Type
telegram.User

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

Document

class telegram.**Document**(*file_id*, *file_unique_id*, *file_name=None*, *mime_type=None*, *file_size=None*, *thumbnail=None*, ***, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.get_file()*
- *telegram.Bot.send_document()*

Available In

- *telegram.BackgroundTypePattern.document*
- *telegram.BackgroundTypeWallpaper.document*
- *telegram.ExternalReplyInfo.document*
- *telegram.InputMediaDocument.media*
- *telegram.Message.document*
- *telegram.Message.effective_attachment*
- *telegram.PollMedia.document*

Changed in version 20.5: Removed the deprecated argument and attribute `thumb`.

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_name*** (`str`, optional) – Original filename as defined by the sender.
- ***mime_type*** (`str`, optional) – MIME type of the file as defined by the sender.
- ***file_size*** (`int`, optional) – File size in bytes.
- ***thumbnail*** (*telegram.PhotoSize*, optional) – Document thumbnail as defined by the sender.

  Added in version 20.2.

file_id

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

Optional. Original filename as defined by the sender.

Type
str

mime_type

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

Type
str

file_size

Optional. File size in bytes.

Type
int

thumbnail

Optional. Document thumbnail as defined by the sender.

Added in version 20.2.

Type
telegram.PhotoSize

classmethod de_json(*data*, *bot=None*)

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 –

ExternalReplyInfo

class telegram.**ExternalReplyInfo**(*origin*, *chat=None*, *message_id=None*, *link_preview_options=None*, *animation=None*, *audio=None*, *document=None*, *photo=None*, *sticker=None*, *story=None*, *video=None*, *video_note=None*, *voice=None*, *has_media_spoiler=None*, *contact=None*, *dice=None*, *game=None*, *giveaway=None*, *giveaway_winners=None*, *invoice=None*, *location=None*, *poll=None*, *venue=None*, *paid_media=None*, *checklist=None*, *live_photo=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

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

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

Available In

telegram.Message.external_reply

Added in version 20.8.

Parameters

- ***origin*** (*telegram.MessageOrigin*) – Origin of the message replied to by the given message.
- ***chat*** (*telegram.Chat*, optional) – Chat the original message belongs to. Available only if the chat is a supergroup or a channel.
- ***message_id*** (`int`, optional) – Unique message identifier inside the original chat. Available only if the original chat is a supergroup or a channel.
- ***link_preview_options*** (*telegram.LinkPreviewOptions*, optional) – Options used for link preview generation for the original message, if it is a text message
- ***animation*** (*telegram.Animation*, optional) – Message is an animation, information about the animation.
- ***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.
- ***photo*** (Sequence[*telegram.PhotoSize*], optional) – Message is a photo, available sizes of the photo.
- ***sticker*** (*telegram.Sticker*, optional) – Message is a sticker, information about the sticker.
- ***story*** (*telegram.Story*, optional) – Message is a forwarded story.
- ***video*** (*telegram.Video*, optional) – Message is a video, information about the video.
- ***video_note*** (*telegram.VideoNote*, optional) – Message is a video note, information about the video message.
- ***voice*** (*telegram.Voice*, optional) – Message is a voice message, information about the file.
- ***has_media_spoiler*** (`bool`, optional) – `True`, if the message media is covered by a spoiler animation.
- ***checklist*** (*telegram.Checklist*, optional) – Message is a checklist.

  Added in version 22.3.
- ***contact*** (*telegram.Contact*, optional) – Message is a shared contact, information about the contact.
- ***dice*** (*telegram.Dice*, optional) – Message is a dice with random value.
- ***game*** (*telegram.Game*. optional) – Message is a game, information about the game. *More about games >>*.
- ***giveaway*** (*telegram.Giveaway*, optional) – Message is a scheduled giveaway, information about the giveaway.
- ***giveaway_winners*** (*telegram.GiveawayWinners*, optional) – A giveaway with public winners was completed.
- ***invoice*** (*telegram.Invoice*, optional) – Message is an invoice for a payment, information about the invoice. *More about payments >>*.

- ***location*** (*telegram.Location*, optional) – Message is a shared location, information about the location.
- ***poll*** (*telegram.Poll*, optional) – Message is a native poll, information about the poll.
- ***venue*** (*telegram.Venue*, optional) – Message is a venue, information about the venue.
- ***paid_media*** (*telegram.PaidMedia*, optional) – Message contains paid media; information about the paid media.

  Added in version 21.4.
- ***live_photo*** (*telegram.LivePhoto*, optional) – Message is a live photo, information about the live photo.

  Added in version 22.8.

origin

Origin of the message replied to by the given message.

Type
telegram.MessageOrigin

chat

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

Type
telegram.Chat

message_id

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

Type
int

link_preview_options

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

Type
telegram.LinkPreviewOptions

animation

Optional. Message is an animation, information about the animation.

Type
telegram.Animation

audio

Optional. Message is an audio file, information about the file.

Type
telegram.Audio

document

Optional. Message is a general file, information about the file.

Type
telegram.Document

photo

Optional. Message is a photo, available sizes of the photo.

Type
tuple[*telegram.PhotoSize*]

sticker

Optional. Message is a sticker, information about the sticker.

Type
telegram.Sticker

story

Optional. Message is a forwarded story.

Type
telegram.Story

video

Optional. Message is a video, information about the video.

Type
telegram.Video

video_note

Optional. Message is a video note, information about the video message.

Type
telegram.VideoNote

voice

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

Type
telegram.Voice

has_media_spoiler

Optional. `True`, if the message media is covered by a spoiler animation.

Type
`bool`

checklist

Optional. Message is a checklist.

Added in version 22.3.

Type
telegram.Checklist

contact

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

Type
telegram.Contact

dice

Optional. Message is a dice with random value.

Type
telegram.Dice

game

Optional. Message is a game, information about the game. *More about games >>*.

Type
telegram.Game

giveaway

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

Type
telegram.Giveaway

giveaway_winners

Optional. A giveaway with public winners was completed.

Type
telegram.GiveawayWinners

invoice

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

Type
telegram.Invoice

location

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

Type
telegram.Location

poll

Optional. Message is a native poll, information about the poll.

Type
telegram.Poll

venue

Optional. Message is a venue, information about the venue.

Type
telegram.Venue

paid_media

Optional. Message contains paid media; information about the paid media.

Added in version 21.4.

Type
telegram.PaidMedia

live_photo

Optional. Message is a live photo, information about the live photo.

Added in version 22.8.

Type
telegram.LivePhoto

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

File

class telegram.**File**(*file_id*, *file_unique_id*, *file_size=None*, *file_path=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a file ready to be downloaded. The file can be e.g. downloaded with *download_to_drive*. 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.

Available In

telegram.Sticker.premium_animation

Returned In

- *telegram.Bot.get_file()*
- *telegram.Bot.upload_sticker_file()*

Changed in version 20.0: `download` was split into *download_to_drive()* and *download_to_memory()*.

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 e.g. *download_to_drive()*.

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) – File size in bytes, if known.
- ***file_path*** (`str`, optional) – File path. Use e.g. *download_to_drive()* to get the file.

file_id

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

Type
`int`

file_path

Optional. File path. Use e.g. *download_to_drive()* to get the file.

Type
`str`

async download_as_bytearray(*buf=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*)

Download this file and return it as a bytearray.

Changed in version 21.7: Raises `RuntimeError` if *file_path* is not set. Note that files without a *file_path* could never be downloaded, as this attribute is mandatory for that operation.

Parameters

buf (`bytearray`, optional) – Extend the given bytearray with the downloaded data.

Keyword Arguments

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

  Added in version 20.0.

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

  Added in version 20.0.

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

  Added in version 20.0.

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

  Added in version 20.0.

Returns

The same object as *buf* if it was specified. Otherwise a newly allocated `bytearray`.

Return type

`bytearray`

Raises

RuntimeError – If *file_path* is not set.

async download_to_drive(*custom_path=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 *file_path* as file name. If *custom_path* is supplied as a `str` or `pathlib.Path`, it will be saved to that path.

Note

If *custom_path* isn't 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.

The only exception to this are encrypted files (e.g. a passport file). For these, a file with the prefix *decrypted_* will be created in the same directory as the original file in order to decrypt the file without changing the existing one in-place.

See also

Working with Files and Media

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.

- This method was previously called `download`. It was split into *download_to_drive()* and *download_to_memory()*.

Changed in version 21.7: Raises `RuntimeError` if *file_path* is not set. Note that files without a *file_path* could never be downloaded, as this attribute is mandatory for that operation.

Parameters

custom_path (`pathlib.Path` | `str` , optional) – The path where the file will be saved to. If not specified, will be saved in the current working directory with *file_path* as file name or the *file_id* if *file_path* is not 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*.

Returns

Returns the Path object the file was downloaded to.

Return type

`pathlib.Path`

Raises

RuntimeError – If *file_path* is not set.

async download_to_memory(*out*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*)

Download this file into memory. *out* needs to be supplied with a `io.BufferedIOBase`, the file contents will be saved to that object using the `out.write` method.

See also

Working with Files and Media

Hint

If you want to immediately read the data from `out` after calling this method, you should call `out.seek(0)` first. See also `io.IOBase.seek()`.

Added in version 20.0.

Changed in version 21.7: Raises `RuntimeError` if *file_path* is not set. Note that files without a *file_path* could never be downloaded, as this attribute is mandatory for that operation.

Parameters

out (`io.BufferedIOBase`) – A file-like object. Must be opened for writing in binary mode.

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*.

Raises

RuntimeError – If *file_path* is not set.

set_credentials(*credentials*)

Sets the passport credentials for the file.

Parameters

credentials (*telegram.FileCredentials*) – The credentials.

ForceReply

class telegram.**ForceReply**(*selective=None*, *input_field_placeholder=None*, ***, *api_kwargs=None*)

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. Not supported in channels and for messages sent on behalf of a user account.

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

Use In

- *telegram.Bot.copy_message()*
- *telegram.Bot.send_animation()*
- *telegram.Bot.send_audio()*
- *telegram.Bot.send_contact()*
- *telegram.Bot.send_dice()*
- *telegram.Bot.send_document()*
- *telegram.Bot.send_live_photo()*
- *telegram.Bot.send_location()*
- *telegram.Bot.send_message()*
- *telegram.Bot.send_paid_media()*
- *telegram.Bot.send_photo()*
- *telegram.Bot.send_poll()*
- *telegram.Bot.send_sticker()*
- *telegram.Bot.send_venue()*
- *telegram.Bot.send_video_note()*
- *telegram.Bot.send_video()*
- *telegram.Bot.send_voice()*

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

  Added in version 13.7.

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. Targets:

1) Users that are @mentioned in the *text* of the *telegram.Message* object.

2)**If the bot's message is a reply to a message in the same chat and forum topic,** sender of the original message.

Type
`bool`

input_field_placeholder

Optional. The placeholder to be shown in the input field when the reply is active; *1- 64* characters.

Added in version 13.7.

Type
`str`

MAX_INPUT_FIELD_PLACEHOLDER = 64

telegram.constants.ReplyLimit.MAX_INPUT_FIELD_PLACEHOLDER

Added in version 20.0.

MIN_INPUT_FIELD_PLACEHOLDER = 1

telegram.constants.ReplyLimit.MIN_INPUT_FIELD_PLACEHOLDER

Added in version 20.0.

ForumTopic

class telegram.**ForumTopic**(*message_thread_id*, *name*, *icon_color*, *icon_custom_emoji_id=None*, *is_name_implicit=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a forum topic.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *message_thread_id*, *name* and *icon_color* are equal.

Returned In

telegram.Bot.create_forum_topic()

Added in version 20.0.

Parameters

- ***message_thread_id*** (int) – Unique identifier of the forum topic
- ***name*** (str) – Name of the topic
- ***icon_color*** (int) – Color of the topic icon in RGB format
- ***icon_custom_emoji_id*** (str, optional) – Unique identifier of the custom emoji shown as the topic icon.
- ***is_name_implicit*** (bool, optional) – True, if the name of the topic wasn't specified explicitly by its creator and likely needs to be changed by the bot.

  Added in version 22.6.

message_thread_id

Unique identifier of the forum topic

Type
int

name

Name of the topic

Type
str

icon_color

Color of the topic icon in RGB format

Type
int

icon_custom_emoji_id

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

Type
str

is_name_implicit

Optional. True, if the name of the topic wasn't specified explicitly by its creator and likely needs to be changed by the bot.

Added in version 22.6.

Type
bool

ForumTopicClosed

class telegram.**ForumTopicClosed**(*, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a service message about a forum topic closed in the chat. Currently holds no information.

Available In

telegram.Message.forum_topic_closed

Added in version 20.0.

ForumTopicCreated

class telegram.**ForumTopicCreated**(*name*, *icon_color*, *icon_custom_emoji_id=None*, *is_name_implicit=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

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

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

Available In

telegram.Message.forum_topic_created

Added in version 20.0.

Parameters

- ***name*** (str) – Name of the topic
- ***icon_color*** (int) – Color of the topic icon in RGB format
- ***icon_custom_emoji_id*** (str, optional) – Unique identifier of the custom emoji shown as the topic icon.
- ***is_name_implicit*** (bool, optional) – True, if the name of the topic wasn't specified explicitly by its creator and likely needs to be changed by the bot.

  Added in version 22.6.

name

Name of the topic

Type
str

icon_color

Color of the topic icon in RGB format

Type
int

icon_custom_emoji_id

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

Type
str

is_name_implicit

Optional. True, if the name of the topic wasn't specified explicitly by its creator and likely needs to be changed by the bot.

Added in version 22.6.

Type
bool

ForumTopicEdited

class telegram.**ForumTopicEdited**(*name=None*, *icon_custom_emoji_id=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

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

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

Available In

telegram.Message.forum_topic_edited

Added in version 20.0.

Parameters

- ***name*** (str, optional) – New name of the topic, if it was edited.
- ***icon_custom_emoji_id*** (str, optional) – New identifier of the custom emoji shown as the topic icon, if it was edited; an empty string if the icon was removed.

name

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

Type
str

icon_custom_emoji_id

Optional. New identifier of the custom emoji shown as the topic icon, if it was edited; an empty string if the icon was removed.

Type
str

ForumTopicReopened

class telegram.**ForumTopicReopened**(***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a service message about a forum topic reopened in the chat. Currently holds no information.

Available In

telegram.Message.forum_topic_reopened

Added in version 20.0.

GeneralForumTopicHidden

class telegram.**GeneralForumTopicHidden**(***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a service message about General forum topic hidden in the chat. Currently holds no information.

Available In

telegram.Message.general_forum_topic_hidden

Added in version 20.0.

GeneralForumTopicUnhidden

class telegram.**GeneralForumTopicUnhidden**(*, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a service message about General forum topic unhidden in the chat. Currently holds no information.

Available In

telegram.Message.general_forum_topic_unhidden

Added in version 20.0.

GiftBackground

class telegram.**GiftBackground**(*center_color*, *edge_color*, *text_color*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object describes the background of a gift.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal if their *center_color*, *edge_color* and *text_color* are equal.

Available In

telegram.Gift.background

Added in version 22.6.

Parameters

- ***center_color*** (int) – Center color of the background in RGB format.
- ***edge_color*** (int) – Edge color of the background in RGB format.
- ***text_color*** (int) – Text color of the background in RGB format.

center_color

Center color of the background in RGB format.

Type
int

edge_color

Edge color of the background in RGB format.

Type
int

text_color

Text color of the background in RGB format.

Type
int

GiftInfo

class telegram.**GiftInfo**(*gift*, *owned_gift_id=None*, *convert_star_count=None*, *prepaid_upgrade_star_count=None*, *can_be_upgraded=None*, *text=None*, *entities=None*, *is_private=None*, *unique_gift_number=None*, *is_upgrade_separate=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a service message about a regular gift that was sent or received.

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

Available In

- *telegram.Message.gift_upgrade_sent*
- *telegram.Message.gift*

Added in version 22.1.

Parameters

- ***gift*** (*Gift*) – Information about the gift.
- ***owned_gift_id*** (str, optional) – Unique identifier of the received gift for the bot; only present for gifts received on behalf of business accounts.
- ***convert_star_count*** (int, optional) – the receiver by converting the gift; omitted if conversion to Telegram Stars is impossible.
- ***prepaid_upgrade_star_count*** (int, optional) – Number of Telegram Stars that were prepaid for the ability to upgrade the gift.
- ***can_be_upgraded*** (bool, optional) – True, if the gift can be upgraded to a unique gift.
- ***text*** (str, optional) – Text of the message that was added to the gift.
- ***entities*** (Sequence[*telegram.MessageEntity*], optional) – Special entities that appear in the text.
- ***is_private*** (bool, optional) – True, if the sender and gift text are shown only to the gift receiver; otherwise, everyone will be able to see them.
- ***is_upgrade_separate*** (bool, optional) – True, if the gift's upgrade was purchased after the gift was sent.

  Added in version 22.6.
- ***unique_gift_number*** (int, optional) – Unique number reserved for this gift when upgraded. See the number field in *UniqueGift*.

  Added in version 22.6.

gift

Information about the gift.

Type
Gift

owned_gift_id

Optional. Unique identifier of the received gift for the bot; only present for gifts received on behalf of business accounts.

Type
str

convert_star_count

Optional. Number of Telegram Stars that can be claimed by the receiver by converting the gift; omitted if conversion to Telegram Stars is impossible.

Type
int

prepaid_upgrade_star_count

Optional. Number of Telegram Stars that were prepaid for the ability to upgrade the gift.

Type
int

can_be_upgraded

Optional. True, if the gift can be upgraded to a unique gift.

Type
bool

text

Optional. Text of the message that was added to the gift.

Type
str

entities

Optional. Special entities that appear in the text.

Type
Sequence[*telegram.MessageEntity*]

is_private

Optional. True, if the sender and gift text are shown only to the gift receiver; otherwise, everyone will be able to see them.

Type
bool

is_upgrade_separate

Optional. True, if the gift's upgrade was purchased after the gift was sent.

Added in version 22.6.

Type
bool

unique_gift_number

Optional. Unique number reserved for this gift when upgraded. See the number field in *UniqueGift*.

Added in version 22.6.

Type
int

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

parse_entities(*types=None*)

Returns a dict that maps *telegram.MessageEntity* to str. It contains entities from this gift info's text 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 *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 `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]

Raises
RuntimeError – If the gift info has no text.

`parse_entity`(*entity*)

Returns the text in *text* from a given *telegram.MessageEntity* of *entities*.

> **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 *entities*.

Returns
The text of the given entity.

Return type
str

Raises
RuntimeError – If the gift info has no text.

Giveaway

`class telegram.Giveaway`(*chats*, *winners_selection_date*, *winner_count*, *only_new_members=None*, *has_public_winners=None*, *prize_description=None*, *country_codes=None*, *premium_subscription_month_count=None*, *prize_star_count=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a message about a scheduled giveaway.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *chats*, *winners_selection_date* and *winner_count* are equal.

> **Available In**

- *telegram.ExternalReplyInfo.giveaway*
- *telegram.Message.giveaway*

Added in version 20.8.

Parameters

- ***chats*** (tuple[*telegram.Chat*]) – The list of chats which the user must join to participate in the giveaway.
- ***winners_selection_date*** (datetime.datetime) – The date when the giveaway winner will be selected. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***winner_count*** (int) – The number of users which are supposed to be selected as winners of the giveaway.
- ***only_new_members*** (True, optional) – If True, only users who join the chats after the giveaway started should be eligible to win.
- ***has_public_winners*** (True, optional) – True, if the list of giveaway winners will be visible to everyone
- ***prize_description*** (str, optional) – Description of additional giveaway prize
- ***country_codes*** (Sequence[str]) – A list of two-letter ISO 3166-1 alpha-2 country codes indicating the countries from which eligible users for the giveaway must come. If empty, then all users can participate in the giveaway. Users with a phone number that was bought on Fragment can always participate in giveaways.
- ***prize_star_count*** (int, optional) – The number of Telegram Stars to be split between giveaway winners; for Telegram Star giveaways only.

  Added in version 21.6.
- ***premium_subscription_month_count*** (int, optional) – The number of months the Telegram Premium subscription won from the giveaway will be active for; for Telegram Premium giveaways only.

chats

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

Type
Sequence[*telegram.Chat*]

winners_selection_date

The date when the giveaway winner will be selected. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

winner_count

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

Type
int

only_new_members

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

Type
True

has_public_winners

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

Type
True

prize_description

Optional. Description of additional giveaway prize

Type
str

country_codes

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

Type
tuple[str]

prize_star_count

Optional. The number of Telegram Stars to be split between giveaway winners; for Telegram Star giveaways only.

Added in version 21.6.

Type
int

premium_subscription_month_count

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

Type
int

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

GiveawayCompleted

class telegram.**GiveawayCompleted**(*winner_count*, *unclaimed_prize_count=None*, *giveaway_message=None*, *is_star_giveaway=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

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

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

Available In

telegram.Message.giveaway_completed

Added in version 20.8.

Parameters

- ***winner_count*** (int) – Number of winners in the giveaway
- ***unclaimed_prize_count*** (int, optional) – Number of undistributed prizes

- ***giveaway_message*** (*telegram.Message*, optional) – Message with the giveaway that was completed, if it wasn't deleted
- ***is_star_giveaway*** (bool, optional) – True, if the giveaway is a Telegram Star giveaway. Otherwise, currently, the giveaway is a Telegram Premium giveaway.

  Added in version 21.6.

winner_count

Number of winners in the giveaway

Type
int

unclaimed_prize_count

Optional. Number of undistributed prizes

Type
int

giveaway_message

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

Type
telegram.Message

is_star_giveaway

Optional. True, if the giveaway is a Telegram Star giveaway. Otherwise, currently, the giveaway is a Telegram Premium giveaway.

Added in version 21.6.

Type
bool

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

GiveawayCreated

class telegram.**GiveawayCreated**(*prize_star_count=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a service message about the creation of a scheduled giveaway.

Available In

telegram.Message.giveaway_created

Parameters
prize_star_count (int, optional) – The number of Telegram Stars to be split between giveaway winners; for Telegram Star giveaways only.

Added in version 21.6.

prize_star_count

Optional. The number of Telegram Stars to be split between giveaway winners; for Telegram Star giveaways only.

Added in version 21.6.

Type
int

GiveawayWinners

class telegram.**GiveawayWinners**(*chat*, *giveaway_message_id*, *winners_selection_date*, *winner_count*, *winners*, *additional_chat_count=None*, *premium_subscription_month_count=None*, *unclaimed_prize_count=None*, *only_new_members=None*, *was_refunded=None*, *prize_description=None*, *prize_star_count=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *chat*, *giveaway_message_id*, *winners_selection_date*, *winner_count* and *winners* are equal.

Available In

- *telegram.ExternalReplyInfo.giveaway_winners*
- *telegram.Message.giveaway_winners*

Added in version 20.8.

Parameters

- ***chat*** (*telegram.Chat*) – The chat that created the giveaway
- ***giveaway_message_id*** (int) – Identifier of the message with the giveaway in the chat
- ***winners_selection_date*** (datetime.datetime) – Point in time when winners of the giveaway were selected. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***winner_count*** (int) – Total number of winners in the giveaway
- ***winners*** (Sequence[*telegram.User*]) – List of up to *100* winners of the giveaway
- ***prize_star_count*** (int, optional) – The number of Telegram Stars to be split between giveaway winners; for Telegram Star giveaways only.

  Added in version 21.6.
- ***additional_chat_count*** (int, optional) – The number of other chats the user had to join in order to be eligible for the giveaway
- ***premium_subscription_month_count*** (int, optional) – The number of months the Telegram Premium subscription won from the giveaway will be active for
- ***unclaimed_prize_count*** (int, optional) – Number of undistributed prizes
- ***only_new_members*** (True, optional) – True, if only users who had joined the chats after the giveaway started were eligible to win
- ***was_refunded*** (True, optional) – True, if the giveaway was canceled because the payment for it was refunded
- ***prize_description*** (str, optional) – Description of additional giveaway prize

chat

The chat that created the giveaway

Type

telegram.Chat

giveaway_message_id

Identifier of the message with the giveaway in the chat

Type
int

winners_selection_date

Point in time when winners of the giveaway were selected. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

winner_count

Total number of winners in the giveaway

Type
int

winners

tuple of up to *100* winners of the giveaway

Type
tuple[*telegram.User*]

additional_chat_count

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

Type
int

prize_star_count

Optional. The number of Telegram Stars to be split between giveaway winners; for Telegram Star giveaways only.

Added in version 21.6.

Type
int

premium_subscription_month_count

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

Type
int

unclaimed_prize_count

Optional. Number of undistributed prizes

Type
int

only_new_members

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

Type
True

was_refunded

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

Type
True

prize_description

Optional. Description of additional giveaway prize

Type
str

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

InaccessibleMessage

class telegram.**InaccessibleMessage**(*chat*, *message_id*, *, *api_kwargs=None*)

Bases: *telegram.MaybeInaccessibleMessage*

This object represents an inaccessible message.

These are messages that are e.g. deleted.

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

Available In

- *telegram.CallbackQuery.message*
- *telegram.Message.pinned_message*
- *telegram.PollOptionAdded.poll_message*
- *telegram.PollOptionDeleted.poll_message*

Added in version 20.8.

Parameters

- ***message_id*** (int) – Unique message identifier.
- ***chat*** (*telegram.Chat*) – Chat the message belongs to.

message_id

Unique message identifier.

Type
int

date

Always *datetime.datetime(1970, 1, 1, 0, 0, tzinfo=datetime.timezone.utc)*. The field can be used to differentiate regular and inaccessible messages.

Type
constants.ZERO_DATE

chat

Chat the message belongs to.

Type
telegram.Chat

InlineKeyboardButton

class telegram.**InlineKeyboardButton**(*text*, *url=None*, *callback_data=None*, *switch_inline_query=None*, *switch_inline_query_current_chat=None*, *callback_game=None*, *pay=None*, *login_url=None*, *web_app=None*, *switch_inline_query_chosen_chat=None*, *copy_text=None*, *style=None*, *icon_custom_emoji_id=None*, *, *api_kwargs=None*)

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*, *pay*, *style* and *icon_custom_emoji_id* are equal.

Note

- Exactly one of the fields other than *text*, *icon_custom_emoji_id*, and *style* must be used to specify type of the button.
- 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* may be an instance of *telegram.ext.InvalidCallbackData*. This will be the case, if the data associated with the button was already deleted.

  Added 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*.
- *style* option will only work in Telegram versions released after February 9, 2026. Older clients will display buttons without styling.

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*.

Examples

- *Inline Keyboard 1*
- *Inline Keyboard 2*

See also

telegram.InlineKeyboardMarkup

Available In

telegram.InlineKeyboardMarkup.inline_keyboard

Changed in version 20.0: *web_app* is considered as well when comparing objects of this type in terms of equality.

Changed in version 22.7: *style* and *icon_custom_emoji_id* are 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 the 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*.

  See also

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

  Added 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. May be empty, in which case just the bot's username will be inserted. Not supported for messages sent on behalf of a Telegram Business account.

  Tip

This is similar to the parameter *switch_inline_query_chosen_chat*, but gives no control over which chats can be selected.

- ***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. May 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. Not supported in channels and for messages sent on behalf of a Telegram Business account.

- ***copy_text*** (*telegram.CopyTextButton*, optional) – Description of the button that copies the specified text to the clipboard.

  Added in version 21.7.

- ***callback_game*** (*telegram.CallbackGame*, optional) – Description of the game that will be launched when the user presses the button

  Note

  This type of button **must** always be the first button in the first row.

- ***pay*** (bool, optional) – Specify True, to send a Pay button. Substrings "" and "XTR" in the buttons's text will be replaced with a Telegram Star icon.

  Note

  This type of button **must** always be the first button in the first row and can only be used in invoice messages.

- ***style*** (str, optional) – Style of the button. Must be one of *'primary'* (blue), *'success'* (green), and *'danger'* (red). Color name aliases *'primary'*, *'success'*, and *'danger'* are also available. If omitted, then an app-specific style is used.

  Added in version 22.7.

- ***switch_inline_query_chosen_chat*** (*telegram.SwitchInlineQueryChosenChat*, optional) – If set, pressing the button will prompt the user to select one of their chats of the specified type, open that chat and insert the bot's username and the specified inline query in the input field. Not supported for messages sent on behalf of a Telegram Business account.

  Added in version 20.3.

  Tip

  This is similar to *switch_inline_query*, but gives more control on which chats can be selected.

> **Caution**
>
> The PTB team has discovered that this field works correctly only if your Telegram client is released after April 20th 2023.

- ***icon_custom_emoji_id*** (str, optional) – Unique identifier of the custom emoji shown before the text of the button. Can only be used by bots that purchased additional usernames on Fragment or in the messages directly sent by the bot to private, group and supergroup chats if the owner of the bot has a Telegram Premium subscription.

  Added in version 22.7.

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

Added in version 20.0.

Type
telegram.WebAppInfo

switch_inline_query

Optional. If set, pressing the button will prompt the user to select one of their chats, open that chat and insert the bot's username and the specified inline query in the input field. May be empty, in which case just the bot's username will be inserted. Not supported for messages sent on behalf of a Telegram Business account.

> **Tip**
>
> This is similar to the parameter *switch_inline_query_chosen_chat*, but gives no control over which chats can be selected.

Type
str

switch_inline_query_current_chat

Optional. If set, pressing the button will insert the bot's username and the specified inline query in the current chat's input field. May be empty, in which case only the bot's username will be inserted.

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. Not supported in channels and for messages sent on behalf of a Telegram Business account.

Type
str

copy_text

Optional. Description of the button that copies the specified text to the clipboard.

Added in version 21.7.

Type
telegram.CopyTextButton

style

Optional. Style of the button. Must be one of *'primary'* (blue), *'success'* (green), and *'danger'* (red). Color name aliases *'primary'*, *'success'*, and *'danger'* are also available. If omitted, then an app-specific style is used.

Added in version 22.7.

Type
str

callback_game

Optional. Description of the game that will be launched when the user presses the button.

> **Note**
>
> This type of button **must** always be the first button in the first row.

Type
telegram.CallbackGame

pay

Optional. Specify `True`, to send a Pay button. Substrings "" and "XTR" in the buttons's text will be replaced with a Telegram Star icon.

> **Note**
>
> This type of button **must** always be the first button in the first row and can only be used in invoice messages.

Type
bool

switch_inline_query_chosen_chat

Optional. If set, pressing the button will prompt the user to select one of their chats of the specified type, open that chat and insert the bot's username and the specified inline query in the input field. Not supported for messages sent on behalf of a Telegram Business account.

Added in version 20.3.

> **Tip**
>
> This is similar to *switch_inline_query*, but gives more control on which chats can be selected.

> **Caution**
>
> The PTB team has discovered that this field works correctly only if your Telegram client is released after April 20th 2023.

Type
telegram.SwitchInlineQueryChosenChat

icon_custom_emoji_id

Optional. Unique identifier of the custom emoji shown before the text of the button. Can only be used by bots that purchased additional usernames on Fragment or in the messages directly sent by the bot to private, group and supergroup chats if the owner of the bot has a Telegram Premium subscription.

Added in version 22.7.

Type
str

MAX_CALLBACK_DATA = 64

telegram.constants.InlineKeyboardButtonLimit.MAX_CALLBACK_DATA

Added in version 20.0.

MIN_CALLBACK_DATA = 1

telegram.constants.InlineKeyboardButtonLimit.MIN_CALLBACK_DATA

Added in version 20.0.

classmethod de_json(*data*, *bot=None*)

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*.

Added in version 13.6.

Parameters
callback_data (object) – The new callback data.

InlineKeyboardMarkup

class telegram.**InlineKeyboardMarkup**(*inline_keyboard*, ***, *api_kwargs=None*)

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.

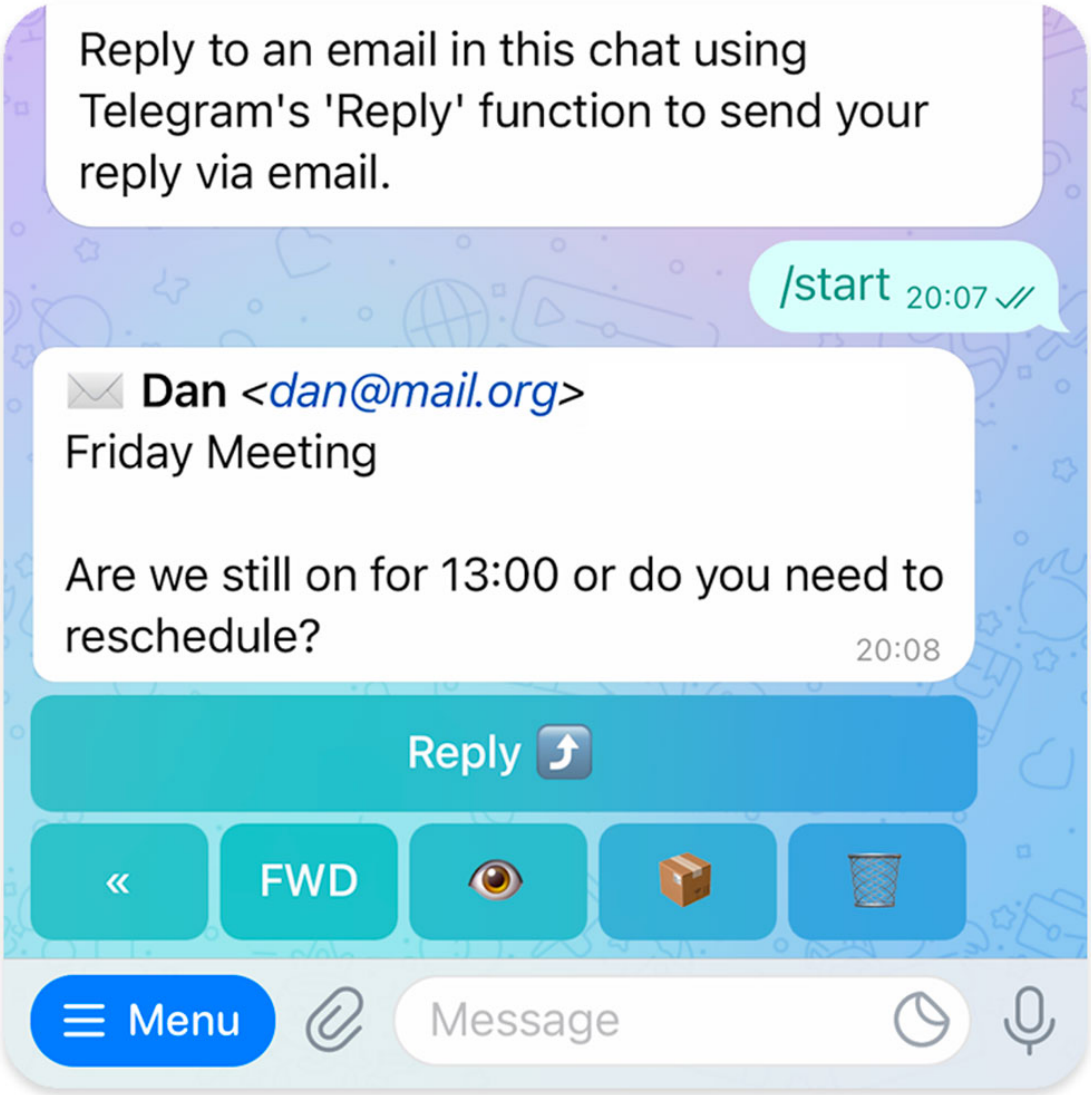

Fig. 1: An inline keyboard on a message

See also

Another kind of keyboard would be the *telegram.ReplyKeyboardMarkup*.

Examples

- *Inline Keyboard 1*
- *Inline Keyboard 2*

Use In

- *telegram.Bot.copy_message()*
- *telegram.Bot.edit_message_caption()*
- *telegram.Bot.edit_message_checklist()*
- *telegram.Bot.edit_message_live_location()*
- *telegram.Bot.edit_message_media()*
- *telegram.Bot.edit_message_reply_markup()*
- *telegram.Bot.edit_message_text()*
- *telegram.Bot.send_animation()*
- *telegram.Bot.send_audio()*
- *telegram.Bot.send_checklist()*
- *telegram.Bot.send_contact()*
- *telegram.Bot.send_dice()*
- *telegram.Bot.send_document()*
- *telegram.Bot.send_game()*
- *telegram.Bot.send_invoice()*
- *telegram.Bot.send_live_photo()*
- *telegram.Bot.send_location()*
- *telegram.Bot.send_message()*
- *telegram.Bot.send_paid_media()*
- *telegram.Bot.send_photo()*
- *telegram.Bot.send_poll()*
- *telegram.Bot.send_sticker()*
- *telegram.Bot.send_venue()*
- *telegram.Bot.send_video_note()*
- *telegram.Bot.send_video()*
- *telegram.Bot.send_voice()*
- *telegram.Bot.stop_message_live_location()*
- *telegram.Bot.stop_poll()*

Available In

- *telegram.InlineQueryResultArticle.reply_markup*
- *telegram.InlineQueryResultAudio.reply_markup*
- *telegram.InlineQueryResultCachedAudio.reply_markup*
- *telegram.InlineQueryResultCachedDocument.reply_markup*
- *telegram.InlineQueryResultCachedGif.reply_markup*
- *telegram.InlineQueryResultCachedMpeg4Gif.reply_markup*
- *telegram.InlineQueryResultCachedPhoto.reply_markup*
- *telegram.InlineQueryResultCachedSticker.reply_markup*
- *telegram.InlineQueryResultCachedVideo.reply_markup*
- *telegram.InlineQueryResultCachedVoice.reply_markup*
- *telegram.InlineQueryResultContact.reply_markup*
- *telegram.InlineQueryResultDocument.reply_markup*
- *telegram.InlineQueryResultGame.reply_markup*
- *telegram.InlineQueryResultGif.reply_markup*
- *telegram.InlineQueryResultLocation.reply_markup*
- *telegram.InlineQueryResultMpeg4Gif.reply_markup*
- *telegram.InlineQueryResultPhoto.reply_markup*
- *telegram.InlineQueryResultVenue.reply_markup*
- *telegram.InlineQueryResultVideo.reply_markup*
- *telegram.InlineQueryResultVoice.reply_markup*
- *telegram.Message.reply_markup*

Parameters

inline_keyboard (Sequence[Sequence[*telegram.InlineKeyboardButton*]]) – Sequence of button rows, each represented by a sequence of *InlineKeyboardButton* objects.

Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list. The input is converted to a tuple.

inline_keyboard

Tuple of button rows, each represented by a tuple of *InlineKeyboardButton* objects.

Changed in version 20.0: This attribute is now an immutable tuple.

Type

tuple[tuple[*telegram.InlineKeyboardButton*]]

classmethod de_json(*data*, *bot=None*)

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

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 (Sequence[*telegram.InlineKeyboardButton*]) – The button to use in the markup

Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list.

classmethod from_row(*button_row*, ***kwargs*)

Shortcut for:

```
InlineKeyboardMarkup([button_row], **kwargs)
```

Return an InlineKeyboardMarkup from a single row of InlineKeyboardButtons

Parameters

button_row (Sequence[*telegram.InlineKeyboardButton*]) – The button to use in the markup

Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list.

InputChecklist

class telegram.**InputChecklist**(*title*, *tasks*, *parse_mode=None*, *title_entities=None*, *others_can_add_tasks=None*, *others_can_mark_tasks_as_done=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a checklist to create.

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

Use In

- *telegram.Bot.edit_message_checklist()*
- *telegram.Bot.send_checklist()*

Added in version 22.3.

Parameters

- ***title*** (str) – Title of the checklist; *1-255* characters after entities parsing.
- ***parse_mode*** (str, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***title_entities*** (Sequence[*telegram.MessageEntity*], optional) – List of special entities that appear in the title, which can be specified instead of *parse_mode*. Currently, only bold, italic, underline, strikethrough, spoiler, and custom_emoji, and date_time entities are allowed

- ***tasks*** (Sequence[*telegram.InputChecklistTask*]) – List of *1-30* tasks in the checklist.
- ***others_can_add_tasks*** (bool, optional) – Pass True if other users can add tasks to the checklist.
- ***others_can_mark_tasks_as_done*** (bool, optional) – Pass True if other users can mark tasks as done or not done in the checklist.

title

Title of the checklist; *1-255* characters after entities parsing.

Type
str

parse_mode

Optional. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

title_entities

Optional. List of special entities that appear in the title, which can be specified instead of *parse_mode*. Currently, only bold, italic, underline, strikethrough, spoiler, and custom_emoji, and date_time entities are allowed

Type
Sequence[*telegram.MessageEntity*]

tasks

List of *1-30* tasks in the checklist.

Type
Sequence[*telegram.InputChecklistTask*]

others_can_add_tasks

Optional. Pass True if other users can add tasks to the checklist.

Type
bool

others_can_mark_tasks_as_done

Optional. Pass True if other users can mark tasks as done or not done in the checklist.

Type
bool

InputChecklistTask

class telegram.**InputChecklistTask**(*id*, *text*, *parse_mode=None*, *text_entities=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a task to add to a checklist.

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

Available In

telegram.InputChecklist.tasks

Added in version 22.3.

Parameters

- ***id*** (int) – Unique identifier of the task; must be positive and unique among all task identifiers currently present in the checklist.
- ***text*** (str) – Text of the task; *1-100* characters after entities parsing.
- ***parse_mode*** (str, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***text_entities*** (Sequence[*telegram.MessageEntity*], optional) – List of special entities that appear in the text, which can be specified instead of parse_mode. Currently, only bold, italic, underline, strikethrough, spoiler, custom_emoji, and date_time entities are allowed.

id

Unique identifier of the task; must be positive and unique among all task identifiers currently present in the checklist.

Type
int

text

Text of the task; *1-100* characters after entities parsing.

Type
str

parse_mode

Optional. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

text_entities

Optional. List of special entities that appear in the text, which can be specified instead of parse_mode. Currently, only bold, italic, underline, strikethrough, spoiler, custom_emoji, and date_time entities are allowed.

Type
Sequence[*telegram.MessageEntity*]

InputFile

class telegram.**InputFile**(*obj*, *filename=None*, *attach=False*, *read_file_handle=True*)

Bases: object

This object represents a Telegram InputFile.

Use In

- *telegram.Bot.send_animation()*
- *telegram.Bot.send_audio()*
- *telegram.Bot.send_document()*

- *telegram.Bot.send_live_photo()*
- *telegram.Bot.send_photo()*
- *telegram.Bot.send_sticker()*
- *telegram.Bot.send_video_note()*
- *telegram.Bot.send_video()*
- *telegram.Bot.send_voice()*
- *telegram.Bot.set_chat_photo()*
- *telegram.Bot.set_sticker_set_thumbnail()*
- *telegram.Bot.set_webhook()*
- *telegram.Bot.upload_sticker_file()*

Available In

- *telegram.InputMedia.media*
- *telegram.InputMediaAnimation.media*
- *telegram.InputMediaAnimation.thumbnail*
- *telegram.InputMediaAudio.media*
- *telegram.InputMediaAudio.thumbnail*
- *telegram.InputMediaDocument.media*
- *telegram.InputMediaDocument.thumbnail*
- *telegram.InputMediaLivePhoto.media*
- *telegram.InputMediaLivePhoto.photo*
- *telegram.InputMediaPhoto.media*
- *telegram.InputMediaSticker.media*
- *telegram.InputMediaVideo.cover*
- *telegram.InputMediaVideo.media*
- *telegram.InputMediaVideo.thumbnail*
- *telegram.InputPaidMedia.media*
- *telegram.InputPaidMediaLivePhoto.media*
- *telegram.InputPaidMediaLivePhoto.photo*
- *telegram.InputPaidMediaPhoto.media*
- *telegram.InputPaidMediaVideo.cover*
- *telegram.InputPaidMediaVideo.media*
- *telegram.InputPaidMediaVideo.thumbnail*
- *telegram.InputProfilePhotoAnimated.animation*
- *telegram.InputProfilePhotoStatic.photo*
- *telegram.InputSticker.sticker*
- *telegram.InputStoryContentPhoto.photo*

- *telegram.InputStoryContentVideo.video*

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*.
- ***read_file_handle*** (`bool`, optional) – If `True` and *obj* is a file handle, the data will be read from the file handle on initialization of this object. If `False`, the file handle will be passed on to the *networking backend* which will have to handle the reading. Defaults to `True`.

  Tip

  If you upload extremely large files, you may want to set this to `False` to avoid reading the complete file into memory. Additionally, this may be supported better by the networking backend (in particular it is handled better by the default *HTTPXRequest*).

  Important

  If you set this to `False`, you have to ensure that the file handle is still open when the request is made. In particular, the following snippet can *not* work as expected.

  ```
  with open('file.txt', 'rb') as file:
      input_file = InputFile(file, read_file_handle=False)

  # here the file handle is already closed and the upload will␣
  ↪fail
  await bot.send_document(chat_id, input_file)
  ```

  Added in version 21.5.

input_file_content

The binary content of the file to send.

Type

`bytes` | `IO`

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.

Changed in version 21.5: Content may now be a file handle.

Return type
tuple[str, bytes | IO, str]

InputMedia

class telegram.**InputMedia**(*media_type*, *media*, *caption=None*, *caption_entities=None*, *parse_mode=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

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

- *telegram.InputMediaAnimation*
- *telegram.InputMediaAudio*
- *telegram.InputMediaDocument*
- *telegram.InputMediaLivePhoto*
- *telegram.InputMediaPhoto*
- *telegram.InputMediaVideo*

Use In

telegram.Bot.edit_message_media()

Changed in version 20.0: Added arguments and attributes *type*, *media*, *caption*, *caption_entities*, *parse_mode*.

See also

Working with Files and Media

Parameters

- ***media_type*** (str) – Type of media that the instance represents.
- ***media*** (str | *InputFile*) – 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. To upload a file, you can either pass a file object (e.g. open("filename", "rb")) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or pathlib.Path object) is supported as well.
- ***caption*** (str, optional) – Caption of the media to be sent, 0-*1024* characters after entities parsing.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of parse_mode.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.
- ***parse_mode*** (str, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

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, 0-*1024* characters after entities parsing.

Type
str

parse_mode

Optional. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*telegram.MessageEntity*]

InputMediaAnimation

class telegram.**InputMediaAnimation**(*media*, *caption=None*, *parse_mode=None*, *width=None*, *height=None*, *duration=None*, *caption_entities=None*, *filename_depr=None*, *has_spoiler=None*, *thumbnail=None*, *show_caption_above_media=None*, ***, *filename=None*, *api_kwargs=None*)

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 animation, unless otherwise specified with the optional arguments.

See also

Working with Files and Media

Use In

- *telegram.Bot.edit_message_media()*
- *telegram.Bot.send_poll()*

Available In

telegram.InputPollOption.media

Changed in version 20.5: Removed the deprecated argument and attribute `thumb`.

Parameters

- ***media*** (`str` | file object | *InputFile* | `bytes` | `pathlib.Path` | *telegram.Animation*) – 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well. Lastly you can pass an existing *telegram.Animation* object to send.

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

- ***filename_depr*** (`str`, optional) – Positional placeholder for keyword only parameter *filename*. For backward compatibility.

  Added in version 22.8.

  Deprecated since version 22.8: This parameter is deprecated, use *filename* instead.

- ***caption*** (`str`, optional) – Caption of the animation to be sent, 0-*1024* characters after entities parsing.

- ***parse_mode*** (`str`, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of `parse_mode`.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list. The input is converted to a tuple.

- ***width*** (`int`, optional) – Animation width.

- ***height*** (int, optional) – Animation height.
- ***duration*** (int | datetime.timedelta, optional) – Animation duration in seconds.

  Changed in version v22.2: datetime.timedelta objects are accepted in addition to plain int values.
- ***has_spoiler*** (bool, optional) – Pass True, if the animation needs to be covered with a spoiler animation.

  Added in version 20.0.
- ***thumbnail*** (file object | bytes | pathlib.Path | str, 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. To upload a file, you can either pass a file object (e.g. open("filename", "rb")) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or pathlib.Path object) is supported as well.

  Added in version 20.2.
- ***show_caption_above_media*** (bool, optional) – Pass True, if the caption must be shown above the message media.

  Added in version 21.3.

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.

Added in version 13.1.

Changed in version 22.8: This parameter is now keyword-only.

type

'animation'.

Type
str

media

Animation to send.

Type
str | *telegram.InputFile*

caption

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

Type
str

parse_mode

Optional. The parse mode to use for text formatting.

Type
str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.

- This attribute is now always a tuple, that may be empty.

Type
tuple[*telegram.MessageEntity*]

width

Optional. Animation width.

Type
int

height

Optional. Animation height.

Type
int

duration

Optional. Animation duration in seconds.

Deprecated since version v22.2: In a future major version this attribute will be of type `datetime.timedelta`. You can opt-in early by setting *PTB_TIMEDELTA=true* or `PTB_TIMEDELTA=1` as an environment variable.

Type
int | datetime.timedelta

has_spoiler

Optional. `True`, if the animation is covered with a spoiler animation.

Added in version 20.0.

Type
bool

thumbnail

Optional. Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file.

Added in version 20.2.

Type
telegram.InputFile

show_caption_above_media

Optional. `True`, if the caption must be shown above the message media.

Added in version 21.3.

Type
bool

InputMediaAudio

class telegram.**InputMediaAudio**(*media*, *caption=None*, *parse_mode=None*, *duration=None*, *performer=None*, *title=None*, *caption_entities=None*, *filename_depr=None*, *thumbnail=None*, ***, *filename=None*, *api_kwargs=None*)

Bases: *telegram.InputMedia*

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

See also

Working with Files and Media

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.

Use In

- *telegram.Bot.edit_message_media()*
- *telegram.Bot.send_media_group()*
- *telegram.Bot.send_poll()*

Changed in version 20.5: Removed the deprecated argument and attribute `thumb`.

Parameters

- ***media*** (`str` | file object | *InputFile* | `bytes` | `pathlib.Path` | *telegram.Audio*) – File to send. Pass a `file_id` as String to send a file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well. Lastly you can pass an existing *telegram.Audio* object to send.

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

- ***filename_depr*** (`str`, optional) – Positional placeholder for keyword only parameter *filename*. For backward compatibility.

  Added in version 22.8.

  Deprecated since version 22.8: This parameter is deprecated, use *filename* instead.

- ***caption*** (`str`, optional) – Caption of the audio to be sent, 0-*1024* characters after entities parsing.

- ***parse_mode*** (`str`, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of `parse_mode`.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list. The input is converted to a tuple.

- ***duration*** (`int` | `datetime.timedelta`, optional) – Duration of the audio in seconds as defined by the sender.

  Changed in version v22.2: `datetime.timedelta` objects are accepted in addition to plain `int` values.

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

- ***title*** (`str`, optional) – Title of the audio as defined by the sender or by audio tags.

- ***thumbnail*** (file object | bytes | pathlib.Path | str, 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well.

  Added in version 20.2.

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.

Added in version 13.1.

Changed in version 22.8: This parameter is now keyword-only.

type

'audio'.

Type
str

media

Audio file to send.

Type
str | *telegram.InputFile*

caption

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

Type
str

parse_mode

Optional. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*telegram.MessageEntity*]

duration

Optional. Duration of the audio in seconds.

Deprecated since version v22.2: In a future major version this attribute will be of type `datetime.timedelta`. You can opt-in early by setting *PTB_TIMEDELTA=true* or `PTB_TIMEDELTA=1` as an environment variable.

Type
int | datetime.timedelta

performer

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

Type
str

title

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

Type
str

thumbnail

Optional. Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file.

Added in version 20.2.

Type
telegram.InputFile

InputMediaDocument

class telegram.**InputMediaDocument**(*media*, *caption=None*, *parse_mode=None*, *disable_content_type_detection=None*, *caption_entities=None*, *filename_depr=None*, *thumbnail=None*, ***, *filename=None*, *api_kwargs=None*)

Bases: *telegram.InputMedia*

Represents a general file to be sent.

See also

Working with Files and Media

Use In

- *telegram.Bot.edit_message_media()*
- *telegram.Bot.send_media_group()*
- *telegram.Bot.send_poll()*

Changed in version 20.5: Removed the deprecated argument and attribute `thumb`.

Parameters

- ***media*** (str | file object | *InputFile* | 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well. Lastly you can pass an existing *telegram.Document* object to send.

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

- ***filename_depr*** (str, optional) – Positional placeholder for keyword only parameter *filename*. For backward compatibility.

Added in version 22.8.

Deprecated since version 22.8: This parameter is deprecated, use *filename* instead.

- ***caption*** (str, optional) – Caption of the document to be sent, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of parse_mode.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.
- ***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.
- ***thumbnail*** (file object | bytes | pathlib.Path | str, 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. To upload a file, you can either pass a file object (e.g. open("filename", "rb")) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or pathlib.Path object) is supported as well.

  Added in version 20.2.

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.

Added in version 13.1.

Changed in version 22.8: This parameter is now keyword-only.

type

'document'.

Type

str

media

File to send.

Type

str | *telegram.InputFile*

caption

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

Type

str

parse_mode

Optional. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type

str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*telegram.MessageEntity*]

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`

thumbnail

Optional. Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file.

Added in version 20.2.

Type
telegram.InputFile

InputMediaLivePhoto

class telegram.**InputMediaLivePhoto**(*media*, *photo*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, *has_spoiler=None*, ***, *api_kwargs=None*)

Bases: *telegram.InputMedia*

Represents a live photo to be sent.

See also

Working with Files and Media

Use In

- *telegram.Bot.edit_message_media()*
- *telegram.Bot.send_media_group()*
- *telegram.Bot.send_poll()*

Available In

telegram.InputPollOption.media

Added in version 22.8.

Parameters

- ***media*** (str | file object | *InputFile* | bytes | pathlib.Path | *Video*) – Video of the live photo to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended). To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well. Sending live photos by a URL is currently unsupported. Lastly you can pass an existing *telegram.Video* object to send.
- ***photo*** (str | file object | *InputFile* | bytes | pathlib.Path | *PhotoSize*) – The static photo to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended). To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well. Sending live photos by a URL is currently unsupported. Lastly you can pass an existing *telegram.PhotoSize* object to send.
- ***caption*** (str, optional) – Caption of the live photo to be sent, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of `parse_mode`.
- ***show_caption_above_media*** (bool, optional) – Pass `True`, if the caption must be shown above the message media.
- ***has_spoiler*** (bool, optional) – Pass `True`, if the video needs to be covered with a spoiler animation.

type

'live_photo'.

Type
str

media

Video of the live photo to send.

Type
str | *telegram.InputFile*

photo

The static photo to send.

Type
str | *telegram.InputFile*

caption

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

Type
str

parse_mode

Optional. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

caption_entities

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

Type
tuple[*telegram.MessageEntity*]

show_caption_above_media

Optional. `True`, if the caption must be shown above the message media.

Type
`bool`

has_spoiler

Optional. `True`, if the video is covered with a spoiler animation.

Type
`bool`

InputMediaLocation

class telegram.**InputMediaLocation**(*latitude*, *longitude*, *horizontal_accuracy=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Represents a location to be sent.

Use In

telegram.Bot.send_poll()

Available In

telegram.InputPollOption.media

Added in version 22.8.

Parameters

- ***latitude*** (`float`) – Latitude of the location.
- ***longitude*** (`float`) – Longitude of the location.
- ***horizontal_accuracy*** (`float`, optional) – The radius of uncertainty for the location, measured in meters; 0-*1500*.

type

'location'.

Type
`str`

latitude

Latitude of the location.

Type
`float`

longitude

Longitude of the location.

Type
float

horizontal_accuracy

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

Type
float

InputMediaPhoto

class telegram.**InputMediaPhoto**(*media*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *filename_depr=None*, *has_spoiler=None*, *show_caption_above_media=None*, *, *filename=None*, *api_kwargs=None*)

Bases: *telegram.InputMedia*

Represents a photo to be sent.

See also

Working with Files and Media

Use In

- *telegram.Bot.edit_message_media()*
- *telegram.Bot.send_media_group()*
- *telegram.Bot.send_poll()*

Available In

telegram.InputPollOption.media

Parameters

- ***media*** (str | file object | *InputFile* | bytes | pathlib.Path | *telegram.PhotoSize*) – 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well. Lastly you can pass an existing *telegram.PhotoSize* object to send.

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

- ***filename_depr*** (str, optional) – Positional placeholder for keyword only parameter *filename*. For backward compatibility.

  Added in version 22.8.

  Deprecated since version 22.8: This parameter is deprecated, use *filename* instead.

- ***caption*** (str, optional) – Caption of the photo to be sent, 0-*1024* characters after entities parsing.

- ***parse_mode*** (str, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of `parse_mode`.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list. The input is converted to a tuple.

- ***has_spoiler*** (`bool`, optional) – Pass `True`, if the photo needs to be covered with a spoiler animation.

  Added in version 20.0.

- ***show_caption_above_media*** (`bool`, optional) – Pass `True`, if the caption must be shown above the message media.

  Added in version 21.3.

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.

Added in version 13.1.

Changed in version 22.8: This parameter is now keyword-only.

type

'photo'.

Type
`str`

media

Photo to send.

Type
`str` | *telegram.InputFile*

caption

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

Type
`str`

parse_mode

Optional. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
`str`

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*telegram.MessageEntity*]

has_spoiler

Optional. `True`, if the photo is covered with a spoiler animation.

Added in version 20.0.

Type
bool

show_caption_above_media

Optional. True, if the caption must be shown above the message media.

Added in version 21.3.

Type
bool

InputMediaSticker

class telegram.**InputMediaSticker**(*media*, *emoji=None*, ***, *filename=None*, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Represents a sticker file to be sent.

See also

Working with Files and Media

Available In

telegram.InputPollOption.media

Added in version 22.8.

Parameters

- ***media*** (str | file object | *InputFile* | bytes | pathlib.Path | *telegram.Sticker*) – 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. To upload a file, you can either pass a file object (e.g. open("filename", "rb")) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or pathlib.Path object) is supported as well.

  Lastly you can pass an existing *telegram.Sticker* object to send.

- ***emoji*** (str, optional) – Emoji associated with the sticker; only for just uploaded stickers.

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

type
'sticker'.

Type
str

media
Sticker file to send.

Type
str | *telegram.InputFile*

emoji
Optional. Emoji associated with the sticker; only for just uploaded stickers.

Type
str

InputMediaVenue

class telegram.**InputMediaVenue**(*latitude*, *longitude*, *title*, *address*, *foursquare_id=None*, *foursquare_type=None*, *google_place_id=None*, *google_place_type=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Represents a venue to be sent.

Use In

telegram.Bot.send_poll()

Available In

telegram.InputPollOption.media

Added in version 22.8.

Parameters

- ***latitude*** (float) – Latitude of the location.
- ***longitude*** (float) – Longitude of the 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, 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)

type
'venue'.

Type
str

latitude
Latitude of the location.

Type
float

longitude
Longitude of the location.

Type
float

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, if known. (For example, "arts_entertainment/default", "arts_entertainment/aquarium" or "food/icecream").

Type
str

google_place_id

Optional. Google Places identifier of the venue.

Type
str

google_place_type

Optional. Google Places type of the venue. (See supported types)

Type
str

InputMediaVideo

class telegram.**InputMediaVideo**(*media*, *caption=None*, *width=None*, *height=None*, *duration=None*, *supports_streaming=None*, *parse_mode=None*, *caption_entities=None*, *filename_depr=None*, *has_spoiler=None*, *thumbnail=None*, *show_caption_above_media=None*, *cover=None*, *start_timestamp=None*, ***, *filename=None*, *api_kwargs=None*)

Bases: *telegram.InputMedia*

Represents a video to be sent.

See also

Working with Files and Media

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.
- ***thumbnail* 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.

Use In

- *telegram.Bot.edit_message_media()*
- *telegram.Bot.send_media_group()*
- *telegram.Bot.send_poll()*

Available In

telegram.InputPollOption.media

Changed in version 20.5: Removed the deprecated argument and attribute `thumb`.

Parameters

- ***media*** (`str` | file object | *InputFile* | `bytes` | `pathlib.Path` | *telegram.Video*) – 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well. Lastly you can pass an existing *telegram.Video* object to send.

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

- ***filename_depr*** (`str`, optional) – Positional placeholder for keyword only parameter *filename*. For backward compatibility.

  Added in version 22.8.

  Deprecated since version 22.8: This parameter is deprecated, use *filename* instead.

- ***caption*** (`str`, optional) – Caption of the video to be sent, 0-*1024* characters after entities parsing.

- ***parse_mode*** (`str`, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of `parse_mode`.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list. The input is converted to a tuple.

- ***width*** (`int`, optional) – Video width.

- ***height*** (`int`, optional) – Video height.

- ***duration*** (`int` | `datetime.timedelta`, optional) – Video duration in seconds.

  Changed in version v22.2: `datetime.timedelta` objects are accepted in addition to plain `int` values.

- ***supports_streaming*** (`bool`, optional) – Pass `True`, if the uploaded video is suitable for streaming.

- ***has_spoiler*** (`bool`, optional) – Pass `True`, if the video needs to be covered with a spoiler animation.

  Added in version 20.0.

- ***thumbnail*** (file object | `bytes` | `pathlib.Path` | `str`, 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well.

Added in version 20.2.

- ***cover*** (file object | `bytes` | `pathlib.Path` | `str`, optional) – Cover for the video in the message. 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well.

  Changed in version 21.11.

- ***start_timestamp*** (`int`, optional) – Start timestamp for the video in the message

  Changed in version 21.11.

- ***show_caption_above_media*** (`bool`, optional) – Pass `True`, if the caption must be shown above the message media.

  Added in version 21.3.

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.

Added in version 13.1.

Changed in version 22.8: This parameter is now keyword-only.

type

'video'.

Type
`str`

media

Video file to send.

Type
`str` | *telegram.InputFile*

caption

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

Type
`str`

parse_mode

Optional. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
`str`

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*telegram.MessageEntity*]

width

Optional. Video width.

Type
int

height

Optional. Video height.

Type
int

duration

Optional. Video duration in seconds.

Deprecated since version v22.2: In a future major version this attribute will be of type datetime.timedelta. You can opt-in early by setting *PTB_TIMEDELTA=true* or PTB_TIMEDELTA=1 as an environment variable.

Type
int | datetime.timedelta

supports_streaming

Optional. True, if the uploaded video is suitable for streaming.

Type
bool

has_spoiler

Optional. True, if the video is covered with a spoiler animation.

Added in version 20.0.

Type
bool

thumbnail

Optional. Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file.

Added in version 20.2.

Type
telegram.InputFile

show_caption_above_media

Optional. True, if the caption must be shown above the message media.

Added in version 21.3.

Type
bool

cover

Optional. Cover for the video in the message. 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well.

Changed in version 21.11.

Type
telegram.InputFile

start_timestamp

Optional. Start timestamp for the video in the message

Changed in version 21.11.

Type
int

InputPaidMedia

class telegram.**InputPaidMedia**(*type*, *media*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Base class for Telegram InputPaidMedia Objects. Currently, it can be one of:

- *telegram.InputPaidMediaPhoto*
- *telegram.InputPaidMediaVideo*
- *telegram.InputPaidMediaLivePhoto*

See also

Working with Files and Media

Use In

telegram.Bot.send_paid_media()

Added in version 21.4.

Parameters

- ***type*** (str) – Type of media that the instance represents.
- ***media*** (str | *InputFile*) – 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well.

type

Type of the input media.

Type
str

media

Media to send.

Type
str | *telegram.InputFile*

LIVE_PHOTO = 'live_photo'

telegram.constants.InputPaidMediaType.LIVE_PHOTO

Added in version 22.8.

PHOTO = 'photo'

telegram.constants.InputPaidMediaType.PHOTO

VIDEO = 'video'

telegram.constants.InputPaidMediaType.VIDEO

InputPaidMediaLivePhoto

class telegram.**InputPaidMediaLivePhoto**(*media*, *photo*, *, *api_kwargs=None*)

Bases: *telegram.InputPaidMedia*

The paid media to send is a live photo.

See also

Working with Files and Media

Use In

telegram.Bot.send_paid_media()

Added in version 22.8.

Parameters

- ***media*** (str | file object | *InputFile* | bytes | pathlib.Path | *Video*) – Video of the live photo to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended). To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or pathlib.Path object) is supported as well. Sending live photos by a URL is currently unsupported. Lastly you can pass an existing *telegram.Video* object to send.
- ***photo*** (str | file object | *InputFile* | bytes | pathlib.Path | *PhotoSize*) – Photo of the live photo to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended). To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or pathlib.Path object) is supported as well. Sending live photos by a URL is currently unsupported. Lastly you can pass an existing *telegram.PhotoSize* object to send.

type

Type of the media, always *'live_photo'*.

Type
str

media

Video of the live photo 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well.

Type
`str` | *telegram.InputFile*

photo

Photo of the live photo 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well.

Type
`str` | *telegram.InputFile*

InputPaidMediaPhoto

class telegram.**InputPaidMediaPhoto**(*media*, ***, *api_kwargs=None*)

Bases: *telegram.InputPaidMedia*

The paid media to send is a photo.

See also

Working with Files and Media

Use In

telegram.Bot.send_paid_media()

Added in version 21.4.

Parameters

media (`str` | file object | *InputFile* | `bytes` | `pathlib.Path` | *telegram.PhotoSize*) – 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well. Lastly you can pass an existing *telegram.PhotoSize* object to send.

type

Type of the media, always *'photo'*.

Type
`str`

media

Photo to send.

Type
`str` | *telegram.InputFile*

InputPaidMediaVideo

class telegram.**InputPaidMediaVideo**(*media*, *thumbnail=None*, *width=None*, *height=None*, *duration=None*, *supports_streaming=None*, *cover=None*, *start_timestamp=None*, *, *api_kwargs=None*)

Bases: *telegram.InputPaidMedia*

The paid media to send is a video.

See also

Working with Files and Media

Use In

telegram.Bot.send_paid_media()

Added in version 21.4.

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.
- *thumbnail* 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 | *InputFile* | bytes | pathlib.Path | *telegram.Video*) – 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. To upload a file, you can either pass a file object (e.g. open("filename", "rb")) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or pathlib.Path object) is supported as well. Lastly you can pass an existing *telegram.Video* object to send.
- ***thumbnail*** (file object | bytes | pathlib.Path | str, 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. To upload a file, you can either pass a file object (e.g. open("filename", "rb")) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or pathlib.Path object) is supported as well.
- ***cover*** (file object | bytes | pathlib.Path | str, optional) – Cover for the video in the message. 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. To upload a file, you can either pass a file object (e.g. open("filename", "rb")) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or pathlib.Path object) is supported as well.

  Changed in version 21.11.
- ***start_timestamp*** (int, optional) – Start timestamp for the video in the message

Changed in version 21.11.

- ***width*** (int, optional) – Video width.
- ***height*** (int, optional) – Video height.
- ***duration*** (int | datetime.timedelta, optional) – Video duration in seconds.

  Changed in version v22.2: datetime.timedelta objects are accepted in addition to plain int values.
- ***supports_streaming*** (bool, optional) – Pass True, if the uploaded video is suitable for streaming.

type

Type of the media, always *'video'*.

Type
str

media

Video to send.

Type
str | *telegram.InputFile*

thumbnail

Optional. Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can't be reused and can be only uploaded as a new file.

Type
telegram.InputFile

cover

Optional. Cover for the video in the message. 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. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or pathlib.Path object) is supported as well.

Changed in version 21.11.

Type
telegram.InputFile

start_timestamp

Optional. Start timestamp for the video in the message

Changed in version 21.11.

Type
int

width

Optional. Video width.

Type
int

height

Optional. Video height.

Type
int

duration

Optional. Video duration in seconds.

Deprecated since version v22.2: In a future major version this attribute will be of type `datetime.timedelta`. You can opt-in early by setting *PTB_TIMEDELTA=true* or `PTB_TIMEDELTA=1` as an environment variable.

Type
`int` | `datetime.timedelta`

supports_streaming

Optional. `True`, if the uploaded video is suitable for streaming.

Type
`bool`

InputPollMedia

Added in version NEXT.VERSION.

`telegram.`**InputPollMedia**

alias of *InputMediaAnimation* | *InputMediaAudio* | *InputMediaDocument* | *InputMediaLivePhoto* | *InputMediaLocation* | *InputMediaPhoto* | *InputMediaVenue* | *InputMediaVideo*

InputProfilePhoto

class `telegram.`**InputProfilePhoto**(*type*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object describes a profile photo to set. Currently, it can be one of

- *InputProfilePhotoStatic*
- *InputProfilePhotoAnimated*

Use In

- *telegram.Bot.set_business_account_profile_photo()*
- *telegram.Bot.set_my_profile_photo()*

Added in version 22.1.

Parameters
type (`str`) – Type of the profile photo.

type

Type of the profile photo.

Type
`str`

ANIMATED = 'animated'

'animated'.

Type
`str`

STATIC = 'static'

'static'.

Type
`str`

InputProfilePhotoAnimated

class telegram.**InputProfilePhotoAnimated**(*animation*, *main_frame_timestamp=None*, ***, *api_kwargs=None*)

Bases: *telegram.InputProfilePhoto*

An animated profile photo in the MPEG4 format.

Use In

- *telegram.Bot.set_business_account_profile_photo()*
- *telegram.Bot.set_my_profile_photo()*

Added in version 22.1.

Parameters

- ***animation*** (file object | *InputFile* | `bytes` | `pathlib.Path`) – The animated profile photo. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well.
- ***main_frame_timestamp*** (`datetime.timedelta` | `int` | `float`, optional) – Timestamp in seconds of the frame that will be used as the static profile photo. Defaults to `0.0`.

type

'animated'.

Type
`str`

animation

The animated profile photo.

Type
telegram.InputFile | `str`

main_frame_timestamp

Optional. Timestamp in seconds of the frame that will be used as the static profile photo. Defaults to `0.0`.

Type
`datetime.timedelta`

InputProfilePhotoStatic

class telegram.**InputProfilePhotoStatic**(*photo*, ***, *api_kwargs=None*)

Bases: *telegram.InputProfilePhoto*

A static profile photo in the .JPG format.

Use In

- *telegram.Bot.set_business_account_profile_photo()*
- *telegram.Bot.set_my_profile_photo()*

Added in version 22.1.

Parameters

photo (file object | *InputFile* | bytes | pathlib.Path) – The static profile photo. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or pathlib.Path object) is supported as well.

type

'static'.

Type

str

photo

The static profile photo.

Type

telegram.InputFile | str

InputPollOption

class telegram.**InputPollOption**(*text*, *text_parse_mode=None*, *text_entities=None*, *media=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

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

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

Use In

telegram.Bot.send_poll()

Added in version 21.2.

Parameters

- ***text*** (str) – Option text, *1-100* characters.
- ***text_parse_mode*** (str, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details. Currently, only custom emoji entities are allowed.
- ***text_entities*** (Sequence[*telegram.MessageEntity*], optional) – Special entities that appear in the option *text*. It can be specified instead of *text_parse_mode*. Currently, only custom emoji entities are allowed. This list is empty if the text does not contain entities.
- ***media*** (*telegram.InputPollOptionMedia*, optional) – Media added to the poll option.

  Added in version 22.8.

text

Option text, *1-100* characters.

Type

str

text_parse_mode

Optional. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details. Currently, only custom emoji entities are allowed.

Type
str

text_entities

Special entities that appear in the option *text*. It can be specified instead of *text_parse_mode*. Currently, only custom emoji entities are allowed. This list is empty if the text does not contain entities.

Type
Sequence[*telegram.MessageEntity*]

media

Optional. Media added to the poll option.

Added in version 22.8.

Type
telegram.InputPollOptionMedia

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*. The *media* field will not be included for deserialization.

Deprecated since version 22.8: This class is input only and will be removed in the next version.

InputPollOptionMedia

Added in version NEXT.VERSION.

telegram.**InputPollOptionMedia**

alias of *InputMediaAnimation* | *InputMediaLivePhoto* | *InputMediaLocation* | *InputMediaPhoto* | *InputMediaSticker* | *InputMediaVenue* | *InputMediaVideo*

InputStoryContent

class telegram.**InputStoryContent**(*type*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object describes the content of a story to post. Currently, it can be one of:

- *telegram.InputStoryContentPhoto*
- *telegram.InputStoryContentVideo*

Use In

- *telegram.Bot.edit_story()*
- *telegram.Bot.post_story()*

Added in version 22.1.

Parameters
type (str) – Type of the content.

type

Type of the content.

Type
str

PHOTO = 'photo'

telegram.constants.InputStoryContentType.PHOTO

VIDEO = 'video'

telegram.constants.InputStoryContentType.VIDEO

InputStoryContentPhoto

class telegram.**InputStoryContentPhoto**(*photo*, *, *api_kwargs=None*)

Bases: *telegram.InputStoryContent*

Describes a photo to post as a story.

Use In

- *telegram.Bot.edit_story()*
- *telegram.Bot.post_story()*

Added in version 22.1.

Parameters

photo (file object | bytes | pathlib.Path | str, optional) – The photo to post as a story. The photo must be of the size *'1080'* x *'1920'* and must not exceed *'10000000'* MB. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or pathlib.Path object) is supported as well..

type

Type of the content, must be *PHOTO*.

Type

str

photo

The photo to post as a story. The photo must be of the size *'1080'* x *'1920'* and must not exceed *'10000000'* MB.

Type

telegram.InputFile

InputStoryContentVideo

class telegram.**InputStoryContentVideo**(*video*, *duration=None*, *cover_frame_timestamp=None*, *is_animation=None*, *, *api_kwargs=None*)

Bases: *telegram.InputStoryContent*

Describes a video to post as a story.

Use In

- *telegram.Bot.edit_story()*
- *telegram.Bot.post_story()*

Added in version 22.1.

Parameters

- ***video*** (file object | bytes | pathlib.Path | str, optional) – The video to post as a story. The video must be of the size *'720'* x *'1080'*, streamable, encoded with H.265 codec, with key frames added each second in the MPEG4 format, and must not exceed *'30000000'*

MB. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or `pathlib.Path` object) is supported as well..

- ***duration*** (`datetime.timedelta` | `int` | `float`, optional) – Precise duration of the video in seconds; 0-*'60'*
- ***cover_frame_timestamp*** (`datetime.timedelta` | `int` | `float`, optional) – Timestamp in seconds of the frame that will be used as the static cover for the story. Defaults to `0.0`.
- ***is_animation*** (`bool`, optional) – Pass `True` if the video has no sound

type

Type of the content, must be *VIDEO*.

Type
`str`

video

The video to post as a story. The video must be of the size *'720'* x *'1080'*, streamable, encoded with `H.265` codec, with key frames added each second in the `MPEG4` format, and must not exceed *'300000000'* MB.

Type
telegram.InputFile

duration

Optional. Precise duration of the video in seconds; 0-*'60'*

Type
`datetime.timedelta`

cover_frame_timestamp

Optional. Timestamp in seconds of the frame that will be used as the static cover for the story. Defaults to `0.0`.

Type
`datetime.timedelta`

is_animation

Optional. Pass `True` if the video has no sound

Type
`bool`

KeyboardButton

class telegram.**KeyboardButton**(*text*, *request_contact=None*, *request_location=None*, *request_poll=None*, *web_app=None*, *request_chat=None*, *request_users=None*, *style=None*, *icon_custom_emoji_id=None*, *request_managed_bot=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents one button of the reply keyboard. At most one of the optional fields other than *text*, *icon_custom_emoji_id*, and *style* must be used to specify the type of the button. For simple text buttons, `str` 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*, *web_app*, *request_users*, *request_chat*, *style* and *icon_custom_emoji_id* 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.
- *request_users* and *request_chat* options will only work in Telegram versions released after 3 February, 2023. Older clients will display unsupported message.
- *style* option will only work in Telegram versions released after February 9, 2026. Older clients will display buttons without styling.

Use In

telegram.Bot.save_prepared_keyboard_button()

Available In

telegram.ReplyKeyboardMarkup.keyboard

Changed in version 21.0: Removed deprecated argument and attribute `request_user`.

Changed in version 20.0: *web_app* is considered as well when comparing objects of this type in terms of equality.

Changed in version 20.5: *request_users* and *request_chat* are considered as well when comparing objects of this type in terms of equality.

Changed in version 22.7: *icon_custom_emoji_id* is considered as well when comparing objects of this type in terms of equality.

Changed in version 22.7: *style* and *icon_custom_emoji_id* are considered as well when comparing objects of this type in terms of equality.

Parameters

- ***text*** (str) – Text of the button. If none of the fields other than *text*, *icon_custom_emoji_id*, and *style* are used, it will be sent 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.

  Added in version 20.0.

- ***request_users*** (*KeyboardButtonRequestUsers*, optional) – If specified, pressing the button will open a list of suitable users. Tapping on any user will send its identifier to the bot in a *telegram.Message.users_shared* service message. Available in private chats only.

  Added in version 20.8.

- ***request_chat*** (*KeyboardButtonRequestChat*, optional) – If specified, pressing the button will open a list of suitable chats. Tapping on a chat will send its identifier to the bot in a *telegram.Message.chat_shared* service message. Available in private chats only.

  Added in version 20.1.

- ***style*** (str, optional) – Style of the button. Must be one of *'primary'* (blue), *'success'* (green), and *'danger'* (red). Color name aliases *'primary'*, *'success'*, and *'danger'* are also available. If omitted, then an app-specific style is used.

  Added in version 22.7.

- ***icon_custom_emoji_id*** (str, optional) – Unique identifier of the custom emoji shown before the text of the button. Can only be used by bots that purchased additional usernames on Fragment or in the messages directly sent by the bot to private, group and supergroup chats if the owner of the bot has a Telegram Premium subscription.

  Added in version 22.7.

- ***request_managed_bot*** (*telegram.KeyboardButtonRequestManagedBot*, optional) – If specified, pressing the button will ask the user to create and share a bot that will be managed by the current bot. Available for bots that enabled management of other bots in the *@BotFather <https://telegram.me/BotFather>* Mini App. Available in private chats only.

  Added in version 22.8.

text

Text of the button. If none of the fields other than *text*, *icon_custom_emoji_id*, and *style* are used, it will be sent as a message when the button is pressed.

Type
str

request_contact

Optional. If True, the user's phone number will be sent as a contact when the button is pressed. Available in private chats only.

Type
bool

request_location

Optional. If True, the user's current location will be sent when the button is pressed. Available in private chats only.

Type
bool

request_poll

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.

Type
KeyboardButtonPollType

web_app

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.

Added in version 20.0.

Type
WebAppInfo

request_users

Optional. If specified, pressing the button will open a list of suitable users. Tapping on any user will send its identifier to the bot in a *telegram.Message.users_shared* service message. Available in private chats only.

Added in version 20.8.

Type
KeyboardButtonRequestUsers

request_chat

Optional. If specified, pressing the button will open a list of suitable chats. Tapping on a chat will send its identifier to the bot in a *telegram.Message.chat_shared* service message. Available in private chats only.

Added in version 20.1.

Type
KeyboardButtonRequestChat

style

Optional. Style of the button. Must be one of *'primary'* (blue), *'success'* (green), and *'danger'* (red). Color name aliases *'primary'*, *'success'*, and *'danger'* are also available. If omitted, then an app-specific style is used.

Added in version 22.7.

Type
str

icon_custom_emoji_id

Optional. Unique identifier of the custom emoji shown before the text of the button. Can only be used by bots that purchased additional usernames on Fragment or in the messages directly sent by the bot to private, group and supergroup chats if the owner of the bot has a Telegram Premium subscription.

Added in version 22.7.

Type
str

request_managed_bot

Optional. If specified, pressing the button will ask the user to create and share a bot that will be managed by the current bot. Available for bots that enabled management of other bots in the *@BotFather <https://telegram.me/BotFather>* Mini App. Available in private chats only.

Added in version 22.8.

Type
telegram.KeyboardButtonRequestManagedBot

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

KeyboardButtonPollType

class telegram.**KeyboardButtonPollType**(*type=None*, ***, *api_kwargs=None*)

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.

Examples

Poll Bot

Available In

telegram.KeyboardButton.request_poll

Parameters

type (str, 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

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

Type

str

KeyboardButtonRequestChat

class telegram.**KeyboardButtonRequestChat**(*request_id*, *chat_is_channel*, *chat_is_forum=None*, *chat_has_username=None*, *chat_is_created=None*, *user_administrator_rights=None*, *bot_administrator_rights=None*, *bot_is_member=None*, *request_title=None*, *request_username=None*, *request_photo=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object defines the criteria used to request a suitable chat. The identifier of the selected user will be shared with the bot when the corresponding button is pressed. More about requesting users ».

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

Available In

telegram.KeyboardButton.request_chat

Added in version 20.1.

Parameters

- ***request_id*** (int) – Signed 32-bit identifier of the request, which will be received back in the *telegram.ChatShared* object. Must be unique within the message.
- ***chat_is_channel*** (bool) – Pass True to request a channel chat, pass False to request a group or a supergroup chat.
- ***chat_is_forum*** (bool, optional) – Pass True to request a forum supergroup, pass False to request a non-forum chat. If not specified, no additional restrictions are applied.

- ***chat_has_username*** (bool, optional) – Pass True to request a supergroup or a channel with a username, pass False to request a chat without a username. If not specified, no additional restrictions are applied.
- ***chat_is_created*** (bool, optional) – Pass True to request a chat owned by the user. Otherwise, no additional restrictions are applied.
- ***user_administrator_rights*** (*ChatAdministratorRights*, optional) – Specifies the required administrator rights of the user in the chat. If not specified, no additional restrictions are applied.
- ***bot_administrator_rights*** (*ChatAdministratorRights*, optional) – Specifies the required administrator rights of the bot in the chat. The rights must be a subset of *user_administrator_rights*. If not specified, no additional restrictions are applied.
- ***bot_is_member*** (bool, optional) – Pass True to request a chat with the bot as a member. Otherwise, no additional restrictions are applied.
- ***request_title*** (bool, optional) – Pass True to request the chat's title.

  Added in version 21.1.
- ***request_username*** (bool, optional) – Pass True to request the chat's username.

  Added in version 21.1.
- ***request_photo*** (bool, optional) – Pass True to request the chat's photo.

  Added in version 21.1.

request_id

Identifier of the request.

Type
int

chat_is_channel

Pass True to request a channel chat, pass False to request a group or a supergroup chat.

Type
bool

chat_is_forum

Optional. Pass True to request a forum supergroup, pass False to request a non-forum chat. If not specified, no additional restrictions are applied.

Type
bool

chat_has_username

Optional. Pass True to request a supergroup or a channel with a username, pass False to request a chat without a username. If not specified, no additional restrictions are applied.

Type
bool

chat_is_created

user. Otherwise, no additional restrictions are applied.

Type
bool

user_administrator_rights

required administrator rights of the user in the chat. If not specified, no additional restrictions are applied.

Type
ChatAdministratorRights

bot_administrator_rights

required administrator rights of the bot in the chat. The rights must be a subset of *user_administrator_rights*. If not specified, no additional restrictions are applied.

Type
ChatAdministratorRights

bot_is_member

as a member. Otherwise, no additional restrictions are applied.

Type
bool

request_title

Optional. Pass `True` to request the chat's title.

Added in version 21.1.

Type
bool

request_username

Optional. Pass `True` to request the chat's username.

Added in version 21.1.

Type
bool

request_photo

Optional. Pass `True` to request the chat's photo.

Added in version 21.1.

Type
bool

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

KeyboardButtonRequestManagedBot

class telegram.**KeyboardButtonRequestManagedBot**(*request_id*, *suggested_name=None*, *suggested_username=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object defines the parameters for the creation of a managed bot. Information about the created bot will be shared with the bot using the update managed_bot and a *telegram.Message* with the field *telegram.Message.managed_bot_created*.

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

Available In

telegram.KeyboardButton.request_managed_bot

Added in version 22.8.

Parameters

- ***request_id*** (int) – Signed 32-bit identifier of the request. Must be unique within the message.

- ***suggested_name*** (str, optional) – Suggested name for the bot.
- ***suggested_username*** (str, optional) – Suggested username for the bot.

request_id

Signed 32-bit identifier of the request. Must be unique within the message.

Type
int

suggested_name

Optional. Suggested name for the bot.

Type
str

suggested_username

Optional. Suggested username for the bot.

Type
str

KeyboardButtonRequestUsers

class telegram.**KeyboardButtonRequestUsers**(*request_id*, *user_is_bot=None*, *user_is_premium=None*, *max_quantity=None*, *request_name=None*, *request_username=None*, *request_photo=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

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

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

Available In

telegram.KeyboardButton.request_users

Added in version 20.8: This class was previously named KeyboardButtonRequestUser.

Parameters

- ***request_id*** (int) – Signed 32-bit identifier of the request, which will be received back in the *telegram.UsersShared* object. Must be unique within the message.
- ***user_is_bot*** (bool, optional) – Pass True to request a bot, pass False to request a regular user. If not specified, no additional restrictions are applied.
- ***user_is_premium*** (bool, optional) – Pass True to request a premium user, pass False to request a non-premium user. If not specified, no additional restrictions are applied.
- ***max_quantity*** (int, optional) – The maximum number of users to be selected; *1* - *10*. Defaults to *1* .

  Added in version 20.8.
- ***request_name*** (bool, optional) – Pass True to request the users' first and last name.

  Added in version 21.1.
- ***request_username*** (bool, optional) – Pass True to request the users' username.

  Added in version 21.1.

- ***request_photo*** (bool, optional) – Pass True to request the users' photo.

  Added in version 21.1.

request_id

Identifier of the request.

Type
int

user_is_bot

Optional. Pass True to request a bot, pass False to request a regular user. If not specified, no additional restrictions are applied.

Type
bool

user_is_premium

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

Type
bool

max_quantity

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

Added in version 20.8.

Type
int

request_name

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

Added in version 21.1.

Type
bool

request_username

Optional. Pass True to request the users' username.

Added in version 21.1.

Type
bool

request_photo

Optional. Pass True to request the users' photo.

Added in version 21.1.

Type
bool

LinkPreviewOptions

class telegram.**LinkPreviewOptions**(*is_disabled=None*, *url=None*, *prefer_small_media=None*, *prefer_large_media=None*, *show_above_text=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes the options used for link preview generation.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *is_disabled*, *url*, *prefer_small_media*, *prefer_large_media*, and *show_above_text* are equal.

Use In

- *telegram.Bot.edit_message_text()*
- *telegram.Bot.send_message()*

Available In

- *telegram.ExternalReplyInfo.link_preview_options*
- *telegram.InputTextMessageContent.link_preview_options*
- *telegram.Message.link_preview_options*
- *telegram.ext.Defaults.link_preview_options*

Added in version 20.8.

Parameters

- ***is_disabled*** (bool, optional) – True, if the link preview is disabled.
- ***url*** (str, optional) – The URL to use for the link preview. If empty, then the first URL found in the message text will be used.
- ***prefer_small_media*** (bool, optional) – True, if the media in the link preview is supposed to be shrunk; ignored if the URL isn't explicitly specified or media size change isn't supported for the preview.
- ***prefer_large_media*** (bool, optional) – True, if the media in the link preview is supposed to be enlarged; ignored if the URL isn't explicitly specified or media size change isn't supported for the preview.
- ***show_above_text*** (bool, optional) – True, if the link preview must be shown above the message text; otherwise, the link preview will be shown below the message text.

is_disabled

Optional. True, if the link preview is disabled.

Type
bool

url

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

Type
str

prefer_small_media

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

Type
bool

prefer_large_media

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

Type
`bool`

show_above_text

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

Type
`bool`

LivePhoto

class telegram.**LivePhoto**(*file_id*, *file_unique_id*, *width*, *height*, *duration*, *photo=None*, *mime_type=None*, *file_size=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a live photo.

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.

Use In

telegram.Bot.send_live_photo()

Available In

- *telegram.ExternalReplyInfo.live_photo*
- *telegram.Message.effective_attachment*
- *telegram.Message.live_photo*
- *telegram.PaidMediaLivePhoto.live_photo*
- *telegram.PollMedia.live_photo*

Added in version 22.8.

Parameters

- ***file_id*** (`str`) – Identifier for the video 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 the sender.
- ***height*** (`int`) – Video height as defined by the sender.
- ***duration*** (`int` | `datetime.timedelta`) – Duration of the video in seconds as defined by the sender.
- ***photo*** (Sequence[*telegram.PhotoSize*], optional) – Available sizes of the corresponding static photo.
- ***mime_type*** (`str`, optional) – MIME type of a file as defined by the sender.

- ***file_size*** (int, optional) – File size in bytes.

file_id

Identifier for the video file which can be used to download or reuse the 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 the sender.

Type
int

height

Video height as defined by the sender.

Type
int

duration

Duration of the video in seconds as defined by the sender.

Type
datetime.timedelta

photo

Optional. Available sizes of the corresponding static photo.

Type
tuple[*telegram.PhotoSize*]

mime_type

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

Type
str

file_size

Optional. File size in bytes.

Type
int

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

Location

class telegram.**Location**(*longitude*, *latitude*, *horizontal_accuracy=None*, *live_period=None*, *heading=None*, *proximity_alert_radius=None*, ***, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.edit_message_live_location()*
- *telegram.Bot.send_location()*

Available In

- *telegram.BusinessLocation.location*
- *telegram.ChatLocation.location*
- *telegram.ChosenInlineResult.location*
- *telegram.ExternalReplyInfo.location*
- *telegram.InlineQuery.location*
- *telegram.Message.effective_attachment*
- *telegram.Message.location*
- *telegram.PollMedia.location*
- *telegram.Venue.location*

Parameters

- ***longitude*** (`float`) – Longitude as defined by the sender.
- ***latitude*** (`float`) – Latitude as defined by the sender.
- ***horizontal_accuracy*** (`float`, optional) – The radius of uncertainty for the location, measured in meters; 0-*1500*.
- ***live_period*** (`int` | `datetime.timedelta`, optional) – Time relative to the message sending date, during which the location can be updated, in seconds. For active live locations only.

  Changed in version v22.2: `datetime.timedelta` objects are accepted in addition to plain `int` values.
- ***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.

longitude

Longitude as defined by the sender.

Type
`float`

latitude

Latitude as defined by the sender.

Type
`float`

horizontal_accuracy

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

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.

Deprecated since version v22.2: In a future major version this attribute will be of type `datetime.timedelta`. You can opt-in early by setting *PTB_TIMEDELTA=true* or `PTB_TIMEDELTA=1` as an environment variable.

Type
`int | datetime.timedelta`

heading

Optional. The direction in which user is moving, in degrees; *1-360*. 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`

HORIZONTAL_ACCURACY = 1500

telegram.constants.LocationLimit.HORIZONTAL_ACCURACY

Added in version 20.0.

MAX_HEADING = 360

telegram.constants.LocationLimit.MAX_HEADING

Added in version 20.0.

MIN_HEADING = 1

telegram.constants.LocationLimit.MIN_HEADING

Added in version 20.0.

LocationAddress

class telegram.**LocationAddress**(*country_code*, *state=None*, *city=None*, *street=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes the physical address of a location.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *country_code*, *state*, *city* and *street* are equal.

Available In

telegram.StoryAreaTypeLocation.address

Added in version 22.1.

Parameters

- ***country_code*** (`str`) – The two-letter `ISO 3166-1 alpha-2` country code of the country where the location is located.
- ***state*** (`str`, optional) – State of the location.
- ***city*** (`str`, optional) – City of the location.

- ***street*** (str, optional) – Street address of the location.

country_code

The two-letter ISO 3166-1 alpha-2 country code of the country where the location is located.

Type
str

state

Optional. State of the location.

Type
str

city

Optional. City of the location.

Type
str

street

Optional. Street address of the location.

Type
str

LoginUrl

class telegram.**LoginUrl**(*url*, *forward_text=None*, *bot_username=None*, *request_write_access=None*, ***, *api_kwargs=None*)

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

Available In

telegram.InlineKeyboardButton.login_url

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 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.

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. 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.

Type
`str`

request_write_access

Optional. Pass `True` to request the permission for your bot to send messages to the user.

Type
`bool`

ManagedBotCreated

class telegram.**ManagedBotCreated**(*bot*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object contains information about the bot that was created to be managed by the current bot.

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

Available In

telegram.Message.managed_bot_created

Added in version 22.8.

Parameters
bot (*telegram.User*) – Information about the bot. The bot's token can be fetched using the method *get_managed_bot_token()*.

bot

Information about the bot. The bot's token can be fetched using the method *get_managed_bot_token()*.

Type
telegram.User

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

ManagedBotUpdated

class telegram.**ManagedBotUpdated**(*user*, *bot*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object contains information about the creation, token update, or owner update of a bot that is managed by the current bot.

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

Available In

telegram.Update.managed_bot

Added in version 22.8.

Parameters

- ***user*** (*telegram.User*) – User that created the bot.
- ***bot*** (*telegram.User*) – Information about the bot. Token of the bot can be fetched using the method *get_managed_bot_token()*.

user

User that created the bot.

Type
telegram.User

bot

Information about the bot. Token of the bot can be fetched using the method *get_managed_bot_token()*.

Type
telegram.User

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

MaybeInaccessibleMessage

class telegram.**MaybeInaccessibleMessage**(*chat*, *message_id*, *date*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Base class for Telegram Message Objects.

Currently, that includes *telegram.Message* and *telegram.InaccessibleMessage*.

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

Available In

- *telegram.CallbackQuery.message*
- *telegram.Message.pinned_message*
- *telegram.PollOptionAdded.poll_message*

- *telegram.PollOptionDeleted.poll_message*

Changed in version 21.0: `__bool__` is no longer overriden and defaults to Pythons standard implementation.

Added in version 20.8.

Parameters

- ***message_id*** (`int`) – Unique message identifier.
- ***date*** (`datetime.datetime`) – Date the message was sent in Unix time or 0 in Unix time. Converted to `datetime.datetime`

  The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***chat*** (*telegram.Chat*) – Conversation the message belongs to.

message_id

Unique message identifier.

Type
`int`

date

Date the message was sent in Unix time or 0 in Unix time. Converted to `datetime.datetime`

The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
`datetime.datetime`

chat

Conversation the message belongs to.

Type
telegram.Chat

property is_accessible

Convenience attribute. `True`, if the date is not 0 in Unix time.

Added in version 20.8.

MenuButton

class telegram.**MenuButton**(*type*, ***, *api_kwargs=None*)

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.

Use In

telegram.Bot.set_chat_menu_button()

Returned In

telegram.Bot.get_chat_menu_button()

Added 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=None*)

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*, optional) – The bot associated with this object. Defaults to None, in which case shortcut methods will not be available.

  Changed in version 21.4: *bot* is now optional and defaults to None

Returns

The Telegram object.

MenuButtonCommands

class telegram.**MenuButtonCommands**(*, *api_kwargs=None*)

Bases: *telegram.MenuButton*

Represents a menu button, which opens the bot's list of commands.

Use In

telegram.Bot.set_chat_menu_button()

Returned In

telegram.Bot.get_chat_menu_button()

Added in version 20.0.

type

'commands'.

Type

str

MenuButtonDefault

class telegram.**MenuButtonDefault**(*, *api_kwargs=None*)

Bases: *telegram.MenuButton*

Describes that no specific value for the menu button was set.

Use In

telegram.Bot.set_chat_menu_button()

Returned In

telegram.Bot.get_chat_menu_button()

Added in version 20.0.

type

'default'.

Type

str

MenuButtonWebApp

class telegram.**MenuButtonWebApp**(*text*, *web_app*, *, *api_kwargs=None*)

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.

Use In

telegram.Bot.set_chat_menu_button()

Returned In

telegram.Bot.get_chat_menu_button()

Added 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()* of *Bot*. Alternatively, a `t.me` link to a Web App of the bot can be specified in the object instead of the Web App's URL, in which case the Web App will be opened as if the user pressed the link.

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()* of *Bot*. Alternatively, a `t.me` link to a Web App of the bot can be specified in the object instead of the Web App's URL, in which case the Web App will be opened as if the user pressed the link.

Type
telegram.WebAppInfo

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

Message

class telegram.**Message**(*message_id*, *date*, *chat*, *from_user=None*, *reply_to_message=None*, *edit_date=None*, *text=None*, *entities=None*, *caption_entities=None*, *audio=None*, *document=None*, *game=None*, *photo=None*, *sticker=None*, *video=None*, *voice=None*, *video_note=None*, *new_chat_members=None*, *caption=None*, *contact=None*, *location=None*, *venue=None*, *left_chat_member=None*, *new_chat_title=None*, *new_chat_photo=None*, *delete_chat_photo=None*, *group_chat_created=None*, *supergroup_chat_created=None*, *channel_chat_created=None*, *migrate_to_chat_id=None*, *migrate_from_chat_id=None*, *pinned_message=None*, *invoice=None*, *successful_payment=None*, *author_signature=None*, *media_group_id=None*, *connected_website=None*, *animation=None*, *passport_data=None*, *poll=None*, *reply_markup=None*, *dice=None*, *via_bot=None*, *proximity_alert_triggered=None*, *sender_chat=None*, *video_chat_started=None*, *video_chat_ended=None*, *video_chat_participants_invited=None*, *message_auto_delete_timer_changed=None*, *video_chat_scheduled=None*, *is_automatic_forward=None*, *has_protected_content=None*, *web_app_data=None*, *is_topic_message=None*, *message_thread_id=None*, *forum_topic_created=None*, *forum_topic_closed=None*, *forum_topic_reopened=None*, *forum_topic_edited=None*, *general_forum_topic_hidden=None*, *general_forum_topic_unhidden=None*, *write_access_allowed=None*, *has_media_spoiler=None*, *chat_shared=None*, *story=None*, *giveaway=None*, *giveaway_completed=None*, *giveaway_created=None*, *giveaway_winners=None*, *users_shared=None*, *link_preview_options=None*, *external_reply=None*, *quote=None*, *forward_origin=None*, *reply_to_story=None*, *boost_added=None*, *sender_boost_count=None*, *business_connection_id=None*, *sender_business_bot=None*, *is_from_offline=None*, *chat_background_set=None*, *effect_id=None*, *show_caption_above_media=None*, *paid_media=None*, *refunded_payment=None*, *gift=None*, *unique_gift=None*, *paid_message_price_changed=None*, *paid_star_count=None*, *direct_message_price_changed=None*, *checklist=None*, *checklist_tasks_done=None*, *checklist_tasks_added=None*, *is_paid_post=None*, *direct_messages_topic=None*, *reply_to_checklist_task_id=None*, *suggested_post_declined=None*, *suggested_post_paid=None*, *suggested_post_refunded=None*, *suggested_post_info=None*, *suggested_post_approved=None*, *suggested_post_approval_failed=None*, *gift_upgrade_sent=None*, *chat_owner_changed=None*, *chat_owner_left=None*, *sender_tag=None*, *poll_option_added=None*, *poll_option_deleted=None*, *reply_to_poll_option_id=None*, *managed_bot_created=None*, *guest_bot_caller_user=None*, *guest_bot_caller_chat=None*, *guest_query_id=None*, *live_photo=None*, ***, *api_kwargs=None*)

Bases: *telegram.MaybeInaccessibleMessage*

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.

Use In

telegram.ext.Application.migrate_chat_data()

Available In

- *telegram.CallbackQuery.message*
- *telegram.ChatFullInfo.pinned_message*
- *telegram.ChecklistTasksAdded.checklist_message*
- *telegram.ChecklistTasksDone.checklist_message*
- *telegram.GiveawayCompleted.giveaway_message*
- *telegram.Message.pinned_message*
- *telegram.Message.reply_to_message*
- *telegram.PollOptionAdded.poll_message*
- *telegram.PollOptionDeleted.poll_message*
- *telegram.SuggestedPostApprovalFailed.suggested_post_message*
- *telegram.SuggestedPostApproved.suggested_post_message*
- *telegram.SuggestedPostDeclined.suggested_post_message*
- *telegram.SuggestedPostPaid.suggested_post_message*
- *telegram.SuggestedPostRefunded.suggested_post_message*
- *telegram.Update.business_message*
- *telegram.Update.channel_post*
- *telegram.Update.edited_business_message*
- *telegram.Update.edited_channel_post*
- *telegram.Update.edited_message*
- *telegram.Update.effective_message*
- *telegram.Update.guest_message*
- *telegram.Update.message*

Returned In

- *telegram.Bot.edit_message_caption()*
- *telegram.Bot.edit_message_checklist()*
- *telegram.Bot.edit_message_live_location()*
- *telegram.Bot.edit_message_media()*
- *telegram.Bot.edit_message_reply_markup()*
- *telegram.Bot.edit_message_text()*
- *telegram.Bot.forward_message()*
- *telegram.Bot.get_user_personal_chat_messages()*
- *telegram.Bot.send_animation()*
- *telegram.Bot.send_audio()*
- *telegram.Bot.send_checklist()*
- *telegram.Bot.send_contact()*

- *telegram.Bot.send_dice()*
- *telegram.Bot.send_document()*
- *telegram.Bot.send_game()*
- *telegram.Bot.send_invoice()*
- *telegram.Bot.send_live_photo()*
- *telegram.Bot.send_location()*
- *telegram.Bot.send_media_group()*
- *telegram.Bot.send_message()*
- *telegram.Bot.send_paid_media()*
- *telegram.Bot.send_photo()*
- *telegram.Bot.send_poll()*
- *telegram.Bot.send_sticker()*
- *telegram.Bot.send_venue()*
- *telegram.Bot.send_video_note()*
- *telegram.Bot.send_video()*
- *telegram.Bot.send_voice()*
- *telegram.Bot.set_game_score()*
- *telegram.Bot.stop_message_live_location()*

Changed in version 21.0: Removed deprecated arguments and attributes `user_shared`, `forward_from`, `forward_from_chat`, `forward_from_message_id`, `forward_signature`, `forward_sender_name` and `forward_date`.

Changed in version 20.8: * This class is now a subclass of *telegram.MaybeInaccessibleMessage*. * The *pinned_message* now can be either *telegram.Message* or *telegram.InaccessibleMessage*.

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. In specific instances (e.g., message containing a video sent to a big chat), the server might automatically schedule a message instead of sending it immediately. In such cases, this field will be `0` and the relevant message will be unusable until it is actually sent.
- ***from_user*** (*telegram.User*, optional) – Sender of the message; may be empty for messages sent to channels. For backward compatibility, if the message was sent on behalf of a chat, the field contains a fake sender user in non-channel chats.
- ***sender_chat*** (*telegram.Chat*, optional) – Sender of the message when sent on behalf of a chat. For example, the supergroup itself for messages sent by its anonymous

administrators or a linked channel for messages automatically forwarded to the channel's discussion group. For backward compatibility, if the message was sent on behalf of a chat, the field from contains a fake sender user in non-channel chats.

- ***date*** (datetime.datetime) – Date the message was sent in Unix time. Converted to datetime.datetime.

  Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

- ***chat*** (*telegram.Chat*) – Conversation the message belongs to.

- ***is_automatic_forward*** (bool, optional) – True, if the message is a channel post that was automatically forwarded to the connected discussion group.

  Added in version 13.9.

- ***reply_to_message*** (*telegram.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.

- ***edit_date*** (datetime.datetime, optional) – Date the message was last edited in Unix time. Converted to datetime.datetime.

  Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

- ***has_protected_content*** (bool, optional) – True, if the message can't be forwarded.

  Added in version 13.9.

- ***is_from_offline*** (bool, optional) – True, if the message was sent by an implicit action, for example, as an away or a greeting business message, or as a scheduled message.

  Added in version 21.1.

- ***media_group_id*** (str, optional) – The unique identifier inside this chat 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*** (Sequence[*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. This list is empty if the message does not contain entities.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.

- ***link_preview_options*** (*telegram.LinkPreviewOptions*, optional) – Options used for link preview generation for the message, if it is a text message and link preview options were changed.

  Added in version 20.8.

- ***suggested_post_info*** (*telegram.SuggestedPostInfo*, optional) – Information about suggested post parameters if the message is a suggested post in a channel direct messages chat. If the message is an approved or declined suggested post, then it can't be edited.

  Added in version 22.4.

- ***effect_id*** (str, optional) – Unique identifier of the message effect added to the message.

  Added in version 21.3.

- ***caption_entities*** (Sequence[*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. This list is empty if the message does not contain caption entities.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list. The input is converted to a tuple.

- ***show_caption_above_media*** (`bool`, optional) – `True`, if the caption must be shown above the message media.

  Added in version 21.3.

- ***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. *More about games >>*.

- ***photo*** (Sequence[*telegram.PhotoSize*], optional) – Message is a photo, available sizes of the photo. This list is empty if the message does not contain a photo.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list. The input is converted to a tuple.

- ***sticker*** (*telegram.Sticker*, optional) – Message is a sticker, information about the sticker.

- ***story*** (*telegram.Story*, optional) – Message is a forwarded story.

  Added in version 20.5.

- ***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*** (Sequence[*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). This list is empty if the message does not contain new chat members.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list. The input is converted to a tuple.

- ***caption*** (`str`, optional) – Caption for the animation, audio, document, paid media, 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*** (Sequence[*telegram.PhotoSize*], optional) – A chat photo was changed to this value. This list is empty if the message does not contain a new chat photo.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.
- ***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.

  Added in version 13.4.
- ***migrate_to_chat_id*** (int, optional) – The group has been migrated to a supergroup with the specified identifier.
- ***migrate_from_chat_id*** (int, optional) – The supergroup has been migrated from a group with the specified identifier.
- ***pinned_message*** (*telegram.MaybeInaccessibleMessage*, 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.

  Changed in version 20.8: This attribute now is either *telegram.Message* or *telegram.InaccessibleMessage*.
- ***invoice*** (*telegram.Invoice*, optional) – Message is an invoice for a payment, information about the invoice. *More about payments* >>.
- ***successful_payment*** (*telegram.SuccessfulPayment*, optional) – Message is a service message about a successful payment, information about the payment. *More about payments* >>.
- ***connected_website*** (str, optional) – The domain name of the website on which the user has logged in. More about Telegram Login >>.
- ***author_signature*** (str, optional) – Signature of the post author for messages in channels, or the custom title of an anonymous group administrator.
- ***paid_star_count*** (int, optional) – The number of Telegram Stars that were paid by the sender of the message to send it

  Added in version 22.1.
- ***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.
- ***via_bot*** (*telegram.User*, optional) – Bot through which message was sent.

- ***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.

  Added in version 20.0.
- ***video_chat_started*** (*telegram.VideoChatStarted*, optional) – Service message: video chat started.

  Added in version 20.0.
- ***video_chat_ended*** (*telegram.VideoChatEnded*, optional) – Service message: video chat ended.

  Added in version 20.0.
- ***video_chat_participants_invited*** (*telegram.VideoChatParticipantsInvited* optional) – Service message: new participants invited to a video chat.

  Added in version 20.0.
- ***web_app_data*** (*telegram.WebAppData*, optional) – Service message: data sent by a Web App.

  Added in version 20.0.
- ***reply_markup*** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message. *login_url* buttons are represented as ordinary url buttons.
- ***is_topic_message*** (`bool`, optional) – `True`, if the message is sent to a topic in a forum supergroup or a private chat with the bot.

  Added in version 20.0.
- ***message_thread_id*** (`int`, optional) – Unique identifier of a message thread or forum topic to which the message belongs; for supergroups and private chats only.

  Added in version 20.0.
- ***forum_topic_created*** (*telegram.ForumTopicCreated*, optional) – Service message: forum topic created.

  Added in version 20.0.
- ***forum_topic_closed*** (*telegram.ForumTopicClosed*, optional) – Service message: forum topic closed.

  Added in version 20.0.
- ***forum_topic_reopened*** (*telegram.ForumTopicReopened*, optional) – Service message: forum topic reopened.

  Added in version 20.0.
- ***forum_topic_edited*** (*telegram.ForumTopicEdited*, optional) – Service message: forum topic edited.

  Added in version 20.0.
- ***general_forum_topic_hidden*** (*telegram.GeneralForumTopicHidden*, optional) – Service message: General forum topic hidden.

  Added in version 20.0.
- ***general_forum_topic_unhidden*** (*telegram.GeneralForumTopicUnhidden*, optional) – Service message: General forum topic unhidden.

  Added in version 20.0.

- ***write_access_allowed*** (*telegram.WriteAccessAllowed*, optional) – Service message: the user allowed the bot to write messages after adding it to the attachment or side menu, launching a Web App from a link, or accepting an explicit request from a Web App sent by the method requestWriteAccess.

  Added in version 20.0.

- ***has_media_spoiler*** (`bool`, optional) – `True`, if the message media is covered by a spoiler animation.

  Added in version 20.0.

- ***checklist*** (*telegram.Checklist*, optional) – Message is a checklist

  Added in version 22.3.

- ***users_shared*** (*telegram.UsersShared*, optional) – Service message: users were shared with the bot

  Added in version 20.8.

- ***chat_shared*** (*telegram.ChatShared*, optional) – Service message: a chat was shared with the bot.

  Added in version 20.1.

- ***gift*** (*telegram.GiftInfo*, optional) – Service message: a regular gift was sent or received.

  Added in version 22.1.

- ***unique_gift*** (*telegram.UniqueGiftInfo*, optional) – Service message: a unique gift was sent or received

  Added in version 22.1.

- ***gift_upgrade_sent*** (*telegram.GiftInfo*, optional) – Service message: upgrade of a gift was purchased after the gift was sent

  Added in version 22.6.

- ***giveaway_created*** (*telegram.GiveawayCreated*, optional) – Service message: a scheduled giveaway was created

  Added in version 20.8.

- ***giveaway*** (*telegram.Giveaway*, optional) – The message is a scheduled giveaway message

  Added in version 20.8.

- ***giveaway_winners*** (*telegram.GiveawayWinners*, optional) – A giveaway with public winners was completed

  Added in version 20.8.

- ***giveaway_completed*** (*telegram.GiveawayCompleted*, optional) – Service message: a giveaway without public winners was completed

  Added in version 20.8.

- ***paid_message_price_changed*** (*telegram.PaidMessagePriceChanged*, optional) – Service message: the price for paid messages has changed in the chat

  Added in version 22.1.

- ***suggested_post_approved*** (*telegram.SuggestedPostApproved*, optional) – Service message: a suggested post was approved.

  Added in version 22.4.

- ***suggested_post_approval_failed*** (*telegram.SuggestedPostApproved*, optional) – Service message: approval of a suggested post has failed.

  Added in version 22.4.

- ***suggested_post_declined*** (*telegram.SuggestedPostDeclined*, optional) – Service message: a suggested post was declined.

  Added in version 22.4.

- ***suggested_post_paid*** (*telegram.SuggestedPostPaid*, optional) – Service message: payment for a suggested post was received.

  Added in version 22.4.

- ***suggested_post_refunded*** (*telegram.SuggestedPostRefunded*, optional) – Service message: payment for a suggested post was refunded.

  Added in version 22.4.

- ***external_reply*** (*telegram.ExternalReplyInfo*, optional) – Information about the message that is being replied to, which may come from another chat or forum topic.

  Added in version 20.8.

- ***quote*** (*telegram.TextQuote*, optional) – For replies that quote part of the original message, the quoted part of the message.

  Added in version 20.8.

- ***forward_origin*** (*telegram.MessageOrigin*, optional) – Information about the original message for forwarded messages

  Added in version 20.8.

- ***reply_to_story*** (*telegram.Story*, optional) – For replies to a story, the original story.

  Added in version 21.0.

- ***boost_added*** (*telegram.ChatBoostAdded*, optional) – Service message: user boosted the chat.

  Added in version 21.0.

- ***sender_boost_count*** (`int`, optional) – If the sender of the message boosted the chat, the number of boosts added by the user.

  Added in version 21.0.

- ***business_connection_id*** (`str`, optional) – Unique identifier of the business connection from which the message was received. If non-empty, the message belongs to a chat of the corresponding business account that is independent from any potential bot chat which might share the same identifier.

  Added in version 21.1.

- ***sender_business_bot*** (*telegram.User*, optional) – The bot that actually sent the message on behalf of the business account. Available only for outgoing messages sent on behalf of the connected business account.

  Added in version 21.1.

- ***chat_background_set*** (*telegram.ChatBackground*, optional) – Service message: chat background set.

  Added in version 21.2.

- ***checklist_tasks_done*** (*telegram.ChecklistTasksDone*, optional) – Service message: some tasks in a checklist were marked as done or not done

  Added in version 22.3.

- ***checklist_tasks_added*** (*telegram.ChecklistTasksAdded*, optional) – Service message: tasks were added to a checklist

  Added in version 22.3.

- ***paid_media*** (*telegram.PaidMediaInfo*, optional) – Message contains paid media; information about the paid media.

  Added in version 21.4.

- ***refunded_payment*** (*telegram.RefundedPayment*, optional) – Message is a service message about a refunded payment, information about the payment.

  Added in version 21.4.

- ***direct_message_price_changed*** (*telegram.DirectMessagePriceChanged*, optional) – Service message: the price for paid messages in the corresponding direct messages chat of a channel has changed.

  Added in version 22.3.

- ***is_paid_post*** (`bool`, optional) – `True`, if the message is a paid post. Note that such posts must not be deleted for 24 hours to receive the payment and can't be edited.

  Added in version 22.4.

- ***direct_messages_topic*** (*telegram.DirectMessagesTopic*, optional) – Information about the direct messages chat topic that contains the message.

  Added in version 22.4.

- ***reply_to_checklist_task_id*** (`int`, optional) – Identifier of the specific checklist task that is being replied to.

  Added in version 22.4.

- ***chat_owner_left*** (*telegram.ChatOwnerLeft*, optional) – Service message: chat owner has left.

  Added in version 22.7.

- ***chat_owner_changed*** (*telegram.ChatOwnerChanged*, optional) – Service message: chat owner has changed.

  Added in version 22.7.

- ***sender_tag*** (`str`, optional) – Tag or custom title of the sender of the message; for supergroups only.

  Added in version 22.7.

- ***poll_option_added*** (*telegram.PollOptionAdded*, optional) – Service message: answer option was added to a poll.

  Added in version 22.8.

- ***poll_option_deleted*** (*telegram.PollOptionDeleted*, optional) – Service message: answer option was deleted from a poll.

  Added in version 22.8.

- ***reply_to_poll_option_id*** (`str`, optional) – Persistent identifier of the specific poll option that is being replied to.

  Added in version 22.8.

- ***managed_bot_created*** (*telegram.ManagedBotCreated*, optional) – Service message: user created a bot that will be managed by the current bot.

  Added in version 22.8.

- ***guest_bot_caller_user*** (*telegram.User*, optional) – For a message sent by a guest bot, this is the user whose original message triggered the bot's response.

  Added in version 22.8.

- ***guest_bot_caller_chat*** (*telegram.Chat*, optional) – For a message sent by a guest bot, this is the chat whose original message triggered the bot's response.

  Added in version 22.8.

- ***guest_query_id*** (str, optional) – The unique identifier for the guest query. Use this identifier with the method *telegram.Bot.answer_guest_query()* to send a response message. If non-empty, the message belongs to the chat where the guest bot was summoned, which may not coincide with other existing bot chats sharing the same identifier.

  Added in version 22.8.

- ***live_photo*** (*telegram.LivePhoto*, optional) – Message is a live photo, information about the live photo. For backward compatibility, when this field is set, the photo field will also be set.

  Added in version 22.8.

message_id

Unique message identifier inside this chat. In specific instances (e.g., message containing a video sent to a big chat), the server might automatically schedule a message instead of sending it immediately. In such cases, this field will be `0` and the relevant message will be unusable until it is actually sent.

Type
int

from_user

Optional. Sender of the message; may be empty for messages sent to channels. For backward compatibility, if the message was sent on behalf of a chat, the field contains a fake sender user in non-channel chats.

Type
telegram.User

sender_chat

Optional. Sender of the message when sent on behalf of a chat. For example, the supergroup itself for messages sent by its anonymous administrators or a linked channel for messages automatically forwarded to the channel's discussion group. For backward compatibility, if the message was sent on behalf of a chat, the field from contains a fake sender user in non-channel chats.

Type
telegram.Chat

date

Date the message was sent in Unix time. Converted to datetime.datetime.

Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

chat

Conversation the message belongs to.

Type
telegram.Chat

is_automatic_forward

Optional. True, if the message is a channel post that was automatically forwarded to the connected discussion group.

Added 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 in Unix time. Converted to datetime.datetime.

Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

has_protected_content

Optional. True, if the message can't be forwarded.

Added in version 13.9.

Type
bool

is_from_offline

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

Added in version 21.1.

Type
bool

media_group_id

Optional. The unique identifier inside this chat of a media message group this message belongs to.

Type
str

text

Optional. For text messages, the actual UTF-8 text of the message, 0-*4096* characters.

Type
str

entities

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. This list is empty if the message does not contain entities.

Changed in version 20.0: This attribute is now an immutable tuple.

Type
tuple[*telegram.MessageEntity*]

link_preview_options

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

Added in version 20.8.

Type
telegram.LinkPreviewOptions

suggested_post_info

Optional. Information about suggested post parameters if the message is a suggested post in a channel direct messages chat. If the message is an approved or declined suggested post, then it can't be edited.

Added in version 22.4.

Type
telegram.SuggestedPostInfo

effect_id

Optional. Unique identifier of the message effect added to the message.

Added in version 21.3.

Type
str

caption_entities

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. This list is empty if the message does not contain caption entities.

Changed in version 20.0: This attribute is now an immutable tuple.

Type
tuple[*telegram.MessageEntity*]

show_caption_above_media

Optional. True, if the caption must be shown above the message media.

Added in version 21.3.

Type
bool

audio

Optional. Message is an audio file, information about the file.

See also

Working with Files and Media

Type
telegram.Audio

document

Optional. Message is a general file, information about the file.

> **See also**
>
> Working with Files and Media

Type
telegram.Document

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.

> **See also**
>
> Working with Files and Media

Type
telegram.Animation

game

Optional. Message is a game, information about the game. *More about games >>*.

Type
telegram.Game

photo

Optional. Message is a photo, available sizes of the photo. This list is empty if the message does not contain a photo.

> **See also**
>
> Working with Files and Media

Changed in version 20.0: This attribute is now an immutable tuple.

Type
tuple[*telegram.PhotoSize*]

sticker

Optional. Message is a sticker, information about the sticker.

> **See also**
>
> Working with Files and Media

Type
telegram.Sticker

story

Optional. Message is a forwarded story.

Added in version 20.5.

Type
telegram.Story

video

Optional. Message is a video, information about the video.

> **See also**
>
> Working with Files and Media

Type
telegram.Video

voice

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

> **See also**
>
> Working with Files and Media

Type
telegram.Voice

video_note

Optional. Message is a video note, information about the video message.

> **See also**
>
> Working with Files and Media

Type
telegram.VideoNote

new_chat_members

Optional. New members that were added to the group or supergroup and information about them (the bot itself may be one of these members). This list is empty if the message does not contain new chat members.

Changed in version 20.0: This attribute is now an immutable tuple.

Type
tuple[*telegram.User*]

caption

Optional. Caption for the animation, audio, document, paid media, photo, video or voice, 0-*1024* characters.

Type
str

contact

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

Type
telegram.Contact

location

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

Type
telegram.Location

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.

Type
telegram.Venue

left_chat_member

Optional. A member was removed from the group, information about them (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.

Changed in version 20.0: This attribute is now an immutable tuple.

Type
tuple[*telegram.PhotoSize*]

delete_chat_photo

Optional. Service message: The chat photo was deleted.

Type
bool

group_chat_created

Optional. Service message: The group has been created.

Type
bool

supergroup_chat_created

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.

Type
bool

channel_chat_created

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.

Type
bool

message_auto_delete_timer_changed

Optional. Service message: auto-delete timer settings changed in the chat.

Added 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. Note that the Message object in this field will not contain further *reply_to_message* fields even if it is itself a reply.

Changed in version 20.8: This attribute now is either *telegram.Message* or *telegram.InaccessibleMessage*.

Type
telegram.MaybeInaccessibleMessage

invoice

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

Type
telegram.Invoice

successful_payment

Optional. Message is a service message about a successful payment, information about the payment. *More about payments >>*.

Type
telegram.SuccessfulPayment

connected_website

Optional. The domain name of the website on which the user has logged in. More about Telegram Login >>.

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

paid_star_count

Optional. The number of Telegram Stars that were paid by the sender of the message to send it

Added in version 22.1.

Type
int

passport_data

Optional. Telegram Passport data.

Examples

Passport Bot

Type
telegram.PassportData

poll

Optional. Message is a native poll, information about the poll.

Type
telegram.Poll

dice

Optional. Message is a dice with random value.

Type
telegram.Dice

via_bot

Optional. Bot through which 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.

Added in version 20.0.

Type
telegram.VideoChatScheduled

video_chat_started

Optional. Service message: video chat started.

Added in version 20.0.

Type
telegram.VideoChatStarted

video_chat_ended

Optional. Service message: video chat ended.

Added in version 20.0.

Type
telegram.VideoChatEnded

video_chat_participants_invited

Optional. Service message: new participants invited to a video chat.

Added in version 20.0.

Type
telegram.VideoChatParticipantsInvited

web_app_data

Optional. Service message: data sent by a Web App.

Added in version 20.0.

Type
telegram.WebAppData

reply_markup

Optional. Inline keyboard attached to the message. *login_url* buttons are represented as ordinary url buttons.

Type
telegram.InlineKeyboardMarkup

is_topic_message

Optional. True, if the message is sent to a topic in a forum supergroup or a private chat with the bot.

Added in version 20.0.

Type
bool

message_thread_id

Optional. Unique identifier of a message thread or forum topic to which the message belongs; for supergroups and private chats only.

Added in version 20.0.

Type
int

forum_topic_created

Optional. Service message: forum topic created.

Added in version 20.0.

Type
telegram.ForumTopicCreated

forum_topic_closed

Optional. Service message: forum topic closed.

Added in version 20.0.

Type
telegram.ForumTopicClosed

forum_topic_reopened

Optional. Service message: forum topic reopened.

Added in version 20.0.

Type
telegram.ForumTopicReopened

forum_topic_edited

Optional. Service message: forum topic edited.

Added in version 20.0.

Type
telegram.ForumTopicEdited

general_forum_topic_hidden

Optional. Service message: General forum topic hidden.

Added in version 20.0.

Type
telegram.GeneralForumTopicHidden

general_forum_topic_unhidden

Optional. Service message: General forum topic unhidden.

Added in version 20.0.

Type
telegram.GeneralForumTopicUnhidden

write_access_allowed

Optional. Service message: the user allowed the bot added to the attachment menu to write messages.

Added in version 20.0.

Type
telegram.WriteAccessAllowed

has_media_spoiler

Optional. `True`, if the message media is covered by a spoiler animation.

Added in version 20.0.

Type
`bool`

checklist

Optional. Message is a checklist

Added in version 22.3.

Type
telegram.Checklist

users_shared

Optional. Service message: users were shared with the bot

Added in version 20.8.

Type
telegram.UsersShared

chat_shared

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

Added in version 20.1.

Type
telegram.ChatShared

gift

Optional. Service message: a regular gift was sent or received.

Added in version 22.1.

Type
telegram.GiftInfo

unique_gift

Optional. Service message: a unique gift was sent or received

Added in version 22.1.

Type
telegram.UniqueGiftInfo

gift_upgrade_sent

Optional. Service message: upgrade of a gift was purchased after the gift was sent

Added in version 22.6.

Type
telegram.GiftInfo

giveaway_created

Optional. Service message: a scheduled giveaway was created

Added in version 20.8.

Type
telegram.GiveawayCreated

giveaway

Optional. The message is a scheduled giveaway message

Added in version 20.8.

Type
telegram.Giveaway

giveaway_winners

Optional. A giveaway with public winners was completed

Added in version 20.8.

Type
telegram.GiveawayWinners

giveaway_completed

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

Added in version 20.8.

Type
telegram.GiveawayCompleted

paid_message_price_changed

Optional. Service message: the price for paid messages has changed in the chat

Added in version 22.1.

Type
telegram.PaidMessagePriceChanged

suggested_post_approved

Optional. Service message: a suggested post was approved.

Added in version 22.4.

Type
telegram.SuggestedPostApproved

suggested_post_approval_failed

Optional. Service message: approval of a suggested post has failed.

Added in version 22.4.

Type
telegram.SuggestedPostApproved

suggested_post_declined

Optional. Service message: a suggested post was declined.

Added in version 22.4.

Type
telegram.SuggestedPostDeclined

suggested_post_paid

Optional. Service message: payment for a suggested post was received.

Added in version 22.4.

Type
telegram.SuggestedPostPaid

suggested_post_refunded

Optional. Service message: payment for a suggested post was refunded.

Added in version 22.4.

Type
telegram.SuggestedPostRefunded

external_reply

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

Added in version 20.8.

Type
telegram.ExternalReplyInfo

quote

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

Added in version 20.8.

Type
telegram.TextQuote

forward_origin

Optional. Information about the original message for forwarded messages

Added in version 20.8.

Type
telegram.MessageOrigin

reply_to_story

Optional. For replies to a story, the original story.

Added in version 21.0.

Type
telegram.Story

boost_added

Optional. Service message: user boosted the chat.

Added in version 21.0.

Type
telegram.ChatBoostAdded

sender_boost_count

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

Added in version 21.0.

Type
int

business_connection_id

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

Added in version 21.1.

Type
str

sender_business_bot

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

Added in version 21.1.

Type
telegram.User

chat_background_set

Optional. Service message: chat background set

Added in version 21.2.

Type
telegram.ChatBackground

checklist_tasks_done

Optional. Service message: some tasks in a checklist were marked as done or not done

Added in version 22.3.

Type
telegram.ChecklistTasksDone

checklist_tasks_added

Optional. Service message: tasks were added to a checklist

Added in version 22.3.

Type
telegram.ChecklistTasksAdded

paid_media

Optional. Message contains paid media; information about the paid media.

Added in version 21.4.

Type
telegram.PaidMediaInfo

refunded_payment

Optional. Message is a service message about a refunded payment, information about the payment.

Added in version 21.4.

Type
telegram.RefundedPayment

direct_message_price_changed

Optional. Service message: the price for paid messages in the corresponding direct messages chat of a channel has changed.

Added in version 22.3.

Type
telegram.DirectMessagePriceChanged

is_paid_post

Optional. True, if the message is a paid post. Note that such posts must not be deleted for 24 hours to receive the payment and can't be edited.

Added in version 22.4.

Type
bool

direct_messages_topic

Optional. Information about the direct messages chat topic that contains the message.

Added in version 22.4.

Type
telegram.DirectMessagesTopic

reply_to_checklist_task_id

Optional. Identifier of the specific checklist task that is being replied to.

Added in version 22.4.

Type
int

chat_owner_left

Optional. Service message: chat owner has left.

Added in version 22.7.

Type
telegram.ChatOwnerLeft

chat_owner_changed

Optional. Service message: chat owner has changed.

Added in version 22.7.

Type
telegram.ChatOwnerChanged

sender_tag

Optional. Tag or custom title of the sender of the message; for supergroups only.

Added in version 22.7.

Type
str

poll_option_added

Optional. Service message: answer option was added to a poll.

Added in version 22.8.

Type
telegram.PollOptionAdded

poll_option_deleted

Optional. Service message: answer option was deleted from a poll.

Added in version 22.8.

Type
telegram.PollOptionDeleted

reply_to_poll_option_id

Optional. Persistent identifier of the specific poll option that is being replied to.

Added in version 22.8.

Type
str

managed_bot_created

Optional. Service message: user created a bot that will be managed by the current bot.

Added in version 22.8.

Type
telegram.ManagedBotCreated

guest_bot_caller_user

Optional. For a message sent by a guest bot, this is the user whose original message triggered the bot's response.

Added in version 22.8.

Type
telegram.User

guest_bot_caller_chat

Optional. For a message sent by a guest bot, this is the chat whose original message triggered the bot's response.

Added in version 22.8.

Type
telegram.Chat

guest_query_id

Optional. The unique identifier for the guest query. Use this identifier with the method *telegram.Bot.answer_guest_query()* to send a response message. If non-empty, the message belongs to the chat where the guest bot was summoned, which may not coincide with other existing bot chats sharing the same identifier.

Added in version 22.8.

Type
str

live_photo

Optional. Message is a live photo, information about the live photo. For backward compatibility, when this field is set, the photo field will also be set.

Added in version 22.8.

Type
telegram.LivePhoto

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

Shortcut for:

```
await bot.answer_guest_query(
    self.guest_query_id,
    *args, **kwargs,
)
```

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

Added in version 22.8.

Returns
On success, a *telegram.SentGuestMessage* is returned.

Return type
telegram.SentGuestMessage

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

Shortcut for:

```
await bot.approve_suggested_post(
    chat_id=message.chat_id,
    message_id=message.message_id,
    *args, **kwargs
)
```

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

Added in version 22.4.

Returns
`bool` On success, `True` is returned.

build_reply_arguments(*quote=None*, *quote_index=None*, *target_chat_id=None*, *allow_sending_without_reply=None*, *message_thread_id=None*)

Builds a dictionary with the keys `chat_id` and `reply_parameters`. This dictionary can be used to reply to a message with the given quote and target chat.

Examples

Usage with *telegram.Bot.send_message()*:

```
await bot.send_message(
    text="This is a reply",
    **message.build_reply_arguments(quote="Quoted Text")
)
```

Usage with *reply_text()*, replying in the same chat:

```
await message.reply_text(
    "This is a reply",
    do_quote=message.build_reply_arguments(quote="Quoted Text")
)
```

Usage with *reply_text()*, replying in a different chat:

```
await message.reply_text(
    "This is a reply",
    do_quote=message.build_reply_arguments(
        quote="Quoted Text",
        target_chat_id=-100123456789
    )
)
```

Added in version 20.8.

Parameters

- ***quote*** (`str`, optional) – Passed in *compute_quote_position_and_entities()* as parameter *quote* to compute quote entities. Defaults to `None`.

- ***quote_index*** (int, optional) – Passed in *compute_quote_position_and_entities()* as parameter *quote_index* to compute quote position. Defaults to None.
- ***target_chat_id*** (int | str, optional) – Unique identifier for the target chat or username of the target channel (in the format @channelusername). Defaults to *chat_id*.
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Will be applied only if the reply happens in the same chat and forum topic.
- ***message_thread_id*** (int, optional) – Unique identifier for the target message thread of the forum topic.

Return type
dict

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.

> **Warning**
>
> The return value of this property is a best-effort approach. Unfortunately, it can not be guaranteed that sending a message with the returned string will render in the same way as the original message produces the same *entities*/*caption_entities* as the original message. For example, Telegram recommends that entities of type *BLOCKQUOTE* and *PRE* *should* start and end on a new line, but does not enforce this and leaves rendering decisions up to the clients.

Changed in version 13.10: Spoiler entities are now formatted as HTML.

Changed in version 20.3: Custom emoji entities are now supported.

Changed in version 20.8: Blockquote entities are now supported.

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.

> **Warning**
>
> The return value of this property is a best-effort approach. Unfortunately, it can not be guaranteed that sending a message with the returned string will render in the same way as the original message produces the same *entities*/*caption_entities* as the original message. For example, Telegram recommends that entities of type *BLOCKQUOTE* and *PRE* *should* start and end on a new line, but does not enforce this and leaves rendering decisions up to the clients.

Changed in version 13.10: Spoiler entities are now formatted as HTML.

Changed in version 20.3: Custom emoji entities are now supported.

Changed in version 20.8: Blockquote entities are now supported.

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.

Warning

The return value of this property is a best-effort approach. Unfortunately, it can not be guaranteed that sending a message with the returned string will render in the same way as the original message produces the same *entities*/*caption_entities* as the original message. For example, Telegram recommends that entities of type *BLOCKQUOTE* and *PRE* *should* start and end on a new line, but does not enforce this and leaves rendering decisions up to the clients. Moreover, markdown formatting is inherently less expressive than HTML, so some edge cases may not be coverable at all. For example, markdown formatting can not specify two consecutive block quotes without a blank line in between, but HTML can.

Note

'Markdown' is a legacy mode, retained by Telegram for backward compatibility. You should use *caption_markdown_v2()*

Changed in version 20.5: Since custom emoji entities are not supported by *MARKDOWN*, this method now raises a ValueError when encountering a custom emoji.

Changed in version 20.8: Since block quotation entities are not supported by *MARKDOWN*, this method now raises a ValueError when encountering a block quotation.

Returns
Message caption with caption entities formatted as Markdown.

Return type
str

Raises
ValueError – If the message contains underline, strikethrough, spoiler, blockquote 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.

Warning

The return value of this property is a best-effort approach. Unfortunately, it can not be guaranteed that sending a message with the returned string will render in the same way as the original message produces the same *entities*/*caption_entities* as the original message. For example, Telegram recommends that entities of type *BLOCKQUOTE* and *PRE* *should* start and end on a new line, but does not enforce this and leaves rendering decisions up to the clients. Moreover, markdown

formatting is inherently less expressive than HTML, so some edge cases may not be coverable at all. For example, markdown formatting can not specify two consecutive block quotes without a blank line in between, but HTML can.

Note

'Markdown' is a legacy mode, retained by Telegram for backward compatibility. You should use *caption_markdown_v2_urled()* instead.

Changed in version 20.5: Since custom emoji entities are not supported by *MARKDOWN*, this method now raises a `ValueError` when encountering a custom emoji.

Changed in version 20.8: Since block quotation entities are not supported by *MARKDOWN*, this method now raises a `ValueError` when encountering a block quotation.

Returns
Message caption with caption entities formatted as Markdown.

Return type
`str`

Raises
ValueError – If the message contains underline, strikethrough, spoiler, blockquote 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.

Warning

The return value of this property is a best-effort approach. Unfortunately, it can not be guaranteed that sending a message with the returned string will render in the same way as the original message produces the same *entities*/*caption_entities* as the original message. For example, Telegram recommends that entities of type *BLOCKQUOTE* and *PRE* *should* start and end on a new line, but does not enforce this and leaves rendering decisions up to the clients. Moreover, markdown formatting is inherently less expressive than HTML, so some edge cases may not be coverable at all. For example, markdown formatting can not specify two consecutive block quotes without a blank line in between, but HTML can.

Changed in version 13.10: Spoiler entities are now formatted as Markdown V2.

Changed in version 20.3: Custom emoji entities are now supported.

Changed in version 20.8: Blockquote entities are now supported.

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.

> **Warning**
>
> The return value of this property is a best-effort approach. Unfortunately, it can not be guaranteed that sending a message with the returned string will render in the same way as the original message produces the same *entities*/*caption_entities* as the original message. For example, Telegram recommends that entities of type *BLOCKQUOTE* and *PRE* *should* start and end on a new line, but does not enforce this and leaves rendering decisions up to the clients. Moreover, markdown formatting is inherently less expressive than HTML, so some edge cases may not be coverable at all. For example, markdown formatting can not specify two consecutive block quotes without a blank line in between, but HTML can.

Changed in version 13.10: Spoiler entities are now formatted as Markdown V2.

Changed in version 20.3: Custom emoji entities are now supported.

Changed in version 20.8: Blockquote entities are now supported.

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 close_forum_topic(*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.close_forum_topic(
   chat_id=message.chat_id, message_thread_id=message.message_thread_id,␣
↪*args,
   **kwargs
)
```

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

Added in version 20.0.

Returns
On success, True is returned.

Return type
bool

compute_quote_position_and_entities(*quote*, *index=None*)

Use this function to compute position and entities of a quote in the message text or caption. Useful for filling the parameters *quote_position* and *quote_entities* of *telegram.ReplyParameters* when replying to a message.

> **Example**
>
> Given a message with the text `"Hello, world! Hello, world!"`, the following code will return the position and entities of the second occurrence of `"Hello, world!"`.

```
message.compute_quote_position_and_entities("Hello, world!", 1)
```

Added in version 20.8.

Parameters

- ***quote*** (*str*) – Part of the message which is to be quoted. This is expected to have plain text without formatting entities.
- ***index*** (*int*, optional) – 0-based index of the occurrence of the quote in the message. If not specified, the first occurrence is used.

Returns

On success, a tuple containing information about quote position and entities is returned.

Return type

tuple[*int*, *None* | tuple[*MessageEntity*, ...]]

Raises

- **RuntimeError** – If the message has neither *text* nor *caption*.
- **ValueError** – If the requested index of quote doesn't exist in the message.

async copy(*chat_id*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *disable_notification=None*, *reply_markup=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *show_caption_above_media=None*, *allow_paid_broadcast=None*, *video_start_timestamp=None*, *suggested_post_parameters=None*, *message_effect_id=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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,
    direct_messages_topic_id=self.direct_messages_topic.topic_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=None*)

See *telegram.TelegramObject.de_json()*.

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

Shortcut for:

```
await bot.decline_suggested_post(
    chat_id=message.chat_id,
    message_id=message.message_id,
    *args, **kwargs
)
```

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

Added in version 22.4.

Returns
bool On success, True is returned.

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

Shortcut for either:

```
await bot.delete_message(
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs
)
```

or:

```
await bot.delete_business_messages(
    business_connection_id=self.business_connection_id,
    message_ids=[self.message_id],
    *args,
    **kwargs,
)
```

For the documentation of the arguments, please see *telegram.Bot.delete_message()* and *telegram.Bot.delete_business_messages()*.

Changed in version 22.4: Calls either *telegram.Bot.delete_message()* or *telegram.Bot.delete_business_messages()* based on *business_connection_id*.

Returns
On success, True is returned.

Return type
bool

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

Shortcut for:

```
await bot.delete_forum_topic(
   chat_id=message.chat_id, message_thread_id=message.message_thread_id,␣
↪*args,
   **kwargs
)
```

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

Added in version 20.0.

Returns
On success, True is returned.

Return type
bool

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

Shortcut for:

```
await bot.delete_message_reaction(
    chat_id=message.chat_id,
    message_id=message.message_id,
    *args, **kwargs
)
```

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

Added in version 22.8.

Returns
bool On success, True is returned.

async edit_caption(*caption=None*, *reply_markup=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=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,
    business_connection_id=message.business_connection_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.

Changed in version 21.4: Now also passes *business_connection_id*.

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_checklist(*checklist*, *reply_markup=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.edit_message_checklist(
    business_connection_id=message.business_connection_id,
    chat_id=message.chat_id,
    message_id=message.message_id,
    *args, **kwargs
)
```

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

Added in version 22.3.

> **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, the edited Message is returned.

Return type
telegram.Message

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

Shortcut for:

```
await bot.edit_forum_topic(
    chat_id=message.chat_id, message_thread_id=message.message_thread_id,␣
↪*args,
    **kwargs
)
```

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

Added in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

async edit_live_location(*latitude=None*, *longitude=None*, *reply_markup=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, *live_period=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,
    business_connection_id=message.business_connection_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.

Changed in version 21.4: Now also passes *business_connection_id*.

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,
    business_connection_id=message.business_connection_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.

Changed in version 21.4: Now also passes *business_connection_id*.

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,
    business_connection_id=message.business_connection_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.

Changed in version 21.4: Now also passes *business_connection_id*.

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*, *reply_markup=None*, *entities=None*, *link_preview_options=None*, ***, *disable_web_page_preview=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,
    business_connection_id=message.business_connection_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.

Changed in version 21.4: Now also passes *business_connection_id*.

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 the message is a user generated content which is not a plain text message, this property is set to this content. It may be one of

- *telegram.Audio*
- *telegram.Dice*
- *telegram.Contact*
- *telegram.Document*
- *telegram.Animation*
- *telegram.Game*
- *telegram.Invoice*
- *telegram.LivePhoto*
- *telegram.Location*
- *telegram.PassportData*
- list[*telegram.PhotoSize*]
- *telegram.PaidMediaInfo*
- *telegram.Poll*
- *telegram.Sticker*

- *telegram.Story*
- *telegram.SuccessfulPayment*
- *telegram.Venue*
- *telegram.Video*
- *telegram.VideoNote*
- *telegram.Voice*

Otherwise None is returned.

See also

Working with Files and Media

Changed in version 20.0: *dice*, *passport_data* and *poll* are now also considered to be an attachment.

Changed in version 21.4: *paid_media* is now also considered to be an attachment.

Deprecated since version 21.4: *successful_payment* will be removed in future major versions.

async forward(*chat_id*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *video_start_timestamp=None*, *suggested_post_parameters=None*, *message_effect_id=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,
    direct_messages_topic_id=self.direct_messages_topic.topic_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.ChatFullInfo.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
tuple[*telegram.GameHighScore*]

property id

Shortcut for *message_id*.

Added 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.

Changed in version 20.3: For messages that are replies or part of a forum topic, the link now points to the corresponding thread view.

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,
    business_connection_id=message.business_connection_id,
    *args, **kwargs
)
```

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

Changed in version 21.5: Now also passes *business_connection_id* to *telegram.Bot.pin_chat_message()*.

Returns
On success, True is returned.

Return type
bool

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

Shortcut for:

```
await bot.read_business_message(
    chat_id=message.chat_id,
    message_id=message.message_id,
    business_connection_id=message.business_connection_id,
    *args, **kwargs
)
```

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

Added in version 22.1.

Returns
bool On success, True is returned.

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

Shortcut for:

```
await bot.reopen_forum_topic(
    chat_id=message.chat_id, message_thread_id=message.message_thread_id,␣
↪*args,
    **kwargs
)
```

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

Added in version 20.0.

Returns
On success, `True` is returned.

Return type
`bool`

async reply_animation(*animation*, *duration=None*, *width=None*, *height=None*, *caption=None*, *parse_mode=None*, *disable_notification=None*, *reply_markup=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *thumbnail=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *do_quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_animation(
    update.effective_message.chat_id,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=self.business_connection_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

Keyword Arguments
do_quote (`bool` | `dict`, optional) – If set to `True`, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact `reply_parameters`. If `reply_to_message_id` or `reply_parameters` are passed, this parameter will be ignored. When passing dict-valued input, `do_quote` is mutually exclusive with `allow_sending_without_reply`. Default: `True` in group chats and `False` in private chats.

Added in version 20.8.

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_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *do_quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_audio(
    update.effective_message.chat_id,
    message_thread_id=update.effective_message.message_thread_id,
```

(continues on next page)

(continued from previous page)

```
    business_connection_id=self.business_connection_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

Keyword Arguments

do_quote (`bool` | `dict`, optional) – If set to `True`, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact `reply_parameters`. If `reply_to_message_id` or `reply_parameters` are passed, this parameter will be ignored. When passing dict-valued input, `do_quote` is mutually exclusive with `allow_sending_without_reply`. Default: `True` in group chats and `False` in private chats.

Added in version 20.8.

Returns

On success, instance representing the message posted.

Return type

telegram.Message

async reply_chat_action(*action*, *message_thread_id=None*, ***, *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,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=self.business_connection_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Added in version 13.2.

Returns

On success, `True` is returned.

Return type

`bool`

async reply_checklist(*checklist*, *disable_notification=None*, *protect_content=None*, *message_effect_id=None*, *reply_parameters=None*, *reply_markup=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *do_quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_checklist(
    business_connection_id=self.business_connection_id,
    chat_id=update.effective_message.chat_id,
    *args,
    **kwargs,
)
```

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

Added in version 22.3.

Keyword Arguments

do_quote (bool | dict, optional) – If set to True, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact reply_parameters. If reply_to_message_id or reply_parameters are passed, this parameter will be ignored. When passing dict-valued input, do_quote is mutually exclusive with allow_sending_without_reply. Default: True in group chats and False in private chats.

Returns

On success, instance representing the message posted.

Return type

telegram.Message

async reply_contact(*phone_number=None*, *first_name=None*, *last_name=None*, *disable_notification=None*, *reply_markup=None*, *vcard=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *contact=None*, *do_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,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=self.business_connection_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter quote. Use *do_quote* instead.

Keyword Arguments

do_quote (bool | dict, optional) – If set to True, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact reply_parameters. If reply_to_message_id or reply_parameters are passed, this parameter will be ignored. When passing dict-valued input, do_quote is mutually exclusive with allow_sending_without_reply. Default: True in group chats and False in private chats.

Added in version 20.8.

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_markup=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *show_caption_above_media=None*, *allow_paid_broadcast=None*, *video_start_timestamp=None*, *suggested_post_parameters=None*, *message_effect_id=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *do_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_thread_id=update.effective_message.message_thread_id,
    message_id=message_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

Keyword Arguments
do_quote (`bool | dict`, optional) – If set to `True`, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact `reply_parameters`. If `reply_to_message_id` or `reply_parameters` are passed, this parameter will be ignored. When passing dict-valued input, `do_quote` is mutually exclusive with `allow_sending_without_reply`. Default: `True` in group chats and `False` in private chats.

Added in version 20.8.

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

Return type
telegram.MessageId

async reply_dice(*disable_notification=None*, *reply_markup=None*, *emoji=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *do_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,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=self.business_connection_id,
```

(continues on next page)

(continued from previous page)

```
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

Keyword Arguments

do_quote (`bool | dict`, optional) – If set to `True`, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact `reply_parameters`. If `reply_to_message_id` or `reply_parameters` are passed, this parameter will be ignored. When passing dict-valued input, `do_quote` is mutually exclusive with `allow_sending_without_reply`. Default: `True` in group chats and `False` in private chats.

Added in version 20.8.

Returns

On success, instance representing the message posted.

Return type

telegram.Message

async reply_document(*document*, *caption=None*, *disable_notification=None*, *reply_markup=None*, *parse_mode=None*, *disable_content_type_detection=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *do_quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_document(
    update.effective_message.chat_id,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=self.business_connection_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

Keyword Arguments

do_quote (`bool | dict`, optional) – If set to `True`, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact `reply_parameters`. If `reply_to_message_id` or `reply_parameters` are passed, this parameter will be ignored. When passing dict-valued input, `do_quote` is mutually exclusive with `allow_sending_without_reply`. Default: `True` in group chats and `False` in private chats.

Added in version 20.8.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_game(*game_short_name*, *disable_notification=None*, *reply_markup=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *do_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,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=self.business_connection_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

Keyword Arguments
do_quote (`bool | dict`, optional) – If set to `True`, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact `reply_parameters`. If `reply_to_message_id` or `reply_parameters` are passed, this parameter will be ignored. When passing dict-valued input, `do_quote` is mutually exclusive with `allow_sending_without_reply`. Default: `True` in group chats and `False` in private chats.

Added in version 20.8.

Added in version 13.2.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_html(*text*, *disable_notification=None*, *reply_markup=None*, *entities=None*, *protect_content=None*, *message_thread_id=None*, *link_preview_options=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *disable_web_page_preview=None*, *do_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,
    message_thread_id=update.effective_message.message_thread_id,
    parse_mode=ParseMode.HTML,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
```

(continues on next page)

(continued from previous page)

```
    business_connection_id=self.business_connection_id,
    *args,
    **kwargs,
)
```

Sends a message with HTML formatting.

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

Keyword Arguments

do_quote (`bool | dict`, optional) – If set to `True`, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact `reply_parameters`. If `reply_to_message_id` or `reply_parameters` are passed, this parameter will be ignored. When passing dict-valued input, `do_quote` is mutually exclusive with `allow_sending_without_reply`. Default: `True` in group chats and `False` in private chats.

Added in version 20.8.

Returns

On success, instance representing the message posted.

Return type

telegram.Message

async reply_invoice(*title*, *description*, *payload*, *currency*, *prices*, *provider_token=None*, *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_markup=None*, *provider_data=None*, *send_phone_number_to_provider=None*, *send_email_to_provider=None*, *max_tip_amount=None*, *suggested_tip_amounts=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *do_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,
    message_thread_id=update.effective_message.message_thread_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

> **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.

Added in version 13.2.

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

Keyword Arguments

do_quote (bool | dict, optional) – If set to True, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact reply_parameters. If reply_to_message_id or reply_parameters are passed, this parameter will be ignored. When passing dict-valued input, do_quote is mutually exclusive with allow_sending_without_reply. Default: True in group chats and False in private chats.

Added in version 20.8.

Returns

On success, instance representing the message posted.

Return type

telegram.Message

async reply_live_photo(*live_photo*, *photo*, *caption=None*, *disable_notification=None*, *reply_markup=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, *has_spoiler=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *do_quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_live_photo(
    update.effective_message.chat_id,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=self.business_connection_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs,
)
```

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

Added in version 22.8.

Keyword Arguments

do_quote (bool | dict, optional) – If set to True, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact reply_parameters. If reply_to_message_id or reply_parameters are passed, this parameter will be ignored. When passing dict-valued input, do_quote is mutually exclusive with allow_sending_without_reply. 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_markup=None*, *live_period=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *location=None*, *do_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,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=self.business_connection_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

Keyword Arguments

do_quote (`bool` | `dict`, optional) – If set to `True`, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact `reply_parameters`. If `reply_to_message_id` or `reply_parameters` are passed, this parameter will be ignored. When passing dict-valued input, `do_quote` is mutually exclusive with `allow_sending_without_reply`. Default: `True` in group chats and `False` in private chats.

Added in version 20.8.

Returns

On success, instance representing the message posted.

Return type

telegram.Message

async reply_markdown(*text*, *disable_notification=None*, *reply_markup=None*, *entities=None*, *protect_content=None*, *message_thread_id=None*, *link_preview_options=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *disable_web_page_preview=None*, *do_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,
    message_thread_id=update.effective_message.message_thread_id,
    parse_mode=ParseMode.MARKDOWN,
    business_connection_id=self.business_connection_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs,
)
```

Sends a message with Markdown version 1 formatting.

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

> **Note**
>
> *'Markdown'* is a legacy mode, retained by Telegram for backward compatibility. You should use *reply_markdown_v2()* instead.

Keyword Arguments

do_quote (`bool` | `dict`, optional) – If set to `True`, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact `reply_parameters`. If `reply_to_message_id` or `reply_parameters` are passed, this parameter will be ignored. When passing dict-valued input, `do_quote` is mutually exclusive with `allow_sending_without_reply`. Default: `True` in group chats and `False` in private chats.

Added in version 20.8.

Returns

On success, instance representing the message posted.

Return type

telegram.Message

async reply_markdown_v2(*text*, *disable_notification=None*, *reply_markup=None*, *entities=None*, *protect_content=None*, *message_thread_id=None*, *link_preview_options=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *disable_web_page_preview=None*, *do_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,
    message_thread_id=update.effective_message.message_thread_id,
    parse_mode=ParseMode.MARKDOWN_V2,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    business_connection_id=self.business_connection_id,
    *args,
    **kwargs,
)
```

Sends a message with markdown version 2 formatting.

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

Keyword Arguments

do_quote (`bool` | `dict`, optional) – If set to `True`, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact

reply_parameters. If reply_to_message_id or reply_parameters are passed, this parameter will be ignored. When passing dict-valued input, do_quote is mutually exclusive with allow_sending_without_reply. Default: True in group chats and False in private chats.

Added in version 20.8.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_media_group(*media*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *do_quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*, *caption=None*, *parse_mode=None*, *caption_entities=None*)

Shortcut for:

```
await bot.send_media_group(
    update.effective_message.chat_id,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=self.business_connection_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter quote. Use *do_quote* instead.

Keyword Arguments
do_quote (bool | dict, optional) – If set to True, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact reply_parameters. If reply_to_message_id or reply_parameters are passed, this parameter will be ignored. When passing dict-valued input, do_quote is mutually exclusive with allow_sending_without_reply. Default: True in group chats and False in private chats.

Added in version 20.8.

Returns
An array of the sent Messages.

Return type
tuple[*telegram.Message*]

Raises
telegram.error.TelegramError –

async reply_paid_media(*star_count*, *media*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, *disable_notification=None*, *protect_content=None*, *reply_parameters=None*, *reply_markup=None*, *payload=None*, *allow_paid_broadcast=None*, *suggested_post_parameters=None*, *message_thread_id=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *do_quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_paid_media(
    chat_id=message.chat.id,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=message.business_connection_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs
)
```

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

Added in version 21.7.

Keyword Arguments

do_quote (bool | dict, optional) – If set to True, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact reply_parameters. If reply_to_message_id or reply_parameters are passed, this parameter will be ignored. When passing dict-valued input, do_quote is mutually exclusive with allow_sending_without_reply. Default: True in group chats and False in private chats.

Returns

On success, the sent message is returned.

Return type

telegram.Message

async reply_photo(*photo*, *caption=None*, *disable_notification=None*, *reply_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *do_quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_photo(
    update.effective_message.chat_id,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=self.business_connection_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter quote. Use *do_quote* instead.

Keyword Arguments

do_quote (bool | dict, optional) – If set to True, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact reply_parameters. If reply_to_message_id or reply_parameters are passed, this parameter will be ignored. When passing dict-valued input, do_quote is mutually

exclusive with `allow_sending_without_reply`. Default: `True` in group chats and `False` in private chats.

Added in version 20.8.

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_markup=None*, *explanation=None*, *explanation_parse_mode=None*, *open_period=None*, *close_date=None*, *explanation_entities=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *question_parse_mode=None*, *question_entities=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *shuffle_options=None*, *allows_revoting=None*, *correct_option_ids=None*, *allow_adding_options=None*, *hide_results_until_closes=None*, *description=None*, *description_parse_mode=None*, *description_entities=None*, *members_only=None*, *country_codes=None*, *explanation_media=None*, *media=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *do_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,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=self.business_connection_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

Keyword Arguments
do_quote (`bool | dict`, optional) – If set to `True`, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact `reply_parameters`. If `reply_to_message_id` or `reply_parameters` are passed, this parameter will be ignored. When passing dict-valued input, `do_quote` is mutually exclusive with `allow_sending_without_reply`. Default: `True` in group chats and `False` in private chats.

Added in version 20.8.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_sticker(*sticker*, *disable_notification=None*, *reply_markup=None*, *protect_content=None*, *message_thread_id=None*, *emoji=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *do_quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_sticker(
    update.effective_message.chat_id,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=self.business_connection_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

Keyword Arguments

do_quote (`bool` | `dict`, optional) – If set to `True`, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact `reply_parameters`. If `reply_to_message_id` or `reply_parameters` are passed, this parameter will be ignored. When passing dict-valued input, `do_quote` is mutually exclusive with `allow_sending_without_reply`. Default: `True` in group chats and `False` in private chats.

Added in version 20.8.

Returns

On success, instance representing the message posted.

Return type

telegram.Message

async reply_text(*text*, *parse_mode=None*, *disable_notification=None*, *reply_markup=None*, *entities=None*, *protect_content=None*, *message_thread_id=None*, *link_preview_options=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *disable_web_page_preview=None*, *do_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,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=self.business_connection_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

Keyword Arguments

do_quote (`bool | dict`, optional) – If set to `True`, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact `reply_parameters`. If `reply_to_message_id` or `reply_parameters` are passed, this parameter will be ignored. When passing dict-valued input, `do_quote` is mutually exclusive with `allow_sending_without_reply`. Default: `True` in group chats and `False` in private chats.

Added in version 20.8.

Returns

On success, instance representing the message posted.

Return type

telegram.Message

async reply_text_draft(*draft_id*, *text=None*, *parse_mode=None*, *entities=None*, *message_thread_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_message_draft(
    update.effective_message.chat_id,
    message_thread_id=update.effective_message.message_thread_id,
    *args,
    **kwargs,
)
```

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

Note

If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Added in version 22.6.

Changed in version 22.8: Bot API 10.0 makes the `text` argument optional.

Returns

On success, `True` is returned.

Return type

`bool`

async reply_venue(*latitude=None*, *longitude=None*, *title=None*, *address=None*, *foursquare_id=None*, *disable_notification=None*, *reply_markup=None*, *foursquare_type=None*, *google_place_id=None*, *google_place_type=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *venue=None*, *do_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,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=self.business_connection_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

Keyword Arguments

do_quote (`bool` | `dict`, optional) – If set to `True`, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact `reply_parameters`. If `reply_to_message_id` or `reply_parameters` are passed, this parameter will be ignored. When passing dict-valued input, `do_quote` is mutually exclusive with `allow_sending_without_reply`. Default: `True` in group chats and `False` in private chats.

Added in version 20.8.

Returns

On success, instance representing the message posted.

Return type

telegram.Message

async reply_video(*video*, *duration=None*, *caption=None*, *disable_notification=None*, *reply_markup=None*, *width=None*, *height=None*, *parse_mode=None*, *supports_streaming=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *thumbnail=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *cover=None*, *start_timestamp=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *do_quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_video(
    update.effective_message.chat_id,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=self.business_connection_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

Keyword Arguments

do_quote (`bool` | `dict`, optional) – If set to `True`, the replied message is quoted.

For a dict, it must be the output of *build_reply_arguments()* to specify exact `reply_parameters`. If `reply_to_message_id` or `reply_parameters` are passed, this parameter will be ignored. When passing dict-valued input, `do_quote` is mutually exclusive with `allow_sending_without_reply`. Default: `True` in group chats and `False` in private chats.

Added in version 20.8.

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_markup=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *do_quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_video_note(
    update.effective_message.chat_id,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=self.business_connection_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

Keyword Arguments
do_quote (`bool | dict`, optional) – If set to `True`, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact `reply_parameters`. If `reply_to_message_id` or `reply_parameters` are passed, this parameter will be ignored. When passing dict-valued input, `do_quote` is mutually exclusive with `allow_sending_without_reply`. Default: `True` in group chats and `False` in private chats.

Added in version 20.8.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async reply_voice(*voice*, *duration=None*, *caption=None*, *disable_notification=None*, *reply_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *do_quote=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_voice(
    update.effective_message.chat_id,
    message_thread_id=update.effective_message.message_thread_id,
    business_connection_id=self.business_connection_id,
    direct_messages_topic_id=self.direct_messages_topic.topic_id,
    *args,
    **kwargs,
)
```

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

Changed in version 21.1: If *message_thread_id* is not provided, this will reply to the same thread (topic) of the original message.

Changed in version 22.0: Removed deprecated parameter `quote`. Use *do_quote* instead.

Keyword Arguments

do_quote (`bool | dict`, optional) – If set to `True`, the replied message is quoted. For a dict, it must be the output of *build_reply_arguments()* to specify exact `reply_parameters`. If `reply_to_message_id` or `reply_parameters` are passed, this parameter will be ignored. When passing dict-valued input, `do_quote` is mutually exclusive with `allow_sending_without_reply`. Default: `True` in group chats and `False` in private chats.

Added in version 20.8.

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 set_reaction(*reaction=None*, *is_big=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.set_message_reaction(chat_id=message.chat_id, message_id=message.
↪message_id,
    *args, **kwargs)
```

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

Added in version 20.8.

Returns
bool On success, True is returned.

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,
    business_connection_id=message.business_connection_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.

Changed in version 21.4: Now also passes *business_connection_id*.

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,
    business_connection_id=message.business_connection_id,
    *args, **kwargs
)
```

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

Changed in version 21.4: Now also passes *business_connection_id*.

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.

> **Warning**
>
> The return value of this property is a best-effort approach. Unfortunately, it can not be guaranteed that sending a message with the returned string will render in the same way as the original message produces the same *entities*/*caption_entities* as the original message. For example, Telegram recommends that entities of type *BLOCKQUOTE* and *PRE* *should* start and end on a new line, but does not enforce this and leaves rendering decisions up to the clients.

Changed in version 13.10: Spoiler entities are now formatted as HTML.

Changed in version 20.3: Custom emoji entities are now supported.

Changed in version 20.8: Blockquote entities are now supported.

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.

> **Warning**
>
> The return value of this property is a best-effort approach. Unfortunately, it can not be guaranteed that sending a message with the returned string will render in the same way as the original message produces the same *entities*/*caption_entities* as the original message. For example, Telegram recommends that entities of type *BLOCKQUOTE* and *PRE* *should* start and end on a new line, but does not enforce this and leaves rendering decisions up to the clients.

Changed in version 13.10: Spoiler entities are now formatted as HTML.

Changed in version 20.3: Custom emoji entities are now supported.

Changed in version 20.8: Blockquote entities are now supported.

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.

> **Warning**
>
> The return value of this property is a best-effort approach. Unfortunately, it can not be guaranteed that sending a message with the returned string will render in the same way as the original message produces the same *entities*/*caption_entities* as the original message. For example, Telegram recommends that entities of type *BLOCKQUOTE* and *PRE* *should* start and end on a new line, but does not enforce this and leaves rendering decisions up to the clients. Moreover, markdown formatting is inherently less expressive than HTML, so some edge cases may not be coverable at all. For example, markdown formatting can not specify two consecutive block quotes without a blank line in between, but HTML can.

> **Note**
>
> *'Markdown'* is a legacy mode, retained by Telegram for backward compatibility. You should use *text_markdown_v2()* instead.

Changed in version 20.5: Since custom emoji entities are not supported by *MARKDOWN*, this method now raises a `ValueError` when encountering a custom emoji.

Changed in version 20.8: Since block quotation entities are not supported by *MARKDOWN*, this method now raises a `ValueError` when encountering a block quotation.

Returns
Message text with entities formatted as Markdown.

Return type
`str`

Raises
ValueError – If the message contains underline, strikethrough, spoiler, blockquote 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.

> **Warning**
>
> The return value of this property is a best-effort approach. Unfortunately, it can not be guaranteed that sending a message with the returned string will render in the same way as the original message produces the same *entities*/*caption_entities* as the original message. For example, Telegram recommends that entities of type *BLOCKQUOTE* and *PRE* *should* start and end on a new line, but does not enforce this and leaves rendering decisions up to the clients. Moreover, markdown formatting is inherently less expressive than HTML, so some edge cases may not be coverable at all. For example, markdown formatting can not specify two consecutive block quotes without a blank line in between, but HTML can.

> **Note**
>
> *'Markdown'* is a legacy mode, retained by Telegram for backward compatibility. You should use *text_markdown_v2_urled()* instead.

Changed in version 20.5: Since custom emoji entities are not supported by *MARKDOWN*, this method now raises a `ValueError` when encountering a custom emoji.

Changed in version 20.8: Since block quotation entities are not supported by *MARKDOWN*, this method now raises a `ValueError` when encountering a block quotation.

Returns
Message text with entities formatted as Markdown.

Return type
`str`

Raises
ValueError – If the message contains underline, strikethrough, spoiler, blockquote 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.

> **Warning**
>
> The return value of this property is a best-effort approach. Unfortunately, it can not be guaranteed that sending a message with the returned string will render in the same way as the original message produces the same *entities*/*caption_entities* as the original message. For example, Telegram recommends that entities of type *BLOCKQUOTE* and *PRE* *should* start and end on a new line, but does not enforce this and leaves rendering decisions up to the clients. Moreover, markdown formatting is inherently less expressive than HTML, so some edge cases may not be coverable at all. For example, markdown formatting can not specify two consecutive block quotes without a blank line in between, but HTML can.

Changed in version 13.10: Spoiler entities are now formatted as Markdown V2.

Changed in version 20.3: Custom emoji entities are now supported.

Changed in version 20.8: Blockquote entities are now supported.

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.

> **Warning**

The return value of this property is a best-effort approach. Unfortunately, it can not be guaranteed that sending a message with the returned string will render in the same way as the original message produces the same *entities*/*caption_entities* as the original message. For example, Telegram recommends that entities of type *BLOCKQUOTE* and *PRE* *should* start and end on a new line, but does not enforce this and leaves rendering decisions up to the clients. Moreover, markdown formatting is inherently less expressive than HTML, so some edge cases may not be coverable at all. For example, markdown formatting can not specify two consecutive block quotes without a blank line in between, but HTML can.

Changed in version 13.10: Spoiler entities are now formatted as Markdown V2.

Changed in version 20.3: Custom emoji entities are now supported.

Changed in version 20.8: Blockquote entities are now supported.

Returns
Message text with entities formatted as Markdown.

Return type
str

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,
    business_connection_id=message.business_connection_id,
    *args, **kwargs
)
```

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

Changed in version 21.5: Now also passes *business_connection_id* to *telegram.Bot.pin_chat_message()*.

Returns
On success, True is returned.

Return type
bool

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

Shortcut for:

```
await bot.unpin_all_forum_topic_messages(
   chat_id=message.chat_id, message_thread_id=message.message_thread_id,␣
↪*args,
   **kwargs
)
```

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

Added in version 20.0.

Returns
On success, True is returned.

Return type
bool

MessageAutoDeleteTimerChanged

class telegram.**MessageAutoDeleteTimerChanged**(*message_auto_delete_time*, ***, *api_kwargs=None*)

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.

Available In

telegram.Message.message_auto_delete_timer_changed

Added in version 13.4.

Parameters

message_auto_delete_time (int | datetime.timedelta) – New auto-delete time for messages in the chat.

Changed in version v22.2: datetime.timedelta objects are accepted in addition to plain int values.

message_auto_delete_time

New auto-delete time for messages in the chat.

Deprecated since version v22.2: In a future major version this attribute will be of type datetime.timedelta. You can opt-in early by setting *PTB_TIMEDELTA=true* or PTB_TIMEDELTA=1 as an environment variable.

Type
int | datetime.timedelta

MessageEntity

class telegram.**MessageEntity**(*type*, *offset*, *length*, *url=None*, *user=None*, *language=None*, *custom_emoji_id=None*, *date_time_format=None*, *unix_time=None*, ***, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.copy_message()*
- *telegram.Bot.edit_message_caption()*
- *telegram.Bot.edit_message_text()*
- *telegram.Bot.edit_story()*
- *telegram.Bot.gift_premium_subscription()*
- *telegram.Bot.post_story()*

- *telegram.Bot.send_animation()*
- *telegram.Bot.send_audio()*
- *telegram.Bot.send_document()*
- *telegram.Bot.send_gift()*
- *telegram.Bot.send_live_photo()*
- *telegram.Bot.send_media_group()*
- *telegram.Bot.send_message_draft()*
- *telegram.Bot.send_message()*
- *telegram.Bot.send_paid_media()*
- *telegram.Bot.send_photo()*
- *telegram.Bot.send_poll()*
- *telegram.Bot.send_video()*
- *telegram.Bot.send_voice()*

Available In

- *telegram.Checklist.title_entities*
- *telegram.ChecklistTask.text_entities*
- *telegram.Game.text_entities*
- *telegram.GiftInfo.entities*
- *telegram.InlineQueryResultAudio.caption_entities*
- *telegram.InlineQueryResultCachedAudio.caption_entities*
- *telegram.InlineQueryResultCachedDocument.caption_entities*
- *telegram.InlineQueryResultCachedGif.caption_entities*
- *telegram.InlineQueryResultCachedMpeg4Gif.caption_entities*
- *telegram.InlineQueryResultCachedPhoto.caption_entities*
- *telegram.InlineQueryResultCachedVideo.caption_entities*
- *telegram.InlineQueryResultCachedVoice.caption_entities*
- *telegram.InlineQueryResultDocument.caption_entities*
- *telegram.InlineQueryResultGif.caption_entities*
- *telegram.InlineQueryResultMpeg4Gif.caption_entities*
- *telegram.InlineQueryResultPhoto.caption_entities*
- *telegram.InlineQueryResultVideo.caption_entities*
- *telegram.InlineQueryResultVoice.caption_entities*
- *telegram.InputChecklist.title_entities*
- *telegram.InputChecklistTask.text_entities*
- *telegram.InputMedia.caption_entities*
- *telegram.InputMediaAnimation.caption_entities*

- *telegram.InputMediaAudio.caption_entities*
- *telegram.InputMediaDocument.caption_entities*
- *telegram.InputMediaLivePhoto.caption_entities*
- *telegram.InputMediaPhoto.caption_entities*
- *telegram.InputMediaVideo.caption_entities*
- *telegram.InputPollOption.text_entities*
- *telegram.InputTextMessageContent.entities*
- *telegram.Message.caption_entities*
- *telegram.Message.entities*
- *telegram.OwnedGiftRegular.entities*
- *telegram.Poll.description_entities*
- *telegram.Poll.explanation_entities*
- *telegram.Poll.question_entities*
- *telegram.PollOption.text_entities*
- *telegram.PollOptionAdded.option_text_entities*
- *telegram.PollOptionDeleted.option_text_entities*
- *telegram.ReplyParameters.quote_entities*
- *telegram.TextQuote.entities*

Parameters

- ***type*** (str) – Type of the entity. Can be *MENTION* (@username), *HASHTAG* (#hashtag or #hashtag@chatusername), *CASHTAG* ($USD or USD@chatusername), *BOT_COMMAND* (/start@jobs_bot), *URL* (https://telegram.org), *EMAIL* (do-not-reply@telegram.org), *PHONE_NUMBER* (+1-212-555-0123), *BOLD* (**bold text**), *ITALIC* (*italic text*), *UNDERLINE* (underlined text), *STRIKETHROUGH*, *SPOILER* (spoiler message), *BLOCKQUOTE* (block quotation), *CODE* (monowidth string), *PRE* (monowidth block), *TEXT_LINK* (for clickable text URLs), *TEXT_MENTION* (for users without usernames), *CUSTOM_EMOJI* (for inline custom emoji stickers) or *DATE_TIME* (for formatted date and time).

  Added in version 20.0: Added inline custom emoji

  Added in version 20.8: Added block quotation

  Added in version 22.7: Added date_time

- ***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.

  Added in version 20.0.

- ***date_time_format*** (str, optional) – For *DATE_TIME* only, the string that defines the formatting of the date and time. See date-time entity formatting for more details and *telegram.constants.MessageEntityDateTimeFormats* for all possible formats.

  Added in version 22.7.

- ***unix_time*** (datetime.datetime, optional) – For *DATE_TIME* only, the time associated with the entity. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

  Added in version 22.7.

type

Type of the entity. Can be *MENTION* (@username), *HASHTAG* (#hashtag or #hashtag@chatusername), *CASHTAG* ($USD or USD@chatusername), *BOT_COMMAND* (/start@jobs_bot), *URL* (https://telegram.org), *EMAIL* (do-not-reply@telegram.org), *PHONE_NUMBER* (+1-212-555-0123), *BOLD* (**bold text**), *ITALIC* (*italic text*), *UNDERLINE* (underlined text), *STRIKETHROUGH*, *SPOILER* (spoiler message), *BLOCKQUOTE* (block quotation), *CODE* (monowidth string), *PRE* (monowidth block), *TEXT_LINK* (for clickable text URLs), *TEXT_MENTION* (for users without usernames), *CUSTOM_EMOJI* (for inline custom emoji stickers) or *DATE_TIME* (for formatted date and time).

Added in version 20.0: Added inline custom emoji

Added in version 20.8: Added block quotation

Added in version 22.7: Added date_time

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. For *TEXT_LINK* only, url that will be opened after user taps on the text.

Type
str

user

Optional. For *TEXT_MENTION* only, the mentioned user.

Type
telegram.User

language

Optional. For *PRE* only, the programming language of the entity text.

Type
str

custom_emoji_id

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.

Added in version 20.0.

Type
str

date_time_format

Optional. For *DATE_TIME* only, the string that defines the formatting of the date and time. See date-time entity formatting for more details and *telegram.constants.MessageEntityDateTimeFormats* for all possible formats.

Added in version 22.7.

Type
str

unix_time

Optional. For *DATE_TIME* only, the time associated with the entity. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Added in version 22.7.

Type
datetime.datetime

ALL_TYPES = [<MessageEntityType.BLOCKQUOTE>, <MessageEntityType.BOLD>, <MessageEntityType.BOT_COMMAND>, <MessageEntityType.CASHTAG>, <MessageEntityType.CODE>, <MessageEntityType.CUSTOM_EMOJI>, <MessageEntityType.DATE_TIME>, <MessageEntityType.EMAIL>, <MessageEntityType.EXPANDABLE_BLOCKQUOTE>, <MessageEntityType.HASHTAG>, <MessageEntityType.ITALIC>, <MessageEntityType.MENTION>, <MessageEntityType.PHONE_NUMBER>, <MessageEntityType.PRE>, <MessageEntityType.SPOILER>, <MessageEntityType.STRIKETHROUGH>, <MessageEntityType.TEXT_LINK>, <MessageEntityType.TEXT_MENTION>, <MessageEntityType.UNDERLINE>, <MessageEntityType.URL>]

A list of all available message entity types.

Type
list[str]

BLOCKQUOTE = 'blockquote'

telegram.constants.MessageEntityType.BLOCKQUOTE

Added in version 20.8.

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

Added in version 20.0.

DATE_TIME = 'date_time'

telegram.constants.MessageEntityType.DATE_TIME

Added in version 22.7.

EMAIL = 'email'

telegram.constants.MessageEntityType.EMAIL

EXPANDABLE_BLOCKQUOTE = 'expandable_blockquote'

telegram.constants.MessageEntityType.EXPANDABLE_BLOCKQUOTE

Added in version 21.3.

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

Added 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

static adjust_message_entities_to_utf_16(*text*, *entities*)

Utility functionality for converting the offset and length of entities from Unicode (`str`) to UTF-16 (`utf-16-le` encoded `bytes`).

Tip

Only the offsets and lengths calulated in UTF-16 is acceptable by the Telegram Bot API. If they are calculated using the Unicode string (`str` object), errors may occur when the text contains characters that are not in the Basic Multilingual Plane (BMP). For more information, see Unicode and Plane (Unicode).

Added in version 21.4.

Examples

Below is a snippet of code that demonstrates how to use this function to convert entities from Unicode to UTF-16 space. The `unicode_entities` are calculated in Unicode and the *utf_16_entities* are calculated in UTF-16.

```
text = " bold  italic underlined: "
unicode_entities = [
    MessageEntity(offset=2, length=4, type=MessageEntity.BOLD),
    MessageEntity(offset=9, length=6, type=MessageEntity.ITALIC),
    MessageEntity(offset=28, length=3, type=MessageEntity.UNDERLINE),
]
utf_16_entities = MessageEntity.adjust_message_entities_to_utf_16(
    text, unicode_entities
)
await bot.send_message(
    chat_id=123,
    text=text,
    entities=utf_16_entities,
)
# utf_16_entities[0]: offset=3, length=4
# utf_16_entities[1]: offset=11, length=6
# utf_16_entities[2]: offset=30, length=6
```

Parameters

- ***text*** (str) – The text that the entities belong to
- ***entities*** (Sequence[*telegram.MessageEntity*]) – Sequence of entities with offset and length calculated in Unicode

Returns

Sequence of entities with offset and length calculated in UTF-16 encoding

Return type

Sequence[*telegram.MessageEntity*]

classmethod concatenate(**args*)

Utility functionality for concatenating two text along with their formatting entities.

Tip

This function is useful for prefixing an already formatted text with a new text and its formatting entities. In particular, it automatically correctly handles UTF-16 encoding.

Examples

This example shows a callback function that can be used to add a prefix and suffix to the message in a *CallbackQueryHandler*:

```
async def prefix_message(update: Update, context: ContextTypes.DEFAULT_
↪TYPE):
    prefix = " bold  italic underlined:  | "
    prefix_entities = [
        MessageEntity(offset=2, length=4, type=MessageEntity.BOLD),
        MessageEntity(offset=9, length=6, type=MessageEntity.ITALIC),
        MessageEntity(offset=28, length=3, type=MessageEntity.UNDERLINE),
    ]
    suffix = " |  bold  italic underlined: "
    suffix_entities = [
        MessageEntity(offset=5, length=4, type=MessageEntity.BOLD),
        MessageEntity(offset=12, length=6, type=MessageEntity.ITALIC),
        MessageEntity(offset=31, length=3, type=MessageEntity.UNDERLINE),
```

```
    ]

    message = update.effective_message
    first = (prefix, prefix_entities, True)
    second = (message.text, message.entities)
    third = (suffix, suffix_entities, True)

    new_text, new_entities = MessageEntity.concatenate(first, second,␣
↪third)
    await update.callback_query.edit_message_text(
        text=new_text,
        entities=new_entities,
    )
```

Hint

The entities are *not* modified in place. The function returns a new sequence of objects.

Added in version 21.5.

Parameters

***args** (tuple[str, Sequence[*telegram.MessageEntity*]] | tuple[str, Sequence[*telegram.MessageEntity*], bool]) – Arbitrary number of tuples containing the text and its entities to concatenate. If the last element of the tuple is a bool, it is used to determine whether to adjust the entities to UTF-16 via *adjust_message_entities_to_utf_16()*. UTF-16 adjustment is disabled by default.

Returns

The concatenated text and its entities

Return type

tuple[str, Sequence[*telegram.MessageEntity*]]

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

static shift_entities(*by*, *entities*)

Utility functionality for shifting the offset of entities by a given amount.

Examples

Shifting by an integer amount:

```
text = "Hello, world!"
entities = [
    MessageEntity(offset=0, length=5, type=MessageEntity.BOLD),
    MessageEntity(offset=7, length=5, type=MessageEntity.ITALIC),
]
shifted_entities = MessageEntity.shift_entities(1, entities)
await bot.send_message(
    chat_id=123,
    text="!" + text,
    entities=shifted_entities,
)
```

Shifting using a string:

```
text = "Hello, world!"
prefix = ""
entities = [
    MessageEntity(offset=0, length=5, type=MessageEntity.BOLD),
    MessageEntity(offset=7, length=5, type=MessageEntity.ITALIC),
]
shifted_entities = MessageEntity.shift_entities(prefix, entities)
await bot.send_message(
    chat_id=123,
    text=prefix + text,
    entities=shifted_entities,
)
```

Tip

The *entities* are *not* modified in place. The function returns a sequence of new objects.

Added in version 21.5.

Parameters

- ***by*** (str | int) – Either the amount to shift the offset by or a string whose length will be used as the amount to shift the offset by. In this case, UTF-16 encoding will be used to calculate the length.
- ***entities*** (Sequence[*telegram.MessageEntity*]) – Sequence of entities

Returns

Sequence of entities with the offset shifted

Return type

Sequence[*telegram.MessageEntity*]

MessageId

class telegram.**MessageId**(*message_id*, *, *api_kwargs=None*)

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.

Returned In

- *telegram.Bot.copy_message()*
- *telegram.Bot.copy_messages()*
- *telegram.Bot.forward_messages()*

Parameters

message_id (int) – Unique message identifier. In specific instances (e.g., message containing a video sent to a big chat), the server might automatically schedule a message instead of sending it immediately. In such cases, this field will be 0 and the relevant message will be unusable until it is actually sent.

message_id

Unique message identifier. In specific instances (e.g., message containing a video sent to a big chat), the server might automatically schedule a message instead of sending it immediately. In such cases, this field will be `0` and the relevant message will be unusable until it is actually sent.

Type
int

MessageOrigin

class telegram.**MessageOrigin**(*type*, *date*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Base class for telegram MessageOrigin object, it can be one of:

- *MessageOriginUser*
- *MessageOriginHiddenUser*
- *MessageOriginChat*
- *MessageOriginChannel*

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

Available In

- *telegram.ExternalReplyInfo.origin*
- *telegram.Message.forward_origin*

Added in version 20.8.

Parameters

- ***type*** (str) – Type of the message origin, can be on of: *USER*, *HIDDEN_USER*, *CHAT*, or *CHANNEL*.
- ***date*** (datetime.datetime) – Date the message was sent originally. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

type

Type of the message origin, can be on of: *USER*, *HIDDEN_USER*, *CHAT*, or *CHANNEL*.

Type
str

date

Date the message was sent originally. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

CHANNEL = 'channel'

telegram.constants.MessageOriginType.CHANNEL

CHAT = 'chat'

telegram.constants.MessageOriginType.CHAT

HIDDEN_USER = 'hidden_user'

telegram.constants.MessageOriginType.HIDDEN_USER

USER = 'user'

telegram.constants.MessageOriginType.USER

classmethod de_json(*data*, *bot=None*)

Converts JSON data to the appropriate *MessageOrigin* object, i.e. takes care of selecting the correct subclass.

MessageOriginChannel

class telegram.**MessageOriginChannel**(*date*, *chat*, *message_id*, *author_signature=None*, *, *api_kwargs=None*)

Bases: *telegram.MessageOrigin*

The message was originally sent to a channel chat.

Available In

- *telegram.ExternalReplyInfo.origin*
- *telegram.Message.forward_origin*

Added in version 20.8.

Parameters

- ***date*** (datetime.datetime) – Date the message was sent originally. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***chat*** (*telegram.Chat*) – Channel chat to which the message was originally sent.
- ***message_id*** (int) – Unique message identifier inside the chat.
- ***author_signature*** (str, optional) – Signature of the original post author.

type

Type of the message origin. Always *'channel'*.

Type
str

date

Date the message was sent originally. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

chat

Channel chat to which the message was originally sent.

Type
telegram.Chat

message_id

Unique message identifier inside the chat.

Type
int

author_signature

Optional. Signature of the original post author.

Type
str

MessageOriginChat

class telegram.**MessageOriginChat**(*date*, *sender_chat*, *author_signature=None*, ***, *api_kwargs=None*)

Bases: *telegram.MessageOrigin*

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

Available In

- *telegram.ExternalReplyInfo.origin*
- *telegram.Message.forward_origin*

Added in version 20.8.

Parameters

- ***date*** (datetime.datetime) – Date the message was sent originally. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***sender_chat*** (*telegram.Chat*) – Chat that sent the message originally.
- ***author_signature*** (str, optional) – For messages originally sent by an anonymous chat administrator, original message author signature

type

Type of the message origin. Always *'chat'*.

Type
str

date

Date the message was sent originally. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

sender_chat

Chat that sent the message originally.

Type
telegram.Chat

author_signature

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

Type
str

MessageOriginHiddenUser

class telegram.**MessageOriginHiddenUser**(*date*, *sender_user_name*, ***, *api_kwargs=None*)

Bases: *telegram.MessageOrigin*

The message was originally sent by an unknown user.

Available In

- *telegram.ExternalReplyInfo.origin*
- *telegram.Message.forward_origin*

Added in version 20.8.

Parameters

- ***date*** (datetime.datetime) – Date the message was sent originally. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***sender_user_name*** (str) – Name of the user that sent the message originally.

type

Type of the message origin. Always *'hidden_user'*.

Type
str

date

Date the message was sent originally. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

sender_user_name

Name of the user that sent the message originally.

Type
str

MessageOriginUser

class telegram.**MessageOriginUser**(*date*, *sender_user*, *, *api_kwargs=None*)

Bases: *telegram.MessageOrigin*

The message was originally sent by a known user.

Available In

- *telegram.ExternalReplyInfo.origin*
- *telegram.Message.forward_origin*

Added in version 20.8.

Parameters

- ***date*** (datetime.datetime) – Date the message was sent originally. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***sender_user*** (*telegram.User*) – User that sent the message originally.

type

Type of the message origin. Always *'user'*.

Type
str

date

Date the message was sent originally. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

sender_user

User that sent the message originally.

Type
telegram.User

MessageReactionCountUpdated

class telegram.**MessageReactionCountUpdated**(*chat*, *message_id*, *date*, *reactions*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if the *chat*, *message_id*, *date* and *reactions* is equal.

Available In

telegram.Update.message_reaction_count

Added in version 20.8.

Parameters

- ***chat*** (*telegram.Chat*) – The chat containing the message.
- ***message_id*** (int) – Unique message identifier inside the chat.
- ***date*** (datetime.datetime) – Date of the change in Unix time The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***reactions*** (Sequence[*telegram.ReactionCount*]) – List of reactions that are present on the message

chat

The chat containing the message.

Type
telegram.Chat

message_id

Unique message identifier inside the chat.

Type
int

date

Date of the change in Unix time The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

reactions

List of reactions that are present on the message

Type
tuple[*telegram.ReactionCount*]

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

MessageReactionUpdated

class telegram.**MessageReactionUpdated**(*chat*, *message_id*, *date*, *old_reaction*, *new_reaction*, *user=None*, *actor_chat=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if the *chat*, *message_id*, *date*, *old_reaction* and *new_reaction* is equal.

Available In

telegram.Update.message_reaction

Added in version 20.8.

Parameters

- ***chat*** (*telegram.Chat*) – The chat containing the message.
- ***message_id*** (int) – Unique message identifier inside the chat.
- ***date*** (datetime.datetime) – Date of the change in Unix time. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***old_reaction*** (Sequence[*telegram.ReactionType*]) – Previous list of reaction types that were set by the user.
- ***new_reaction*** (Sequence[*telegram.ReactionType*]) – New list of reaction types that were set by the user.
- ***user*** (*telegram.User*, optional) – The user that changed the reaction, if the user isn't anonymous.
- ***actor_chat*** (*telegram.Chat*, optional) – The chat on behalf of which the reaction was changed, if the user is anonymous.

chat

The chat containing the message.

Type
telegram.Chat

message_id

Unique message identifier inside the chat.

Type
int

date

Date of the change in Unix time. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

old_reaction

Previous list of reaction types that were set by the user.

Type
tuple[*telegram.ReactionType*]

new_reaction

New list of reaction types that were set by the user.

Type
tuple[*telegram.ReactionType*]

user

Optional. The user that changed the reaction, if the user isn't anonymous.

Type
telegram.User

actor_chat

Optional. The chat on behalf of which the reaction was changed, if the user is anonymous.

Type
telegram.Chat

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

OwnedGift

class telegram.**OwnedGift**(*type*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object describes a gift received and owned by a user or a chat. Currently, it can be one of:

- *telegram.OwnedGiftRegular*
- *telegram.OwnedGiftUnique*

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

Available In

telegram.OwnedGifts.gifts

Added in version 22.1.

Parameters
type (str) – Type of the owned gift.

type

Type of the owned gift.

Type
str

REGULAR = 'regular'

telegram.constants.OwnedGiftType.REGULAR

UNIQUE = 'unique'

telegram.constants.OwnedGiftType.UNIQUE

classmethod de_json(*data*, *bot=None*)

Converts JSON data to the appropriate *OwnedGift* object, i.e. takes care of selecting the correct subclass.

Parameters

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

Returns

The Telegram object.

OwnedGiftRegular

class telegram.**OwnedGiftRegular**(*gift*, *send_date*, *owned_gift_id=None*, *sender_user=None*, *text=None*, *entities=None*, *is_private=None*, *is_saved=None*, *can_be_upgraded=None*, *was_refunded=None*, *convert_star_count=None*, *prepaid_upgrade_star_count=None*, *is_upgrade_separate=None*, *unique_gift_number=None*, ***, *api_kwargs=None*)

Bases: *telegram.OwnedGift*

Describes a regular gift owned by a user or a chat.

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

Available In

telegram.OwnedGifts.gifts

Added in version 22.1.

Parameters

- ***gift*** (*telegram.Gift*) – Information about the regular gift.
- ***owned_gift_id*** (str, optional) – Unique identifier of the gift for the bot; for gifts received on behalf of business accounts only.
- ***sender_user*** (*telegram.User*, optional) – Sender of the gift if it is a known user.
- ***send_date*** (datetime.datetime) – Date the gift was sent as datetime.datetime. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used..
- ***text*** (str, optional) – Text of the message that was added to the gift.
- ***entities*** (Sequence[*telegram.MessageEntity*], optional) – Special entities that appear in the text.
- ***is_private*** (bool, optional) – True, if the sender and gift text are shown only to the gift receiver; otherwise, everyone will be able to see them.
- ***is_saved*** (bool, optional) – True, if the gift is displayed on the account's profile page; for gifts received on behalf of business accounts only.
- ***can_be_upgraded*** (bool, optional) – True, if the gift can be upgraded to a unique gift; for gifts received on behalf of business accounts only.
- ***was_refunded*** (bool, optional) – True, if the gift was refunded and isn't available anymore.

- ***convert_star_count*** (int, optional) – Number of Telegram Stars that can be claimed by the receiver instead of the gift; omitted if the gift cannot be converted to Telegram Stars; for gifts received on behalf of business accounts only.
- ***prepaid_upgrade_star_count*** (int, optional) – Number of Telegram Stars that were paid for the ability to upgrade the gift.
- ***is_upgrade_separate*** (bool, optional) – True, if the gift's upgrade was purchased after the gift was sent; for gifts received on behalf of business accounts

  Added in version 22.6.
- ***unique_gift_number*** (int, optional) – Unique number reserved for this gift when upgraded. See the number field in *UniqueGift*

  ... versionadded:: 22.6

type

Type of the gift, always *REGULAR*.

Type
str

gift

Information about the regular gift.

Type
telegram.Gift

owned_gift_id

Optional. Unique identifier of the gift for the bot; for gifts received on behalf of business accounts only.

Type
str

sender_user

Optional. Sender of the gift if it is a known user.

Type
telegram.User

send_date

Date the gift was sent as datetime.datetime. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used..

Type
datetime.datetime

text

Optional. Text of the message that was added to the gift.

Type
str

entities

Optional. Special entities that appear in the text.

Type
Sequence[*telegram.MessageEntity*]

is_private

Optional. True, if the sender and gift text are shown only to the gift receiver; otherwise, everyone will be able to see them.

Type
bool

is_saved

Optional. True, if the gift is displayed on the account's profile page; for gifts received on behalf of business accounts only.

Type
bool

can_be_upgraded

Optional. True, if the gift can be upgraded to a unique gift; for gifts received on behalf of business accounts only.

Type
bool

was_refunded

Optional. True, if the gift was refunded and isn't available anymore.

Type
bool

convert_star_count

Optional. Number of Telegram Stars that can be claimed by the receiver instead of the gift; omitted if the gift cannot be converted to Telegram Stars; for gifts received on behalf of business accounts only.

Type
int

prepaid_upgrade_star_count

Optional. Number of Telegram Stars that were paid for the ability to upgrade the gift.

Type
int

is_upgrade_separate

Optional. True, if the gift's upgrade was purchased after the gift was sent; for gifts received on behalf of business accounts

Added in version 22.6.

Type
bool

unique_gift_number

Optional. Unique number reserved for this gift when upgraded. See the number field in *UniqueGift*

... versionadded:: 22.6

Type
int

classmethod de_json(*data*, *bot=None*)

See *telegram.OwnedGift.de_json()*.

parse_entities(*types=None*)

Returns a dict that maps *telegram.MessageEntity* to str. It contains entities from this owned gift's text 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 *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 `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]

Raises
RuntimeError – If the owned gift has no text.

`parse_entity`(*entity*)

Returns the text in *text* from a given *telegram.MessageEntity* of *entities*.

> **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 `OwnedGiftRegular.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 *entities*.

Returns
The text of the given entity.

Return type
str

Raises
RuntimeError – If the owned gift has no text.

OwnedGifts

`class` telegram.`OwnedGifts`(*total_count*, *gifts*, *next_offset=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Contains the list of gifts received and owned by a user or a chat.

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

> **Returned In**
>
> - *telegram.Bot.get_business_account_gifts()*
> - *telegram.Bot.get_chat_gifts()*
> - *telegram.Bot.get_user_gifts()*

Added in version 22.1.

Parameters

- ***total_count*** (int) – The total number of gifts owned by the user or the chat.
- ***gifts*** (Sequence[*telegram.OwnedGift*]) – The list of gifts.

- ***next_offset*** (str, optional) – Offset for the next request. If empty, then there are no more results.

total_count

The total number of gifts owned by the user or the chat.

Type
int

gifts

The list of gifts.

Type
Sequence[*telegram.OwnedGift*]

next_offset

Optional. Offset for the next request. If empty, then there are no more results.

Type
str

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

OwnedGiftUnique

class telegram.**OwnedGiftUnique**(*gift*, *send_date*, *owned_gift_id=None*, *sender_user=None*, *is_saved=None*, *can_be_transferred=None*, *transfer_star_count=None*, *next_transfer_date=None*, ***, *api_kwargs=None*)

Bases: *telegram.OwnedGift*

Describes a unique gift received and owned by a user or a chat.

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

Available In

telegram.OwnedGifts.gifts

Added in version 22.1.

Parameters

- ***gift*** (*telegram.UniqueGift*) – Information about the unique gift.
- ***owned_gift_id*** (str, optional) – Unique identifier of the received gift for the bot; for gifts received on behalf of business accounts only.
- ***sender_user*** (*telegram.User*, optional) – Sender of the gift if it is a known user.
- ***send_date*** (datetime.datetime) – Date the gift was sent as datetime.datetime. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***is_saved*** (bool, optional) – True, if the gift is displayed on the account's profile page; for gifts received on behalf of business accounts only.
- ***can_be_transferred*** (bool, optional) – True, if the gift can be transferred to another owner; for gifts received on behalf of business accounts only.
- ***transfer_star_count*** (int, optional) – Number of Telegram Stars that must be paid to transfer the gift; omitted if the bot cannot transfer the gift.

- ***next_transfer_date*** (datetime.datetime, optional) – Date when the gift can be transferred. If it's in the past, then the gift can be transferred now. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used. .. versionadded:: 22.3

type

Type of the owned gift, always *'unique'*.

Type
str

gift

Information about the unique gift.

Type
telegram.UniqueGift

owned_gift_id

Optional. Unique identifier of the received gift for the bot; for gifts received on behalf of business accounts only.

Type
str

sender_user

Optional. Sender of the gift if it is a known user.

Type
telegram.User

send_date

Date the gift was sent as datetime.datetime. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

is_saved

Optional. True, if the gift is displayed on the account's profile page; for gifts received on behalf of business accounts only.

Type
bool

can_be_transferred

Optional. True, if the gift can be transferred to another owner; for gifts received on behalf of business accounts only.

Type
bool

transfer_star_count

Optional. Number of Telegram Stars that must be paid to transfer the gift; omitted if the bot cannot transfer the gift.

Type
int

next_transfer_date

Optional. Date when the gift can be transferred. If it's in the past, then the gift can be transferred now. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used. .. versionadded:: 22.3

Type
datetime.datetime

classmethod de_json(*data*, *bot=None*)

See *telegram.OwnedGift.de_json()*.

PaidMedia

class telegram.**PaidMedia**(*type*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes the paid media added to a message. Currently, it can be one of:

- *telegram.PaidMediaPreview*
- *telegram.PaidMediaPhoto*
- *telegram.PaidMediaVideo*

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

Available In

- *telegram.PaidMediaInfo.paid_media*
- *telegram.TransactionPartnerUser.paid_media*

Added in version 21.4.

Parameters

type (str) – Type of the paid media.

type

Type of the paid media.

Type

str

LIVE_PHOTO = 'live_photo'

telegram.constants.PaidMediaType.LIVE_PHOTO

PHOTO = 'photo'

telegram.constants.PaidMediaType.PHOTO

PREVIEW = 'preview'

telegram.constants.PaidMediaType.PREVIEW

VIDEO = 'video'

telegram.constants.PaidMediaType.VIDEO

classmethod de_json(*data*, *bot=None*)

Converts JSON data to the appropriate *PaidMedia* object, i.e. takes care of selecting the correct subclass.

Parameters

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

Returns

The Telegram object.

PaidMediaInfo

class telegram.**PaidMediaInfo**(*star_count*, *paid_media*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes the paid media added to a message.

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

Available In

- *telegram.ExternalReplyInfo.paid_media*
- *telegram.Message.effective_attachment*
- *telegram.Message.paid_media*

Added in version 21.4.

Parameters

- ***star_count*** (int) – The number of Telegram Stars that must be paid to buy access to the media.
- ***paid_media*** (Sequence[*telegram.PaidMedia*]) – Information about the paid media.

star_count

The number of Telegram Stars that must be paid to buy access to the media.

Type
int

paid_media

Information about the paid media.

Type
tuple[*telegram.PaidMedia*]

classmethod de_json(*data*, *bot=None*)

Converts JSON data to a Telegram object.

Parameters

- ***data*** (dict[str, ...]) – The JSON data.
- ***bot*** (*telegram.Bot*, optional) – The bot associated with this object. Defaults to None, in which case shortcut methods will not be available.

  Changed in version 21.4: *bot* is now optional and defaults to None

Returns
The Telegram object.

PaidMediaLivePhoto

class telegram.**PaidMediaLivePhoto**(*live_photo*, *, *api_kwargs=None*)

Bases: *telegram.PaidMedia*

The paid media is a live photo.

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

Available In

- *telegram.PaidMediaInfo.paid_media*
- *telegram.TransactionPartnerUser.paid_media*

Added in version 22.8.

Parameters

- ***type*** (str) – Type of the paid media, always *'live_photo'*
- ***live_photo*** (*telegram.LivePhoto*) – The photo.

type

Type of the paid media, always *'live_photo'*

Type
str

live_photo

The photo.

Type
telegram.LivePhoto

classmethod de_json(*data*, *bot=None*)

Converts JSON data to the appropriate *PaidMedia* object, i.e. takes care of selecting the correct subclass.

Parameters

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

Returns
The Telegram object.

PaidMediaPhoto

class telegram.**PaidMediaPhoto**(*photo*, *, *api_kwargs=None*)

Bases: *telegram.PaidMedia*

The paid media is a photo.

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

Available In

- *telegram.PaidMediaInfo.paid_media*
- *telegram.TransactionPartnerUser.paid_media*

Added in version 21.4.

Parameters

- ***type*** (str) – Type of the paid media, always *'photo'*.
- ***photo*** (Sequence[*telegram.PhotoSize*]) – The photo.

type

Type of the paid media, always *'photo'*.

Type
str

photo

The photo.

Type
tuple[*telegram.PhotoSize*]

classmethod de_json(*data*, *bot=None*)

Converts JSON data to the appropriate *PaidMedia* object, i.e. takes care of selecting the correct subclass.

Parameters

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

Returns
The Telegram object.

PaidMediaPreview

class telegram.**PaidMediaPreview**(*width=None*, *height=None*, *duration=None*, ***, *api_kwargs=None*)

Bases: *telegram.PaidMedia*

The paid media isn't available before the payment.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *width*, *height*, and *duration* are equal.

Available In

- *telegram.PaidMediaInfo.paid_media*
- *telegram.TransactionPartnerUser.paid_media*

Added in version 21.4.

Changed in version v22.2: As part of the migration to representing time periods using `datetime.timedelta`, equality comparison now considers integer durations and equivalent timedeltas as equal.

Parameters

- ***type*** (str) – Type of the paid media, always *'preview'*.
- ***width*** (int, optional) – Media width as defined by the sender.
- ***height*** (int, optional) – Media height as defined by the sender.
- ***duration*** (int | datetime.timedelta, optional) – Duration of the media in seconds as defined by the sender.

  Changed in version v22.2: `datetime.timedelta` objects are accepted in addition to plain `int` values.

type

Type of the paid media, always *'preview'*.

Type
str

width

Optional. Media width as defined by the sender.

Type
int

height

Optional. Media height as defined by the sender.

Type
int

duration

Optional. Duration of the media in seconds as defined by the sender.

Deprecated since version v22.2: In a future major version this attribute will be of type `datetime.timedelta`. You can opt-in early by setting *PTB_TIMEDELTA=true* or `PTB_TIMEDELTA=1` as an environment variable.

Type
int | datetime.timedelta

PaidMediaPurchased

class telegram.**PaidMediaPurchased**(*from_user*, *paid_media_payload*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object contains information about a paid media purchase.

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

Note

In Python `from` is a reserved word. Use *from_user* instead.

Available In

telegram.Update.purchased_paid_media

Added in version 21.6.

Parameters

- ***from_user*** (*telegram.User*) – User who purchased the media.
- ***paid_media_payload*** (str) – Bot-specified paid media payload.

from_user

User who purchased the media.

Type
telegram.User

paid_media_payload

Bot-specified paid media payload.

Type
str

classmethod de_json(*data*, *bot=None*)

Converts JSON data to a Telegram object.

Parameters

- ***data*** (dict[str, ...]) – The JSON data.
- ***bot*** (*telegram.Bot*, optional) – The bot associated with this object. Defaults to None, in which case shortcut methods will not be available.

  Changed in version 21.4: *bot* is now optional and defaults to None

Returns

The Telegram object.

PaidMediaVideo

class telegram.**PaidMediaVideo**(*video*, ***, *api_kwargs=None*)

Bases: *telegram.PaidMedia*

The paid media is a video.

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

Available In

- *telegram.PaidMediaInfo.paid_media*
- *telegram.TransactionPartnerUser.paid_media*

Added in version 21.4.

Parameters

- ***type*** (str) – Type of the paid media, always *'video'*.
- ***video*** (*telegram.Video*) – The video.

type

Type of the paid media, always *'video'*.

Type

str

video

The video.

Type

telegram.Video

classmethod de_json(*data*, *bot=None*)

Converts JSON data to the appropriate *PaidMedia* object, i.e. takes care of selecting the correct subclass.

Parameters

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

Returns

The Telegram object.

PaidMessagePriceChanged

class telegram.**PaidMessagePriceChanged**(*paid_message_star_count*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a service message about a change in the price of paid messages within a chat.

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

Available In

telegram.Message.paid_message_price_changed

Added in version 22.1.

Parameters

paid_message_star_count (int) – The new number of Telegram Stars that must be paid by non-administrator users of the supergroup chat for each sent message

paid_message_star_count

The new number of Telegram Stars that must be paid by non-administrator users of the supergroup chat for each sent message

Type

int

PhotoSize

class telegram.**PhotoSize**(*file_id*, *file_unique_id*, *width*, *height*, *file_size=None*, ***, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.get_file()*
- *telegram.Bot.send_live_photo()*
- *telegram.Bot.send_photo()*

Available In

- *telegram.Animation.thumbnail*
- *telegram.Audio.thumbnail*
- *telegram.ChatShared.photo*
- *telegram.Document.thumbnail*
- *telegram.ExternalReplyInfo.photo*
- *telegram.Game.photo*
- *telegram.InputMediaLivePhoto.photo*

- *telegram.InputMediaPhoto.media*
- *telegram.InputPaidMediaLivePhoto.photo*
- *telegram.InputPaidMediaPhoto.media*
- *telegram.LivePhoto.photo*
- *telegram.Message.effective_attachment*
- *telegram.Message.new_chat_photo*
- *telegram.Message.photo*
- *telegram.PaidMediaPhoto.photo*
- *telegram.PollMedia.photo*
- *telegram.SharedUser.photo*
- *telegram.Sticker.thumbnail*
- *telegram.StickerSet.thumbnail*
- *telegram.UserProfilePhotos.photos*
- *telegram.Video.cover*
- *telegram.Video.thumbnail*
- *telegram.VideoNote.thumbnail*

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.

file_id

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

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 –

Poll

class telegram.**Poll**(*id*, *question*, *options*, *total_voter_count*, *is_closed*, *is_anonymous*, *type*, *allows_multiple_answers*, *correct_option_id=None*, *explanation=None*, *explanation_entities=None*, *open_period=None*, *close_date=None*, *question_entities=None*, *allows_revoting=None*, *members_only=None*, *correct_option_ids=None*, *description=None*, *description_entities=None*, *country_codes=None*, *media=None*, *explanation_media=None*, *, *api_kwargs=None*)

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.

Examples

Poll Bot

Available In

- *telegram.ExternalReplyInfo.poll*
- *telegram.Message.effective_attachment*
- *telegram.Message.poll*
- *telegram.Update.poll*

Returned In

telegram.Bot.stop_poll()

Parameters

- **id** (str) – Unique poll identifier.
- **question** (str) – Poll question, *1*- *300* characters.
- **options** (Sequence[*PollOption*]) – List of poll options.

Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.

- ***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.
- ***members_only*** (bool) – True, if voting is limited to users who have been members of the chat where the poll was originally sent for more than *24* hours.

  Added in version 22.8.
- ***correct_option_id*** (int, optional) – A zero based identifier of the correct answer option. Available only for closed polls in the quiz mode, which were sent (not forwarded), by the bot or to a private chat with the bot.

  Deprecated since version 22.8: Use *correct_option_ids* instead.
- ***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*** (Sequence[*telegram.MessageEntity*], optional) – 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.

  – Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.
- ***explanation_media*** (*telegram.PollMedia*, optional) – Media added to the quiz explanation.

  Added in version 22.8.
- ***open_period*** (int | datetime.timedelta, optional) – Amount of time in seconds the poll will be active after creation.

  Changed in version v22.2: datetime.timedelta objects are accepted in addition to plain int values.
- ***close_date*** (datetime.datetime, optional) – Point in time (Unix timestamp) when the poll will be automatically closed. Converted to datetime.datetime.

  Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.
- ***question_entities*** (Sequence[*telegram.MessageEntity*], optional) – Special entities that appear in the *question*. Currently, only custom emoji entities are allowed in poll questions.

  Added in version 21.2.
- ***allows_revoting*** (bool, optional) – True, if the poll allows to change the chosenanswer options.

  Added in version 22.8.
- ***correct_option_ids*** (Sequence[int], optional) – Array of 0-based identifiers of the correct answer options. Available only for polls in quiz mode which are closed or were sent (not forwarded) by the bot or to the private chat with the bot.

  Added in version 22.8.

- ***country_codes*** (Sequence[str], optional) – A list of two-letter ISO 3166-1 alpha-2 country codes indicating the countries from which users can vote in the poll. The country code "FT" is used for users with anonymous numbers. If omitted, then users from any country can participate in the poll.

  Added in version 22.8.

- ***description*** (str, optional) – Description of the poll; for polls inside the *Message* object only.

  Added in version 22.8.

- ***description_entities*** (Sequence[*telegram.MessageEntity*], optional) – Special entities like usernames, URLs, bot commands, etc. that appear in the description

  Added in version 22.8.

- ***media*** (*telegram.PollMedia*, optional) – Media added to the poll description; for polls inside the *Message* object only.

  Added in version 22.8.

id

Unique poll identifier.

Type
str

question

Poll question, *1*- *300* characters.

Type
str

options

List of poll options.

Changed in version 20.0: This attribute is now an immutable tuple.

Type
tuple[*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`

members_only

`True`, if voting is limited to users who have been members of the chat where the poll was originally sent for more than *24* hours.

Added in version 22.8.

Type
`bool`

explanation

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.

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 an immutable tuple.

Changed in version 20.0: This attribute is now always a (possibly empty) list and never `None`.

Type
tuple[*telegram.MessageEntity*]

explanation_media

Optional. Media added to the quiz explanation.

Added in version 22.8.

Type
telegram.PollMedia

open_period

Optional. Amount of time in seconds the poll will be active after creation.

Deprecated since version v22.2: In a future major version this attribute will be of type `datetime.timedelta`. You can opt-in early by setting *PTB_TIMEDELTA=true* or `PTB_TIMEDELTA=1` as an environment variable.

Type
`int | datetime.timedelta`

close_date

Optional. Point in time when the poll will be automatically closed.

Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
`datetime.datetime`

question_entities

Special entities that appear in the *question*. Currently, only custom emoji entities are allowed in poll questions. This list is empty if the question does not contain entities.

Added in version 21.2.

Type
tuple[*telegram.MessageEntity*]

allows_revoting

`True`, if the poll allows to change the chosenanswer options

Added in version 22.8.

Type
`bool`

correct_option_ids

Array of 0-based identifiers of the correct answer options. Available only for polls in quiz mode which are closed or were sent (not forwarded) by the bot or to the private chat with the bot.

Added in version 22.8.

Type
tuple[`int`]

country_codes

Optional. A list of two-letter `ISO 3166-1 alpha-2` country codes indicating the countries from which users can vote in the poll. The country code `"FT"` is used for users with anonymous numbers. If omitted, then users from any country can participate in the poll.

Added in version 22.8.

Type
tuple[`str`]

description

Optional. Description of the poll; for polls inside the Message object only

Added in version 22.8.

Type
`str`

description_entities

Special entities like usernames, URLs, bot commands, etc. that appear in the description

Added in version 22.8.

Type
tuple[*telegram.MessageEntity*]

media

Optional. Media added to the poll description; for polls inside the Message object only.

Added in version 22.8.

Type
telegram.PollMedia

MAX_DESCRIPTION_CHARACTERS = 1024

telegram.constants.PollLimit.MAX_DESCRIPTION_CHARACTERS

Added in version 22.8.

MAX_EXPLANATION_LENGTH = 200

telegram.constants.PollLimit.MAX_EXPLANATION_LENGTH

Added in version 20.0.

MAX_EXPLANATION_LINE_FEEDS = 2

telegram.constants.PollLimit.MAX_EXPLANATION_LINE_FEEDS

Added in version 20.0.

MAX_OPEN_PERIOD = 2628000

telegram.constants.PollLimit.MAX_OPEN_PERIOD

Added in version 20.0.

MAX_OPTION_LENGTH = 100

telegram.constants.PollLimit.MAX_OPTION_LENGTH

Added in version 20.0.

MAX_OPTION_NUMBER = 12

telegram.constants.PollLimit.MAX_OPTION_NUMBER

Added in version 20.0.

MAX_QUESTION_LENGTH = 300

telegram.constants.PollLimit.MAX_QUESTION_LENGTH

Added in version 20.0.

MIN_MEMBERSHIP_HOURS = 24

telegram.constants.PollLimit.MIN_MEMBERSHIP_HOURS

Added in version 22.8.

MIN_OPEN_PERIOD = 5

telegram.constants.PollLimit.MIN_OPEN_PERIOD

Added in version 20.0.

MIN_OPTION_LENGTH = 1

telegram.constants.PollLimit.MIN_OPTION_LENGTH

Added in version 20.0.

MIN_OPTION_NUMBER = 1

telegram.constants.PollLimit.MIN_OPTION_NUMBER

Added in version 20.0.

MIN_QUESTION_LENGTH = 1

telegram.constants.PollLimit.MIN_QUESTION_LENGTH

Added in version 20.0.

QUIZ = 'quiz'

telegram.constants.PollType.QUIZ

REGULAR = 'regular'

telegram.constants.PollType.REGULAR

property correct_option_id

A zero based identifier of the correct answer option. Available only for closed polls in the quiz mode, which were sent (not forwarded), by the bot or to a private chat with the bot.

Deprecated since version 22.8: Use *correct_option_ids* instead.

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

parse_description_entities(*types=None*)

Returns a `dict` that maps *telegram.MessageEntity* to `str`. It contains entities from this polls description filtered by their `type` attribute as the key, and the text that each entity belongs to as the value of the `dict`.

Added in version 22.8.

> **Note**
>
> This method should always be used instead of the *description_entities* attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See *parse_description_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]

Raises

RuntimeError – If the poll has no description.

parse_description_entity(*entity*)

Returns the text in *description* from a given *telegram.MessageEntity* of *description_entities*.

Added in version 22.8.

> **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 *description_entities*.

Returns

The text of the given entity.

Return type

str

Raises

RuntimeError – If the poll has no description.

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]

Raises
RuntimeError – If the poll has no explanation.

parse_explanation_entity(*entity*)

Returns the text in *explanation* from a given *telegram.MessageEntity* of *explanation_entities*.

> **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 *explanation_entities*.

Returns
The text of the given entity.

Return type
str

Raises
RuntimeError – If the poll has no explanation.

parse_question_entities(*types=None*)

Returns a `dict` that maps *telegram.MessageEntity* to `str`. It contains entities from this polls question filtered by their `type` attribute as the key, and the text that each entity belongs to as the value of the `dict`.

Added in version 21.2.

> **Note**
>
> This method should always be used instead of the *question_entities* attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See *parse_question_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_question_entity(*entity*)

Returns the text in *question* from a given *telegram.MessageEntity* of *question_entities*.

Added in version 21.2.

> **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 *question_entities*.

Returns
The text of the given entity.

Return type
str

PollAnswer

class telegram.**PollAnswer**(*poll_id*, *option_ids*, *user=None*, *voter_chat=None*, *option_persistent_ids=None*, ***, *api_kwargs=None*)

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.

> **Available In**
>
> *telegram.Update.poll_answer*

Changed in version 20.5: The order of *option_ids* and *user* is changed in 20.5 as the latter one became optional.

Changed in version 20.6: Backward compatiblity for changed order of *option_ids* and *user* was removed.

Parameters

- ***poll_id*** (str) – Unique poll identifier.
- ***option_ids*** (Sequence[int]) – Identifiers of answer options, chosen by the user. May be empty if the user retracted their vote.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.
- ***option_persistent_ids*** (Sequence[str]) – Persistent identifiers of the chosen answer options. May be empty if the vote was retracted.

  Added in version 22.8.
- ***user*** (*telegram.User*, optional) – The user that changed the answer to the poll, if the voter isn't anonymous. If the voter is anonymous, this field will contain the user *136817688* for backwards compatibility.

Changed in version 20.5: *user* became optional.

- ***voter_chat*** (*telegram.Chat*, optional) – The chat that changed the answer to the poll, if the voter is anonymous.

  Added in version 20.5.

poll_id

Unique poll identifier.

Type
str

option_ids

Identifiers of answer options, chosen by the user. May be empty if the user retracted their vote.

Changed in version 20.0: This attribute is now an immutable tuple.

Type
tuple[int]

option_persistent_ids

Persistent identifiers of the chosen answer options. May be empty if the vote was retracted.

Added in version 22.8.

Type
tuple[str]

user

Optional. The user, who changed the answer to the poll, if the voter isn't anonymous. If the voter is anonymous, this field will contain the user *136817688* for backwards compatibility

Changed in version 20.5: *user* became optional.

Type
telegram.User

voter_chat

Optional. The chat that changed the answer to the poll, if the voter is anonymous.

Added in version 20.5.

Type
telegram.Chat

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

PollMedia

class telegram.**PollMedia**(*animation=None*, *audio=None*, *document=None*, *live_photo=None*, *location=None*, *photo=None*, *sticker=None*, *venue=None*, *video=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

At most one of the optional fields can be present in any given object.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if all of their attributes are equal.

Available In

- *telegram.Poll.explanation_media*

- *telegram.Poll.media*
- *telegram.PollOption.media*

Added in version 22.8.

Parameters

- ***animation*** (*telegram.Animation*, optional) – Media is an animation, information about the animation
- ***audio*** (*telegram.Audio*, optional) – Media is an audio file, information about the file; currently, can't be received in a poll option
- ***document*** (*telegram.Document*, optional) – Media is a general file, information about the file; currently, can't be received in a poll option
- ***live_photo*** (*telegram.LivePhoto*, optional) – Media is a live photo, information about the live photo
- ***location*** (*telegram.Location*, optional) – Media is a shared location, information about the location
- ***photo*** (Sequence[*telegram.PhotoSize*], optional) – Media is a photo, available sizes of the photo
- ***sticker*** (*telegram.Sticker*, optional) – Media is a sticker, information about the sticker; currently, for poll options only
- ***venue*** (*telegram.Venue*, optional) – Media is a venue, information about the venue
- ***video*** (*telegram.Video*, optional) – Media is a video, information about the video

animation

Optional. Media is an animation, information about the animation

Type
telegram.Animation

audio

Optional. Media is an audio file, information about the file; currently, can't be received in a poll option

Type
telegram.Audio

document

Optional. Media is a general file, information about the file; currently, can't be received in a poll option

Type
telegram.Document

live_photo

Media is a live photo, information about the live photo

Type
telegram.LivePhoto, optional

location

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

Type
telegram.Location

photo

Optional. Media is a photo, available sizes of the photo

Type
tuple[*telegram.PhotoSize*]

sticker

Optional. Media is a sticker, information about the sticker; currently, for poll options only

Type
telegram.Sticker

venue

Optional. Media is a venue, information about the venue

Type
telegram.Venue

video

Optional. Media is a video, information about the video

Type
telegram.Video

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

PollOption

class telegram.**PollOption**(*text*, *voter_count*, *text_entities=None*, *added_by_user=None*, *added_by_chat=None*, *addition_date=None*, *media=None*, *persistent_id=None*, ***, *api_kwargs=None*)

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*, *voter_count* and *persistent_id* are equal.

Available In

telegram.Poll.options

Changed in version 22.8: Added attribute *persistent_id* to equality checks.

Parameters

- ***persistent_id*** (str) – Unique identifier of the option, persistent on option addition and deletion.

  Added in version 22.8.

- ***text*** (str) – Option text, *1-100* characters.
- ***voter_count*** (int) – Number of users that voted for this option.
- ***text_entities*** (Sequence[*telegram.MessageEntity*], optional) – Special entities that appear in the option text. Currently, only custom emoji entities are allowed in poll option texts.

  Added in version 21.2.

- ***media*** (*telegram.PollMedia*, optional) – Media added to the poll option.

  Added in version 22.8.

- ***added_by_user*** (*telegram.User*, optional) – User who added the option; omitted if the option wasn't added by a user after poll creation.

  Added in version 22.8.

- ***added_by_chat*** (*telegram.Chat*, optional) – Chat that added the option; omitted if the option wasn't added by a chat after poll creation.

  Added in version 22.8.

- ***addition_date*** (datetime.datetime, optional) – Point in time when the option was added; omitted if the option existed in the original poll.

  Added in version 22.8.

persistent_id

Unique identifier of the option, persistent on option addition and deletion.

Added in version 22.8.

Type
str

text

Option text, *1-100* characters.

Type
str

voter_count

Number of users that voted for this option.

Type
int

text_entities

Special entities that appear in the option text. Currently, only custom emoji entities are allowed in poll option texts. This list is empty if the question does not contain entities.

Added in version 21.2.

Type
tuple[*telegram.MessageEntity*]

media

Optional. Media added to the poll option.

Added in version 22.8.

Type
telegram.PollMedia

added_by_user

Optional. User who added the option; omitted if the option wasn't added by a user after poll creation.

Added in version 22.8.

Type
telegram.User

added_by_chat

Optional. Chat that added the option; omitted if the option wasn't added by a chat after poll creation.

Added in version 22.8.

Type
telegram.Chat

addition_date

Optional. Point in time when the option was added; omitted if the option existed in the original poll.

Added in version 22.8.

Type
datetime.datetime

MAX_LENGTH = 100

telegram.constants.PollLimit.MAX_OPTION_LENGTH

Added in version 20.0.

MIN_LENGTH = 1

telegram.constants.PollLimit.MIN_OPTION_LENGTH

Added in version 20.0.

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

parse_entities(*types=None*)

Returns a dict that maps *telegram.MessageEntity* to str. It contains entities from this polls question 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_entity* for more info.

Added in version 21.2.

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_entity(*entity*)

Returns the text in *text* from a given *telegram.MessageEntity* of *text_entities*.

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.)

Added in version 21.2.

Parameters
entity (*telegram.MessageEntity*) – The entity to extract the text from. It must be an entity that belongs to *text_entities*.

Returns
The text of the given entity.

Return type
str

PollOptionAdded

class telegram.**PollOptionAdded**(*option_persistent_id*, *option_text*, *poll_message=None*, *option_text_entities=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a service message about an option added to a poll.

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

Available In

telegram.Message.poll_option_added

Added in version 22.8.

Parameters

- ***option_persistent_id*** (str) – Unique identifier of the added option.
- ***option_text*** (str) – Option text.
- ***poll_message*** (*telegram.MaybeInaccessibleMessage*, optional) – Message containing the poll to which the option was added, if known. Note that the Message object in this field will not contain the *reply_to_message* field even if it itself is a reply.
- ***option_text_entities*** (Sequence[*telegram.MessageEntity*], optional) – Special entities that appear in the *option_text*.

option_persistent_id
Unique identifier of the added option.

Type
str

option_text
Option text.

Type
str

poll_message
Optional. Message containing the poll to which the option was added, if known. Note that the Message object in this field will not contain the *reply_to_message* field even if it itself is a reply.

Type
telegram.MaybeInaccessibleMessage

option_text_entities
Optional. Special entities that appear in the *option_text*.

Type
tuple[*telegram.MessageEntity*]

classmethod de_json(*data*, *bot=None*)
See *telegram.TelegramObject.de_json()*.

parse_option_text_entities(*types=None*)

Returns a dict that maps *telegram.MessageEntity* to str. It contains entities from this polls option text 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 *option_text_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 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_option_text_entity(*entity*)

Returns the text in *option_text* from a given *telegram.MessageEntity* of *option_text_entities*.

> **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 *option_text_entities*.

Returns

The text of the given entity.

Return type

str

PollOptionDeleted

class telegram.**PollOptionDeleted**(*option_persistent_id*, *option_text*, *poll_message=None*, *option_text_entities=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a service message about an option deleted from a poll.

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

Available In

telegram.Message.poll_option_deleted

Added in version 22.8.

Parameters

- ***option_persistent_id*** (str) – Unique identifier of the deleted option.
- ***option_text*** (str) – Option text.
- ***poll_message*** (*telegram.MaybeInaccessibleMessage*, optional) – Message containing the poll to which the option was deleted, if known. Note that the Message object in this field will not contain the *reply_to_message* field even if it itself is a reply.
- ***option_text_entities*** (Sequence[*telegram.MessageEntity*], optional) – Special entities that appear in the option_text.

option_persistent_id

Unique identifier of the deleted option.

Type
str

option_text

Option text.

Type
str

poll_message

Optional. Message containing the poll to which the option was deleted, if known. Note that the Message object in this field will not contain the *reply_to_message* field even if it itself is a reply.

Type
telegram.MaybeInaccessibleMessage

option_text_entities

Optional. Special entities that appear in the option_text.

Type
tuple[*telegram.MessageEntity*]

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

parse_option_text_entities(*types=None*)

Returns a dict that maps *telegram.MessageEntity* to str. It contains entities from this polls option text 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 *option_text_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 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_option_text_entity(*entity*)

Returns the text in *option_text* from a given *telegram.MessageEntity* of *option_text_entities*.

> **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 *option_text_entities*.

Returns
The text of the given entity.

Return type
str

PreparedKeyboardButton

class telegram.**PreparedKeyboardButton**(*id*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a keyboard button to be used by a user of a Mini App.

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

> **Returned In**
>
> *telegram.Bot.save_prepared_keyboard_button()*

Added in version 22.8.

Parameters
id (str) – Unique identifier of the keyboard button.

id

Unique identifier of the keyboard button.

Type
str

ProximityAlertTriggered

class telegram.**ProximityAlertTriggered**(*traveler*, *watcher*, *distance*, ***, *api_kwargs=None*)

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.

Available In

telegram.Message.proximity_alert_triggered

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=None*)

See *telegram.TelegramObject.de_json()*.

ReactionCount

class telegram.**ReactionCount**(*type*, *total_count*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This class represents a reaction added to a message along with the number of times it was added.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if the *type* and *total_count* is equal.

Available In

telegram.MessageReactionCountUpdated.reactions

Added in version 20.8.

Parameters

- ***type*** (*telegram.ReactionType*) – Type of the reaction.
- ***total_count*** (int) – Number of times the reaction was added.

type

Type of the reaction.

Type
telegram.ReactionType

total_count

Number of times the reaction was added.

Type
int

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

ReactionType

class telegram.**ReactionType**(*type*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Base class for Telegram ReactionType Objects. There exist *telegram.ReactionTypeEmoji*, *telegram.ReactionTypeCustomEmoji* and *telegram.ReactionTypePaid*.

Use In

telegram.Bot.set_message_reaction()

Available In

- *telegram.ChatFullInfo.available_reactions*
- *telegram.MessageReactionUpdated.new_reaction*
- *telegram.MessageReactionUpdated.old_reaction*
- *telegram.ReactionCount.type*
- *telegram.StoryAreaTypeSuggestedReaction.reaction_type*

Added in version 20.8.

Changed in version 21.5: Added paid reaction.

Parameters
type (str) – Type of the reaction. Can be *EMOJI*, *CUSTOM_EMOJI* or *PAID*.

type

Type of the reaction. Can be *EMOJI*, *CUSTOM_EMOJI* or *PAID*.

Type
str

CUSTOM_EMOJI = 'custom_emoji'

telegram.constants.ReactionType.CUSTOM_EMOJI

EMOJI = 'emoji'

telegram.constants.ReactionType.EMOJI

PAID = 'paid'

telegram.constants.ReactionType.PAID

Added in version 21.5.

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

ReactionTypeCustomEmoji

class telegram.**ReactionTypeCustomEmoji**(*custom_emoji_id*, *, *api_kwargs=None*)

Bases: *telegram.ReactionType*

Represents a reaction with a custom emoji.

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

Use In

telegram.Bot.set_message_reaction()

Available In

- *telegram.ChatFullInfo.available_reactions*
- *telegram.MessageReactionUpdated.new_reaction*
- *telegram.MessageReactionUpdated.old_reaction*
- *telegram.ReactionCount.type*
- *telegram.StoryAreaTypeSuggestedReaction.reaction_type*

Added in version 20.8.

Parameters

custom_emoji_id (str) – Custom emoji identifier.

type

Type of the reaction, always *'custom_emoji'*.

Type

str

custom_emoji_id

Custom emoji identifier.

Type

str

ReactionTypeEmoji

class telegram.**ReactionTypeEmoji**(*emoji*, *, *api_kwargs=None*)

Bases: *telegram.ReactionType*

Represents a reaction with a normal emoji.

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

Use In

telegram.Bot.set_message_reaction()

Available In

- *telegram.ChatFullInfo.available_reactions*
- *telegram.MessageReactionUpdated.new_reaction*
- *telegram.MessageReactionUpdated.old_reaction*
- *telegram.ReactionCount.type*
- *telegram.StoryAreaTypeSuggestedReaction.reaction_type*

Added in version 20.8.

Parameters
emoji (str) – Reaction emoji. It can be one of *telegram.constants.ReactionEmoji*.

type

Type of the reaction, always *'emoji'*.

Type
str

emoji

Reaction emoji. It can be one of

Type
str

:const:`telegram.constants.ReactionEmoji`.

ReactionTypePaid

class telegram.**ReactionTypePaid**(*, *api_kwargs=None*)

Bases: *telegram.ReactionType*

The reaction is paid.

Use In

telegram.Bot.set_message_reaction()

Available In

- *telegram.ChatFullInfo.available_reactions*
- *telegram.MessageReactionUpdated.new_reaction*
- *telegram.MessageReactionUpdated.old_reaction*
- *telegram.ReactionCount.type*
- *telegram.StoryAreaTypeSuggestedReaction.reaction_type*

Added in version 21.5.

type

Type of the reaction, always *'paid'*.

Type
str

ReplyKeyboardMarkup

class telegram.**ReplyKeyboardMarkup**(*keyboard*, *resize_keyboard=None*, *one_time_keyboard=None*, *selective=None*, *input_field_placeholder=None*, *is_persistent=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a custom keyboard with reply options. Not supported in channels and for messages sent on behalf of a Telegram Business account.

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.

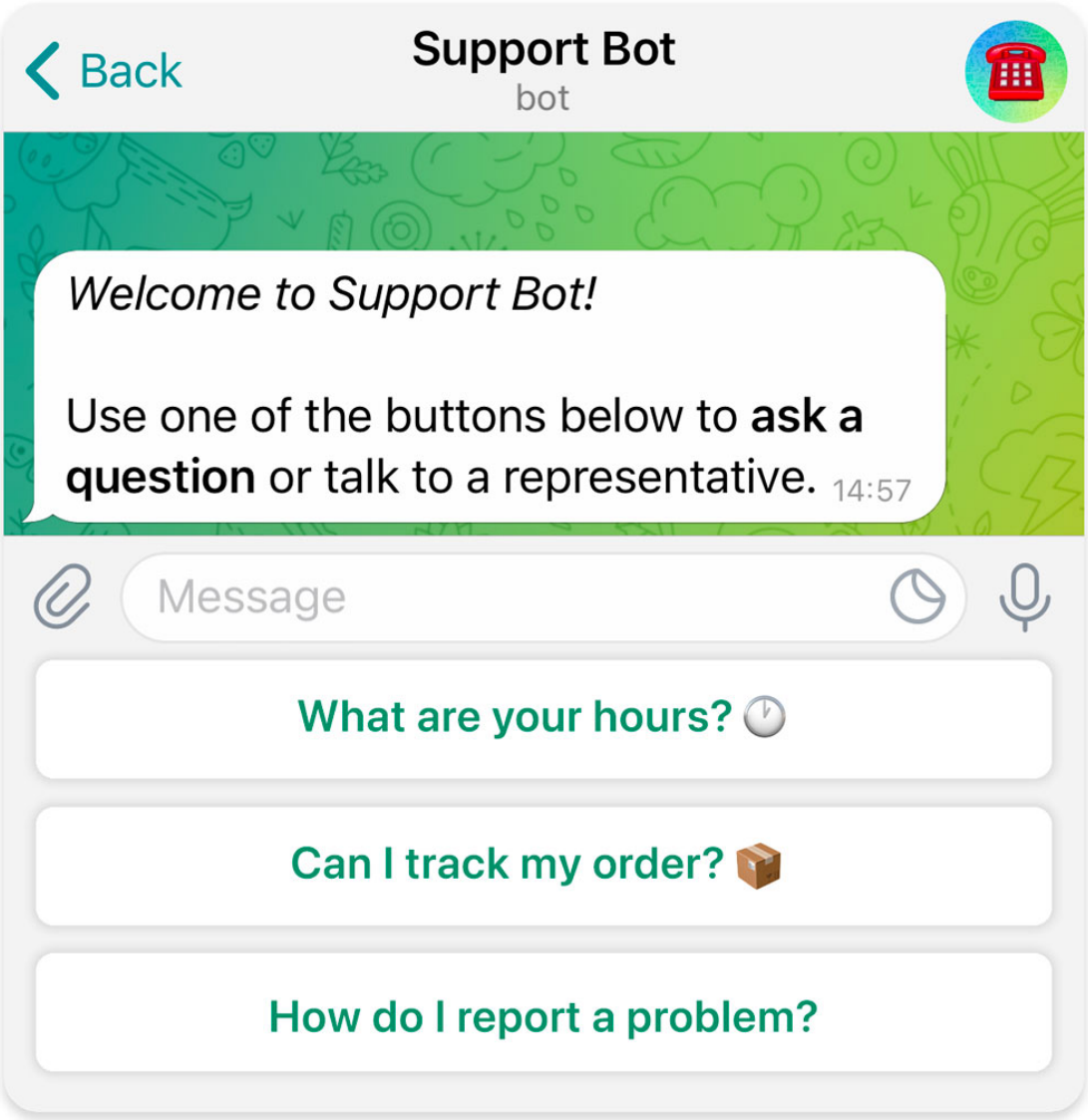

Fig. 2: A reply keyboard with reply options.

See also

Another kind of keyboard would be the *telegram.InlineKeyboardMarkup*.

Examples

- Example usage: 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.
- *Conversation Bot*
- *Conversation Bot 2*

Use In

- *telegram.Bot.copy_message()*
- *telegram.Bot.send_animation()*
- *telegram.Bot.send_audio()*
- *telegram.Bot.send_contact()*
- *telegram.Bot.send_dice()*
- *telegram.Bot.send_document()*
- *telegram.Bot.send_live_photo()*
- *telegram.Bot.send_location()*
- *telegram.Bot.send_message()*
- *telegram.Bot.send_paid_media()*
- *telegram.Bot.send_photo()*
- *telegram.Bot.send_poll()*
- *telegram.Bot.send_sticker()*
- *telegram.Bot.send_venue()*
- *telegram.Bot.send_video_note()*
- *telegram.Bot.send_video()*
- *telegram.Bot.send_voice()*

Parameters

- ***keyboard*** (Sequence[Sequence[`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 to a message in the same chat and forum topic,** 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.

  Added in version 13.7.

- ***is_persistent*** (`bool`, optional) – Requests clients to always show the keyboard when the regular keyboard is hidden. Defaults to `False`, in which case the custom keyboard can be hidden and opened with a keyboard icon.

  Added in version 20.0.

keyboard

Array of button rows, each represented by an Array of *telegram.KeyboardButton* objects.

Type
tuple[tuple[*telegram.KeyboardButton*]]

resize_keyboard

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.

Type
`bool`

one_time_keyboard

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`.

Type
`bool`

selective

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

Defaults to `False`.

Type
`bool`

input_field_placeholder

Optional. The placeholder to be shown in the input field when the keyboard is active; *1- 64* characters.

Added in version 13.7.

Type
`str`

is_persistent

Optional. Requests clients to always show the keyboard when the regular keyboard is hidden. If `False`, the custom keyboard can be hidden and opened with a keyboard icon.

Added in version 20.0.

Type
`bool`

MAX_INPUT_FIELD_PLACEHOLDER = 64

telegram.constants.ReplyLimit.MAX_INPUT_FIELD_PLACEHOLDER

Added in version 20.0.

MIN_INPUT_FIELD_PLACEHOLDER = 1

telegram.constants.ReplyLimit.MIN_INPUT_FIELD_PLACEHOLDER

Added in version 20.0.

classmethod from_button(*button*, *resize_keyboard=False*, *one_time_keyboard=False*, *selective=False*, *input_field_placeholder=None*, *is_persistent=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 to a message in the same chat and forum topic,** 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.

  Added in version 13.7.
- ***is_persistent*** (`bool`) – Optional. Requests clients to always show the keyboard when the regular keyboard is hidden. Defaults to `False`, in which case the custom keyboard can be hidden and opened with a keyboard icon.

  Added in version 20.0.

classmethod from_column(*button_column*, *resize_keyboard=False*, *one_time_keyboard=False*, *selective=False*, *input_field_placeholder=None*, *is_persistent=None*, ***kwargs*)

Shortcut for:

```
ReplyKeyboardMarkup([[button] for button in button_column], **kwargs)
```

Return a ReplyKeyboardMarkup from a single column of KeyboardButtons.

Parameters

- ***button_column*** (Sequence[*telegram.KeyboardButton* | str]) – The button to use in the markup.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list.

- ***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 to a message in the same chat and forum topic,** 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.

  Added in version 13.7.

- ***is_persistent*** (bool) – Optional. Requests clients to always show the keyboard when the regular keyboard is hidden. Defaults to False, in which case the custom keyboard can be hidden and opened with a keyboard icon.

  Added in version 20.0.

classmethod from_row(*button_row*, *resize_keyboard=False*, *one_time_keyboard=False*, *selective=False*, *input_field_placeholder=None*, *is_persistent=None*, ***kwargs*)

Shortcut for:

```
ReplyKeyboardMarkup([button_row], **kwargs)
```

Return a ReplyKeyboardMarkup from a single row of KeyboardButtons.

Parameters

- ***button_row*** (Sequence[*telegram.KeyboardButton* | str]) – The button to use in the markup.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list.

- ***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 to a message in the same chat and forum topic,** 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.

  Added in version 13.7.

- ***is_persistent*** (`bool`) – Optional. Requests clients to always show the keyboard when the regular keyboard is hidden. Defaults to `False`, in which case the custom keyboard can be hidden and opened with a keyboard icon.

  Added in version 20.0.

ReplyKeyboardRemove

class telegram.**ReplyKeyboardRemove**(*selective=None*, ***, *api_kwargs=None*)

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*). Not supported in channels and for messages sent on behalf of a Telegram Business account.

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*.

Examples

- Example usage: 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.
- *Conversation Bot*
- *Conversation Bot 2*

Use In

- *telegram.Bot.copy_message()*
- *telegram.Bot.send_animation()*
- *telegram.Bot.send_audio()*
- *telegram.Bot.send_contact()*
- *telegram.Bot.send_dice()*
- *telegram.Bot.send_document()*
- *telegram.Bot.send_live_photo()*
- *telegram.Bot.send_location()*

- *telegram.Bot.send_message()*
- *telegram.Bot.send_paid_media()*
- *telegram.Bot.send_photo()*
- *telegram.Bot.send_poll()*
- *telegram.Bot.send_sticker()*
- *telegram.Bot.send_venue()*
- *telegram.Bot.send_video_note()*
- *telegram.Bot.send_video()*
- *telegram.Bot.send_voice()*

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

remove_keyboard

Requests clients to remove the custom keyboard.

Type

True

selective

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

Type

bool

ReplyParameters

class telegram.**ReplyParameters**(*message_id*, *chat_id=None*, *allow_sending_without_reply=None*, *quote=None*, *quote_parse_mode=None*, *quote_entities=None*, *quote_position=None*, *checklist_task_id=None*, *poll_option_id=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes reply parameters for the message that is being sent.

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

Use In

- *telegram.Bot.copy_message()*
- *telegram.Bot.send_animation()*
- *telegram.Bot.send_audio()*

- *telegram.Bot.send_checklist()*
- *telegram.Bot.send_contact()*
- *telegram.Bot.send_dice()*
- *telegram.Bot.send_document()*
- *telegram.Bot.send_game()*
- *telegram.Bot.send_invoice()*
- *telegram.Bot.send_live_photo()*
- *telegram.Bot.send_location()*
- *telegram.Bot.send_media_group()*
- *telegram.Bot.send_message()*
- *telegram.Bot.send_paid_media()*
- *telegram.Bot.send_photo()*
- *telegram.Bot.send_poll()*
- *telegram.Bot.send_sticker()*
- *telegram.Bot.send_venue()*
- *telegram.Bot.send_video_note()*
- *telegram.Bot.send_video()*
- *telegram.Bot.send_voice()*

Added in version 20.8.

Changed in version 22.5: The *checklist_task_id* parameter has been moved to the last position to maintain backward compatibility with versions prior to 22.4. This resolves a breaking change accidentally introduced in version 22.4. See the changelog for version 22.5 for more information.

Parameters

- ***message_id*** (int) – Identifier of the message that will be replied to in the current chat, or in the chat *chat_id* if it is specified.
- ***chat_id*** (int | str, optional) – If the message to be replied to is from a different chat, Unique identifier for the target chat or username of the target channel (in the format `@channelusername`). Not supported for messages sent on behalf of a business account and messages from channel direct messages chats.
- ***allow_sending_without_reply*** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found. Can be used only for replies in the same chat and forum topic.
- ***quote*** (str, optional) – Quoted part of the message to be replied to; 0-1024 characters after entities parsing. The quote must be an exact substring of the message to be replied to, including bold, italic, underline, strikethrough, spoiler, custom_emoji, and date_time entities. The message will fail to send if the quote isn't found in the original message.
- ***quote_parse_mode*** (str, optional) – Mode for parsing entities in the quote. See formatting options for more details.
- ***quote_entities*** (Sequence[*telegram.MessageEntity*], optional) – A JSON-serialized list of special entities that appear in the quote. It can be specified instead of *quote_parse_mode*.
- ***quote_position*** (int, optional) – Position of the quote in the original message in UTF-16 code units.

- ***checklist_task_id*** (int, optional) – Identifier of the specific checklist task to be replied to.

  Added in version 22.4.

- ***poll_option_id*** (str, optional) – Persistent identifier of the specific poll option to be replied to.

  Added in version 22.8.

message_id

Identifier of the message that will be replied to in the current chat, or in the chat *chat_id* if it is specified.

Type
int

chat_id

Optional. If the message to be replied to is from a different chat, Unique identifier for the target chat or username of the target channel (in the format `@channelusername`). Not supported for messages sent on behalf of a business account and messages from channel direct messages chats.

Type
int | str

allow_sending_without_reply

Optional. Pass `True`, if the message should be sent even if the specified replied-to message is not found. Can be used only for replies in the same chat and forum topic.

Type
bool

quote

Optional. Quoted part of the message to be replied to; 0-1024 characters after entities parsing. The quote must be an exact substring of the message to be replied to, including bold, italic, underline, strikethrough, spoiler, custom_emoji, and date_time entities. The message will fail to send if the quote isn't found in the original message.

Type
str

quote_parse_mode

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

Type
str

quote_entities

Optional. A JSON-serialized list of special entities that appear in the quote. It can be specified instead of *quote_parse_mode*.

Type
tuple[*telegram.MessageEntity*]

quote_position

Optional. Position of the quote in the original message in UTF-16 code units.

Type
int

checklist_task_id

Optional. Identifier of the specific checklist task to be replied to.

Added in version 22.4.

Type
int

poll_option_id

Optional. Persistent identifier of the specific poll option to be replied to.

Added in version 22.8.

Type
str

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

SentGuestMessage

class telegram.**SentGuestMessage**(*inline_message_id*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes an inline message sent by a guest bot.

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.

Returned In

telegram.Bot.answer_guest_query()

Added in version 22.8.

Parameters
inline_message_id (str) – Identifier of the sent inline message.

inline_message_id

Identifier of the sent inline message.

Type
str

SentWebAppMessage

class telegram.**SentWebAppMessage**(*inline_message_id=None*, *, *api_kwargs=None*)

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.

Returned In

telegram.Bot.answer_web_app_query()

Added 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

SharedUser

class telegram.**SharedUser**(*user_id*, *first_name=None*, *last_name=None*, *username=None*, *photo=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object contains information about a user that was shared with the bot using a *telegram.KeyboardButtonRequestUsers* button.

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

Available In

telegram.UsersShared.users

Added in version 21.1.

Parameters

- ***user_id*** (int) – Identifier of the shared user. This number may have 32 significant bits and some programming languages may have difficulty/silent defects in interpreting it. But it has atmost 52 significant bits, so 64-bit integers or double-precision float types are safe for storing these identifiers. The bot may not have access to the user and could be unable to use this identifier, unless the user is already known to the bot by some other means.
- ***first_name*** (str, optional) – First name of the user, if the name was requested by the bot.
- ***last_name*** (str, optional) – Last name of the user, if the name was requested by the bot.
- ***username*** (str, optional) – Username of the user, if the username was requested by the bot.
- ***photo*** (Sequence[*telegram.PhotoSize*], optional) – Available sizes of the chat photo, if the photo was requested by the bot.

user_id

Identifier of the shared user. This number may have 32 significant bits and some programming languages may have difficulty/silent defects in interpreting it. But it has atmost 52 significant bits, so 64-bit integers or double-precision float types are safe for storing these identifiers. The bot may not have access to the user and could be unable to use this identifier, unless the user is already known to the bot by some other means.

Type
int

first_name

Optional. First name of the user, if the name was requested by the bot.

Type
str

last_name

Optional. Last name of the user, if the name was requested by the bot.

Type
str

username

Optional. Username of the user, if the username was requested by the bot.

Type
str

photo

Available sizes of the chat photo, if the photo was requested by the bot. This list is empty if the photo was not requsted.

Type
tuple[*telegram.PhotoSize*]

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

property full_name

Convenience property. If *first_name* is not None, gives *first_name* followed by (if available) *last_name*.

Added in version 22.4.

Type
str

property link

Convenience property. If *username* is available, returns a t.me link of the user.

Added in version 22.4.

Type
str

property name

Convenience property. If available, returns the user's *username* prefixed with "@". If *username* is not available, returns *full_name*.

Added in version 22.4.

Type
str

Story

class telegram.**Story**(*chat*, *id*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a story.

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

Available In

- *telegram.ExternalReplyInfo.story*
- *telegram.Message.effective_attachment*

- *telegram.Message.reply_to_story*
- *telegram.Message.story*

Returned In

- *telegram.Bot.edit_story()*
- *telegram.Bot.post_story()*
- *telegram.Bot.repost_story()*

Added in version 20.5.

Changed in version 21.0: Added attributes *chat* and *id* and equality based on them.

Parameters

- ***chat*** (*telegram.Chat*) – Chat that posted the story.
- ***id*** (*int*) – Unique identifier for the story in the chat.

chat

Chat that posted the story.

Type
telegram.Chat

id

Unique identifier for the story in the chat.

Type
int

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

async repost(*business_connection_id*, *active_period*, *post_to_chat_page=None*, *protect_content=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.repost_story(
    from_chat_id=story.chat.id,
    from_story_id=story.id,
    *args, **kwargs
)
```

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

Added in version 22.6.

Returns
On success, *Story* is returned.

Return type
Story

StoryArea

class telegram.**StoryArea**(*position*, *type*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a clickable area on a story media.

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

Use In

- *telegram.Bot.edit_story()*
- *telegram.Bot.post_story()*

Added in version 22.1.

Parameters

- ***position*** (*telegram.StoryAreaPosition*) – Position of the area.
- ***type*** (*telegram.StoryAreaType*) – Type of the area.

position

Position of the area.

Type
telegram.StoryAreaPosition

type

Type of the area.

Type
telegram.StoryAreaType

StoryAreaPosition

class telegram.**StoryAreaPosition**(*x_percentage*, *y_percentage*, *width_percentage*, *height_percentage*, *rotation_angle*, *corner_radius_percentage*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes the position of a clickable area within a story.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if all of their attributes are equal.

Available In

telegram.StoryArea.position

Added in version 22.1.

Parameters

- ***x_percentage*** (`float`) – The abscissa of the area's center, as a percentage of the media width.
- ***y_percentage*** (`float`) – The ordinate of the area's center, as a percentage of the media height.
- ***width_percentage*** (`float`) – The width of the area's rectangle, as a percentage of the media width.

- ***height_percentage*** (float) – The height of the area's rectangle, as a percentage of the media height.
- ***rotation_angle*** (float) – The clockwise rotation angle of the rectangle, in degrees; 0-*360*.
- ***corner_radius_percentage*** (float) – The radius of the rectangle corner rounding, as a percentage of the media width.

x_percentage

The abscissa of the area's center, as a percentage of the media width.

Type
float

y_percentage

The ordinate of the area's center, as a percentage of the media height.

Type
float

width_percentage

The width of the area's rectangle, as a percentage of the media width.

Type
float

height_percentage

The height of the area's rectangle, as a percentage of the media height.

Type
float

rotation_angle

The clockwise rotation angle of the rectangle, in degrees; 0-*360*.

Type
float

corner_radius_percentage

The radius of the rectangle corner rounding, as a percentage of the media width.

Type
float

StoryAreaType

class telegram.**StoryAreaType**(*type*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes the type of a clickable area on a story. Currently, it can be one of:

- *telegram.StoryAreaTypeLocation*
- *telegram.StoryAreaTypeSuggestedReaction*
- *telegram.StoryAreaTypeLink*
- *telegram.StoryAreaTypeWeather*
- *telegram.StoryAreaTypeUniqueGift*

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

Available In

telegram.StoryArea.type

Added in version 22.1.

Parameters

type (str) – Type of the area.

type

Type of the area.

Type

str

LINK = 'link'

telegram.constants.StoryAreaTypeType.LINK

LOCATION = 'location'

telegram.constants.StoryAreaTypeType.LOCATION

SUGGESTED_REACTION = 'suggested_reaction'

telegram.constants.StoryAreaTypeType.SUGGESTED_REACTION

UNIQUE_GIFT = 'unique_gift'

telegram.constants.StoryAreaTypeType.UNIQUE_GIFT

WEATHER = 'weather'

telegram.constants.StoryAreaTypeType.WEATHER

StoryAreaTypeLink

class telegram.**StoryAreaTypeLink**(*url*, ***, *api_kwargs=None*)

Bases: *telegram.StoryAreaType*

Describes a story area pointing to an HTTP or `tg://` link. Currently, a story can have up to *3* link areas.

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

Available In

telegram.StoryArea.type

Added in version 22.1.

Parameters

url (str) – HTTP or `tg://` URL to be opened when the area is clicked.

type

Type of the area, always *LINK*.

Type

str

url

HTTP or `tg://` URL to be opened when the area is clicked.

Type

str

StoryAreaTypeLocation

class telegram.**StoryAreaTypeLocation**(*latitude*, *longitude*, *address=None*, ***, *api_kwargs=None*)

Bases: *telegram.StoryAreaType*

Describes a story area pointing to a location. Currently, a story can have up to *10* location areas.

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.

Available In

telegram.StoryArea.type

Added in version 22.1.

Parameters

- ***latitude*** (float) – Location latitude in degrees.
- ***longitude*** (float) – Location longitude in degrees.
- ***address*** (*telegram.LocationAddress*, optional) – Address of the location.

type

Type of the area, always *LOCATION*.

Type
str

latitude

Location latitude in degrees.

Type
float

longitude

Location longitude in degrees.

Type
float

address

Optional. Address of the location.

Type
telegram.LocationAddress

StoryAreaTypeSuggestedReaction

class telegram.**StoryAreaTypeSuggestedReaction**(*reaction_type*, *is_dark=None*, *is_flipped=None*, ***, *api_kwargs=None*)

Bases: *telegram.StoryAreaType*

Describes a story area pointing to a suggested reaction. Currently, a story can have up to *5* suggested reaction areas.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *reaction_type*, *is_dark* and *is_flipped* are equal.

Available In

telegram.StoryArea.type

Added in version 22.1.

Parameters

- ***reaction_type*** (*ReactionType*) – Type of the reaction.
- ***is_dark*** (bool, optional) – Pass True if the reaction area has a dark background.
- ***is_flipped*** (bool, optional) – Pass True if reaction area corner is flipped.

type

Type of the area, always *'suggested_reaction'*.

Type
str

reaction_type

Type of the reaction.

Type
ReactionType

is_dark

Optional. Pass True if the reaction area has a dark background.

Type
bool

is_flipped

Optional. Pass True if reaction area corner is flipped.

Type
bool

StoryAreaTypeUniqueGift

class telegram.**StoryAreaTypeUniqueGift**(*name*, *, *api_kwargs=None*)

Bases: *telegram.StoryAreaType*

Describes a story area pointing to a unique gift. Currently, a story can have at most *1* unique gift area.

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

Available In

telegram.StoryArea.type

Added in version 22.1.

Parameters

name (str) – Unique name of the gift.

type

Type of the area, always *'unique_gift'*.

Type
str

name

Unique name of the gift.

Type
str

StoryAreaTypeWeather

class telegram.**StoryAreaTypeWeather**(*temperature*, *emoji*, *background_color*, *, *api_kwargs=None*)

Bases: *telegram.StoryAreaType*

Describes a story area containing weather information. Currently, a story can have up to *3* weather areas.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *temperature*, *emoji* and *background_color* are equal.

Available In

telegram.StoryArea.type

Added in version 22.1.

Parameters

- ***temperature*** (float) – Temperature, in degree Celsius.
- ***emoji*** (str) – Emoji representing the weather.
- ***background_color*** (int) – A color of the area background in the ARGB format.

type

Type of the area, always *'weather'*.

Type
str

temperature

Temperature, in degree Celsius.

Type
float

emoji

Emoji representing the weather.

Type
str

background_color

A color of the area background in the ARGB format.

Type
int

SuggestedPostApprovalFailed

class telegram.**SuggestedPostApprovalFailed**(*price*, *suggested_post_message=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a service message about the failed approval of a suggested post. Currently, only caused by insufficient user funds at the time of approval.

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

Available In

telegram.Message.suggested_post_approval_failed

Added in version 22.4.

Parameters

- ***suggested_post_message*** (*telegram.Message*, optional) – Message containing the suggested post. Note that the *Message* object in this field will not contain the *reply_to_message* field even if it itself is a reply.
- ***price*** (*SuggestedPostPrice*) – Expected price of the post.

suggested_post_message

Optional. Message containing the suggested post. Note that the *Message* object in this field will not contain the *reply_to_message* field even if it itself is a reply.

Type
telegram.Message

price

Expected price of the post.

Type
SuggestedPostPrice

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

SuggestedPostApproved

class telegram.**SuggestedPostApproved**(*send_date*, *suggested_post_message=None*, *price=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a service message about the approval of a suggested post.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if all of their attributes are equal.

Available In

telegram.Message.suggested_post_approved

Added in version 22.4.

Parameters

- ***suggested_post_message*** (*telegram.Message*, optional) – Message containing the suggested post. Note that the *Message* object in this field will not contain the *reply_to_message* field even if it itself is a reply.
- ***price*** (*SuggestedPostPrice*, optional) – Amount paid for the post.
- ***send_date*** (datetime.datetime) – Date when the post will be published. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

suggested_post_message

Optional. Message containing the suggested post. Note that the *Message* object in this field will not contain the *reply_to_message* field even if it itself is a reply.

Type
telegram.Message

price

Optional. Amount paid for the post.

Type
SuggestedPostPrice

send_date

Date when the post will be published. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

SuggestedPostDeclined

class telegram.**SuggestedPostDeclined**(*suggested_post_message=None*, *comment=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a service message about the rejection of a suggested post.

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

Available In

telegram.Message.suggested_post_declined

Added in version 22.4.

Parameters

- ***suggested_post_message*** (*telegram.Message*, optional) – Message containing the suggested post. Note that the *Message* object in this field will not contain the *reply_to_message* field even if it itself is a reply.
- ***comment*** (str, optional) – Comment with which the post was declined.

suggested_post_message

Optional. Message containing the suggested post. Note that the *Message* object in this field will not contain the *reply_to_message* field even if it itself is a reply.

Type
telegram.Message

comment

Optional. Comment with which the post was declined.

Type
str

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

SuggestedPostInfo

class telegram.**SuggestedPostInfo**(*state*, *price=None*, *send_date=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Contains information about a suggested post.

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

Available In

telegram.Message.suggested_post_info

Added in version 22.4.

Parameters

- ***state*** (str) – State of the suggested post. Currently, it can be one of *'pending'*, *'approved'*, *'declined'*.
- ***price*** (*SuggestedPostPrice*, optional) – Proposed price of the post. If the field is omitted, then the post is unpaid.
- ***send_date*** (datetime.datetime, optional) – Proposed send date of the post. If the field is omitted, then the post can be published at any time within 30 days at the sole discretion of the user or administrator who approves it. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

state

State of the suggested post. Currently, it can be one of *'pending'*, *'approved'*, *'declined'*.

Type
str

price

Optional. Proposed price of the post. If the field is omitted, then the post is unpaid.

Type
SuggestedPostPrice

send_date

Optional. Proposed send date of the post. If the field is omitted, then the post can be published at any time within 30 days at the sole discretion of the user or administrator who approves it. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

APPROVED = 'approved'

telegram.constants.SuggestedPostInfoState.APPROVED

DECLINED = 'declined'

telegram.constants.SuggestedPostInfoState.DECLINED

PENDING = 'pending'

telegram.constants.SuggestedPostInfoState.PENDING

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

SuggestedPostPaid

class telegram.**SuggestedPostPaid**(*currency*, *suggested_post_message=None*, *amount=None*, *star_amount=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a service message about a successful payment for a suggested post.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if all of their attributes are equal.

Available In

telegram.Message.suggested_post_paid

Added in version 22.4.

Parameters

- ***suggested_post_message*** (*telegram.Message*, optional) – Message containing the suggested post. Note that the *Message* object in this field will not contain the *reply_to_message* field even if it itself is a reply.
- ***currency*** (str) – Currency in which the payment was made. Currently, one of "XTR" for Telegram Stars or "TON" for toncoins.
- ***amount*** (int, optional) – The amount of the currency that was received by the channel in nanotoncoins; for payments in toncoins only.
- ***star_amount*** (*telegram.StarAmount*, optional) – The amount of Telegram Stars that was received by the channel; for payments in Telegram Stars only.

suggested_post_message

Optional. Message containing the suggested post. Note that the *Message* object in this field will not contain the *reply_to_message* field even if it itself is a reply.

Type
telegram.Message

currency

Currency in which the payment was made. Currently, one of "XTR" for Telegram Stars or "TON" for toncoins.

Type
str

amount

Optional. The amount of the currency that was received by the channel in nanotoncoins; for payments in toncoins only.

Type
int

star_amount

Optional. The amount of Telegram Stars that was received by the channel; for payments in Telegram Stars only.

Type
telegram.StarAmount

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

SuggestedPostParameters

class telegram.**SuggestedPostParameters**(*price=None*, *send_date=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Contains parameters of a post that is being suggested by the bot.

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

Use In

- *telegram.Bot.copy_message()*
- *telegram.Bot.forward_message()*
- *telegram.Bot.send_animation()*
- *telegram.Bot.send_audio()*
- *telegram.Bot.send_contact()*
- *telegram.Bot.send_dice()*
- *telegram.Bot.send_document()*
- *telegram.Bot.send_invoice()*
- *telegram.Bot.send_live_photo()*
- *telegram.Bot.send_location()*
- *telegram.Bot.send_message()*
- *telegram.Bot.send_paid_media()*
- *telegram.Bot.send_photo()*
- *telegram.Bot.send_sticker()*
- *telegram.Bot.send_venue()*
- *telegram.Bot.send_video_note()*
- *telegram.Bot.send_video()*
- *telegram.Bot.send_voice()*

Added in version 22.4.

Parameters

- ***price*** (*telegram.SuggestedPostPrice*, optional) – Proposed price for the post. If the field is omitted, then the post is unpaid.
- ***send_date*** (datetime.datetime, optional) – Proposed send date of the post. If specified, then the date must be between *300* second and *2678400* seconds (30 days) in the future. If the field is omitted, then the post can be published at any time within *2678400* seconds (30 days) at the sole discretion of the user who approves it. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

price

Optional. Proposed price for the post. If the field is omitted, then the post is unpaid.

Type

telegram.SuggestedPostPrice

send_date

Optional. Proposed send date of the post. If specified, then the date must be between *300* second and *2678400* seconds (30 days) in the future. If the field is omitted, then the post can be published at any time within *2678400* seconds (30 days) at the sole discretion of the user who approves it. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

SuggestedPostPrice

class telegram.**SuggestedPostPrice**(*currency*, *amount*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Desribes the price of a suggested post.

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

Available In

- *telegram.SuggestedPostApprovalFailed.price*
- *telegram.SuggestedPostApproved.price*
- *telegram.SuggestedPostInfo.price*
- *telegram.SuggestedPostParameters.price*

Added in version 22.4.

Parameters

- ***currency*** (str) – Currency in which the post will be paid. Currently, must be one of "XTR" for Telegram Stars or "TON" for toncoins.
- ***amount*** (int) – The amount of the currency that will be paid for the post in the smallest units of the currency, i.e. Telegram Stars or nanotoncoins. Currently, price in Telegram Stars must be between *5* and *100000*, and price in nanotoncoins must be between *10000000* and *10000000000000*.

currency

Currency in which the post will be paid. Currently, must be one of "XTR" for Telegram Stars or "TON" for toncoins.

Type
str

amount

The amount of the currency that will be paid for the post in the smallest units of the currency, i.e. Telegram Stars or nanotoncoins. Currently, price in Telegram Stars must be between *5* and *100000*, and price in nanotoncoins must be between *10000000* and *10000000000000*.

Type
int

SuggestedPostRefunded

class telegram.**SuggestedPostRefunded**(*reason*, *suggested_post_message=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a service message about a payment refund for a suggested post.

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

Available In

telegram.Message.suggested_post_refunded

Added in version 22.4.

Parameters

- ***suggested_post_message*** (*telegram.Message*, optional) – Message containing the suggested post. Note that the *Message* object in this field will not contain the *reply_to_message* field even if it itself is a reply.
- ***reason*** (str) – Reason for the refund. Currently, one of *'post_deleted'* if the post was deleted within 24 hours of being posted or removed from scheduled messages without being posted, or *'payment_refunded'* if the payer refunded their payment.

suggested_post_message

Optional. Message containing the suggested post. Note that the *Message* object in this field will not contain the *reply_to_message* field even if it itself is a reply.

Type
telegram.Message

reason

Reason for the refund. Currently, one of *'post_deleted'* if the post was deleted within 24 hours of being posted or removed from scheduled messages without being posted, or *'payment_refunded'* if the payer refunded their payment.

Type
str

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

SwitchInlineQueryChosenChat

class telegram.**SwitchInlineQueryChosenChat**(*query=None*, *allow_user_chats=None*, *allow_bot_chats=None*, *allow_group_chats=None*, *allow_channel_chats=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents an inline button that switches the current user to inline mode in a chosen chat, with an optional default inline query.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *query*, *allow_user_chats*, *allow_bot_chats*, *allow_group_chats*, and *allow_channel_chats* are equal.

Available In

telegram.InlineKeyboardButton.switch_inline_query_chosen_chat

Added in version 20.3.

Caution

The PTB team has discovered that you must pass at least one of *allow_user_chats*, *allow_bot_chats*, *allow_group_chats*, or *allow_channel_chats* to Telegram. Otherwise, an error will be raised.

Parameters

- ***query*** (str, optional) – The default inline query to be inserted in the input field. If left empty, only the bot's username will be inserted.
- ***allow_user_chats*** (bool, optional) – Pass True, if private chats with users can be chosen.
- ***allow_bot_chats*** (bool, optional) – Pass True, if private chats with bots can be chosen.
- ***allow_group_chats*** (bool, optional) – Pass True, if group and supergroup chats can be chosen.
- ***allow_channel_chats*** (bool, optional) – Pass True, if channel chats can be chosen.

query

Optional. The default inline query to be inserted in the input field. If left empty, only the bot's username will be inserted.

Type
str

allow_user_chats

Optional. True, if private chats with users can be chosen.

Type
bool

allow_bot_chats

Optional. True, if private chats with bots can be chosen.

Type
bool

allow_group_chats

Optional. True, if group and supergroup chats can be chosen.

Type
bool

allow_channel_chats

Optional. True, if channel chats can be chosen.

Type
bool

TelegramObject

class telegram.**TelegramObject**(*, *api_kwargs=None*)

Bases: object

Base class for most Telegram objects.

Objects of this type are subscriptable with strings. See *__getitem__()* for more details. The pickle and deepcopy() behavior of objects of this type are defined by *__getstate__()*, *__setstate__()* and *__deepcopy__()*.

Tip

Objects of this type can be serialized via Python's pickle module and pickled objects from one version of PTB are usually loadable in future versions. However, we can not guarantee that this compatibility will always be provided. At least a manual one-time conversion of the data may be needed on major updates of the library.

Use In

telegram.Bot.do_api_request()

Changed in version 20.0:

- Removed argument and attribute bot for several subclasses. Use *set_bot()* and *get_bot()* instead.
- Removed the possibility to pass arbitrary keyword arguments for several subclasses.
- String representations objects of this type was overhauled. See *__repr__()* for details. As this class doesn't implement object.__str__(), the default implementation will be used, which is equivalent to *__repr__()*.
- Objects of this class (or subclasses) are now immutable. This means that you can't set or delete attributes anymore. Moreover, attributes that were formerly of type list are now of type tuple.

Parameters

api_kwargs (dict[str, any], optional) – Arbitrary keyword arguments. Can be used to store data for which there are no dedicated attributes. These arguments are also considered by *to_dict()* and *to_json()*, i.e. when passing objects to Telegram. Passing them to Telegram is however not guaranteed to work for all kinds of objects, e.g. this will fail for objects that can not directly be JSON serialized.

Added in version 20.0.

api_kwargs

Optional. Arbitrary keyword arguments. Used to store data for which there are no dedicated attributes. These arguments are also considered by *to_dict()* and *to_json()*, i.e. when passing objects to Telegram. Passing them to Telegram is however not guaranteed to work for all kinds of objects, e.g. this will fail for objects that can not directly be JSON serialized.

Added in version 20.0.

Type

types.MappingProxyType [str, any]

__deepcopy__(*memodict*)

Customizes how copy.deepcopy() processes objects of this type. The only difference to the default implementation is that the *telegram.Bot* instance set via *set_bot()* (if any) is not copied, but shared between the original and the copy, i.e.:

```
assert telegram_object.get_bot() is copy.deepcopy(telegram_object).get_bot()
```

Parameters
memodict (dict) – A dictionary that maps objects to their copies.

Returns
The copied object.

Return type
telegram.TelegramObject

__delattr__(*key*)

Overrides object.__delattr__() to prevent the deletion of attributes.

Raises
AttributeError –

__eq__(*other*)

Compares this object with *other* in terms of equality. If this object and *other* are *not* objects of the same class, this comparison will fall back to Python's default implementation of object.__eq__(). Otherwise, both objects may be compared in terms of equality, if the corresponding subclass of *TelegramObject* has defined a set of attributes to compare and the objects are considered to be equal, if all of these attributes are equal. If the subclass has not defined a set of attributes to compare, a warning will be issued.

> **Tip**
>
> If instances of a class in the *telegram* module are comparable in terms of equality, the documentation of the class will state the attributes that will be used for this comparison.

Parameters
other (object) – The object to compare with.

Returns
bool

__getitem__(*item*)

Objects of this type are subscriptable with strings, where `telegram_object["attribute_name"]` is equivalent to `telegram_object.attribute_name`.

> **Tip**
>
> This is useful for dynamic attribute lookup, i.e. `telegram_object[arg]` where the value of `arg` is determined at runtime. In all other cases, it's recommended to use the dot notation instead, i.e. `telegram_object.attribute_name`.

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.

Parameters
item (str) – The name of the attribute to look up.

Returns
object

Raises
KeyError – If the object does not have an attribute with the appropriate name.

__getstate__()

Overrides object.__getstate__() to customize the pickling process of objects of this type. The returned state does *not* contain the *telegram.Bot* instance set with *set_bot()* (if any), as it can't be pickled.

Returns
The state of the object.

Return type
state (dict[str, object])

__hash__()

Builds a hash value for this object such that the hash of two objects is equal if and only if the objects are equal in terms of *__eq__()*.

Returns
int

__repr__()

Gives a string representation of this object in the form ClassName(attr_1=value_1, attr_2=value_2, ...), where attributes are omitted if they have the value None or are empty instances of collections.abc.Sized (e.g. list, dict, set, str, etc.).

As this class doesn't implement object.__str__(), the default implementation will be used, which is equivalent to *__repr__()*.

Returns
str

__setattr__(*key*, *value*)

Overrides object.__setattr__() to prevent the overriding of attributes.

Raises
AttributeError –

__setstate__(*state*)

Overrides object.__setstate__() to customize the unpickling process of objects of this type. Modifies the object in-place.

If any data was stored in the *api_kwargs* of the pickled object, this method checks if the class now has dedicated attributes for those keys and moves the values from *api_kwargs* to the dedicated attributes. This can happen, if serialized data is loaded with a new version of this library, where the new version was updated to account for updates of the Telegram Bot API.

If on the contrary an attribute was removed from the class, the value is not discarded but made available via *api_kwargs*.

Parameters
state (dict) – The data to set as attributes of this object.

classmethod de_json(*data*, *bot=None*)

Converts JSON data to a Telegram object.

Parameters

- ***data*** (dict[str, ...]) – The JSON data.
- ***bot*** (*telegram.Bot*, optional) – The bot associated with this object. Defaults to None, in which case shortcut methods will not be available.

  Changed in version 21.4: *bot* is now optional and defaults to None

Returns
The Telegram object.

classmethod de_list(*data*, *bot=None*)

Converts a list of JSON objects to a tuple of Telegram objects.

Changed in version 20.0:

- Returns a tuple instead of a list.
- Filters out any `None` values.

Parameters

- ***data*** (list[dict[`str`, ...]]) – The JSON data.
- ***bot*** (*telegram.Bot*, optional) – The bot associated with these object. Defaults to `None`, in which case shortcut methods will not be available.

  Changed in version 21.4: *bot* is now optional and defaults to `None`

Returns

A tuple 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(*recursive=True*)

Gives representation of object as `dict`.

Changed in version 20.0:

- Now includes all entries of *api_kwargs*.
- Attributes whose values are empty sequences are no longer included.

Parameters

recursive (`bool`, optional) – If `True`, will convert any TelegramObjects (if found) in the attributes to a dictionary. Else, preserves it as an object itself. Defaults to `True`.

Added in version 20.0.

Returns

`dict`

to_json()

Gives a JSON representation of object.

Changed in version 20.0: Now includes all entries of *api_kwargs*.

Returns

str

TextQuote

class telegram.**TextQuote**(*text*, *position*, *entities=None*, *is_manual=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object contains information about the quoted part of a message that is replied to by the given message.

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

Available In

telegram.Message.quote

Added in version 20.8.

Parameters

- ***text*** (str) – Text of the quoted part of a message that is replied to by the given message.
- ***position*** (int) – Approximate quote position in the original message in UTF-16 code units as specified by the sender.
- ***entities*** (Sequence[*telegram.MessageEntity*], optional) – Special entities that appear in the quote. Currently, only bold, italic, underline, strikethrough, spoiler, custom_emoji, and date_time entities are kept in quotes.
- ***is_manual*** (bool, optional) – True, if the quote was chosen manually by the message sender. Otherwise, the quote was added automatically by the server.

text

Text of the quoted part of a message that is replied to by the given message.

Type

str

position

Approximate quote position in the original message in UTF-16 code units as specified by the sender.

Type

int

entities

Optional. Special entities that appear in the quote. Currently, only bold, italic, underline, strikethrough, spoiler, custom_emoji, and date_time entities are kept in quotes.

Type

tuple[*telegram.MessageEntity*]

is_manual

Optional. True, if the quote was chosen manually by the message sender. Otherwise, the quote was added automatically by the server.

Type

bool

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

UniqueGift

class telegram.**UniqueGift**(*gift_id*, *base_name*, *name*, *number*, *model*, *symbol*, *backdrop*, *publisher_chat=None*, *is_from_blockchain=None*, *is_premium=None*, *colors=None*, *is_burned=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object describes a unique gift that was upgraded from a regular gift.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal if their *base_name*, *name*, *number*, *model*, *symbol*, and *backdrop* are equal.

Available In

- *telegram.OwnedGiftUnique.gift*
- *telegram.UniqueGiftInfo.gift*

Added in version 22.1.

Changed in version 22.7: *gift_id* is now a positional argument.

Parameters

- ***gift_id*** (str) – Identifier of the regular gift from which the gift was upgraded.

  Added in version 22.6.

- ***base_name*** (str) – Human-readable name of the regular gift from which this unique gift was upgraded.

- ***name*** (str) – Unique name of the gift. This name can be used in `https://t.me/nft/...` links and story areas.

- ***number*** (int) – Unique number of the upgraded gift among gifts upgraded from the same regular gift.

- ***model*** (*UniqueGiftModel*) – Model of the gift.

- ***symbol*** (*UniqueGiftSymbol*) – Symbol of the gift.

- ***backdrop*** (*UniqueGiftBackdrop*) – Backdrop of the gift.

- ***publisher_chat*** (*telegram.Chat*, optional) – Information about the chat that published the gift.

  Added in version 22.4.

- ***is_premium*** (bool, optional) – True, if the original regular gift was exclusively purchaseable by Telegram Premium subscribers.

  Added in version 22.6.

- ***is_from_blockchain*** (bool, optional) – True, if the gift is assigned from the TON blockchain and can't be resold or transferred in Telegram.

  Added in version 22.6.

- ***colors*** (*telegram.UniqueGiftColors*, optional) – The color scheme that can be used by the gift's owner for the chat's name, replies to messages and link previews; for business account gifts and gifts that are currently on sale only.

  Added in version 22.6.

- ***is_burned*** (bool, optional) – True, if the gift was used to craft another gift and isn't available anymore.

  Added in version 22.7.

gift_id

Identifier of the regular gift from which the gift was upgraded.

Added in version 22.6.

Type
str

base_name

Human-readable name of the regular gift from which this unique gift was upgraded.

Type
str

name

Unique name of the gift. This name can be used in `https://t.me/nft/...` links and story areas.

Type
str

number

Unique number of the upgraded gift among gifts upgraded from the same regular gift.

Type
int

model

Model of the gift.

Type
telegram.UniqueGiftModel

symbol

Symbol of the gift.

Type
telegram.UniqueGiftSymbol

backdrop

Backdrop of the gift.

Type
telegram.UniqueGiftBackdrop

publisher_chat

Optional. Information about the chat that published the gift.

Added in version 22.4.

Type
telegram.Chat

is_premium

Optional. True, if the original regular gift was exclusively purchaseable by Telegram Premium subscribers.

Added in version 22.6.

Type
bool

is_from_blockchain

Optional. `True`, if the gift is assigned from the TON blockchain and can't be resold or transferred in Telegram.

Added in version 22.6.

Type
`bool`

colors

Optional. The color scheme that can be used by the gift's owner for the chat's name, replies to messages and link previews; for business account gifts and gifts that are currently on sale only.

Added in version 22.6.

Type
telegram.UniqueGiftColors

is_burned

Optional. `True`, if the gift was used to craft another gift and isn't available anymore.

Added in version 22.7.

Type
`bool`

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

UniqueGiftColors

class telegram.**UniqueGiftColors**(*model_custom_emoji_id*, *symbol_custom_emoji_id*, *light_theme_main_color*, *light_theme_other_colors*, *dark_theme_main_color*, *dark_theme_other_colors*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object contains information about the color scheme for a user's name, message replies and link previews based on a unique gift.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal if their *model_custom_emoji_id*, *symbol_custom_emoji_id*, *light_theme_main_color*, *light_theme_other_colors*, *dark_theme_main_color*, and *dark_theme_other_colors* are equal.

Available In

- *telegram.ChatFullInfo.unique_gift_colors*
- *telegram.UniqueGift.colors*

Added in version 22.6.

Parameters

- ***model_custom_emoji_id*** (`str`) – Custom emoji identifier of the unique gift's model.
- ***symbol_custom_emoji_id*** (`str`) – Custom emoji identifier of the unique gift's symbol.
- ***light_theme_main_color*** (`int`) – Main color used in light themes; RGB format.
- ***light_theme_other_colors*** (Sequence[`int`]) – List of 1-3 additional colors used in light themes; RGB format. Accepts any `collections.abc.Sequence` as input instead of just a list. The input is converted to a tuple.

- ***dark_theme_main_color*** (int) – Main color used in dark themes; RGB format.
- ***dark_theme_other_colors*** (Sequence[int]) – List of 1-3 additional colors used in dark themes; RGB format. Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.

model_custom_emoji_id

Custom emoji identifier of the unique gift's model.

Type
str

symbol_custom_emoji_id

Custom emoji identifier of the unique gift's symbol.

Type
str

light_theme_main_color

Main color used in light themes; RGB format.

Type
int

light_theme_other_colors

Tuple of 1-3 additional colors used in light themes; RGB format.

Type
Tuple[int]

dark_theme_main_color

Main color used in dark themes; RGB format.

Type
int

dark_theme_other_colors

Tuple of 1-3 additional colors used in dark themes; RGB format.

Type
Tuple[int]

UniqueGiftBackdrop

class telegram.**UniqueGiftBackdrop**(*name*, *colors*, *rarity_per_mille*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object describes the backdrop of a unique gift.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal if their *name*, *colors*, and *rarity_per_mille* are equal.

Available In

telegram.UniqueGift.backdrop

Added in version 22.1.

Parameters

- ***name*** (str) – Name of the backdrop.
- ***colors*** (*telegram.UniqueGiftBackdropColors*) – Colors of the backdrop.

- ***rarity_per_mille*** (int) – The number of unique gifts that receive this backdrop for every 1000 gifts upgraded.

name

Name of the backdrop.

Type
str

colors

Colors of the backdrop.

Type
telegram.UniqueGiftBackdropColors

rarity_per_mille

The number of unique gifts that receive this backdrop for every 1000 gifts upgraded.

Type
int

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

UniqueGiftBackdropColors

class telegram.**UniqueGiftBackdropColors**(*center_color*, *edge_color*, *symbol_color*, *text_color*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object describes the colors of the backdrop of a unique gift.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal if their *center_color*, *edge_color*, *symbol_color*, and *text_color* are equal.

Available In

telegram.UniqueGiftBackdrop.colors

Added in version 22.1.

Parameters

- ***center_color*** (int) – The color in the center of the backdrop in RGB format.
- ***edge_color*** (int) – The color on the edges of the backdrop in RGB format.
- ***symbol_color*** (int) – The color to be applied to the symbol in RGB format.
- ***text_color*** (int) – The color for the text on the backdrop in RGB format.

center_color

The color in the center of the backdrop in RGB format.

Type
int

edge_color

The color on the edges of the backdrop in RGB format.

Type
int

symbol_color

The color to be applied to the symbol in RGB format.

Type
int

text_color

The color for the text on the backdrop in RGB format.

Type
int

UniqueGiftInfo

class telegram.**UniqueGiftInfo**(*gift*, *origin*, *owned_gift_id=None*, *transfer_star_count=None*, *next_transfer_date=None*, *last_resale_currency=None*, *last_resale_amount=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a service message about a unique gift that was sent or received.

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

Available In

telegram.Message.unique_gift

Added in version 22.1.

Removed in version 22.7: Removed argument and attribute `last_resale_star_count` deprecated since Bot API 9.3.

Parameters

- ***gift*** (*UniqueGift*) – Information about the gift.
- ***origin*** (`str`) – Origin of the gift. Currently, either *UPGRADE* for gifts upgraded from regular gifts, *TRANSFER* for gifts transferred from other users or channels, *RESALE* for gifts bought from other users, *GIFTED_UPGRADE* for upgrades purchased after the gift was sent, or *OFFER* for gifts bought or sold through gift purchase offers

  Changed in version 22.3: The *RESALE* origin was added.

  Changed in version 22.6: Bot API 9.3 added the *GIFTED_UPGRADE* and *OFFER* origins.
- ***owned_gift_id*** (`str`, optional) – bot; only present for gifts received on behalf of business accounts.
- ***transfer_star_count*** (`int`, optional) – Number of Telegram Stars that must be paid to transfer the gift; omitted if the bot cannot transfer the gift.
- ***last_resale_currency*** (`str`, optional) – For gifts bought from other users, the currency in which the payment for the gift was done. Currently, one of `XTR` for Telegram Stars or `TON` for toncoins.

  Added in version 22.6.
- ***last_resale_amount*** (`int`, optional) – For gifts bought from other users, the price paid for the gift in either Telegram Stars or nanotoncoins.

  Added in version 22.6.
- ***next_transfer_date*** (`datetime.datetime`, optional) – Date when the gift can be transferred. If it's in the past, then the gift can be transferred now. The default timezone

of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Added in version 22.3.

gift

Information about the gift.

Type
UniqueGift

origin

Origin of the gift. Currently, either *UPGRADE* for gifts upgraded from regular gifts, *TRANSFER* for gifts transferred from other users or channels, *RESALE* for gifts bought from other users, *GIFTED_UPGRADE* for upgrades purchased after the gift was sent, or *OFFER* for gifts bought or sold through gift purchase offers

Changed in version 22.3: The *RESALE* origin was added.

Changed in version 22.6: Bot API 9.3 added the *GIFTED_UPGRADE* and *OFFER* origins.

Type
str

owned_gift_id

bot; only present for gifts received on behalf of business accounts.

Type
str

transfer_star_count

Optional. Number of Telegram Stars that must be paid to transfer the gift; omitted if the bot cannot transfer the gift.

Type
int

last_resale_currency

Optional. For gifts bought from other users, the currency in which the payment for the gift was done. Currently, one of `XTR` for Telegram Stars or `TON` for toncoins.

Added in version 22.6.

Type
str

last_resale_amount

Optional. For gifts bought from other users, the price paid for the gift in either Telegram Stars or nanotoncoins.

Added in version 22.6.

Type
int

next_transfer_date

Optional. Date when the gift can be transferred. If it's in the past, then the gift can be transferred now. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Added in version 22.3.

Type
datetime.datetime

GIFTED_UPGRADE = 'gifted_upgrade'

telegram.constants.UniqueGiftInfoOrigin.GIFTED_UPGRADE

Added in version 22.6.

OFFER = 'OFFER'

telegram.constants.UniqueGiftInfoOrigin.OFFER

Added in version 22.6.

RESALE = 'resale'

telegram.constants.UniqueGiftInfoOrigin.RESALE

Added in version 22.3.

TRANSFER = 'transfer'

telegram.constants.UniqueGiftInfoOrigin.TRANSFER

UPGRADE = 'upgrade'

telegram.constants.UniqueGiftInfoOrigin.UPGRADE

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

UniqueGiftModel

class telegram.**UniqueGiftModel**(*name*, *sticker*, *rarity_per_mille*, *rarity=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object describes the model of a unique gift.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal if their *name*, *sticker* and *rarity_per_mille* are equal.

Available In

telegram.UniqueGift.model

Added in version 22.1.

Parameters

- ***name*** (str) – Name of the model.
- ***sticker*** (*telegram.Sticker*) – The sticker that represents the unique gift.
- ***rarity_per_mille*** (int) – The number of unique gifts that receive this model for every 1000 gifts upgraded. Always 0 for crafted gifts.
- ***rarity*** (str, optional) – Rarity of the model if it is a crafted model. Currently, can be *'uncommon'*, *'rare'*, *'epic'*, or *'legendary'*.

  Added in version 22.7.

name

Name of the model.

Type

str

sticker

The sticker that represents the unique gift.

Type

telegram.Sticker

rarity_per_mille

The number of unique gifts that receive this model for every `1000` gifts upgraded. Always `0` for crafted gifts.

Type
int

rarity

Optional. Rarity of the model if it is a crafted model. Currently, can be *'uncommon'*, *'rare'*, *'epic'*, or *'legendary'*.

Added in version 22.7.

Type
str

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

UniqueGiftSymbol

class telegram.**UniqueGiftSymbol**(*name*, *sticker*, *rarity_per_mille*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object describes the symbol shown on the pattern of a unique gift.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal if their *name*, *sticker* and *rarity_per_mille* are equal.

Available In

telegram.UniqueGift.symbol

Added in version 22.1.

Parameters

- ***name*** (str) – Name of the symbol.
- ***sticker*** (*telegram.Sticker*) – The sticker that represents the unique gift.
- ***rarity_per_mille*** (int) – The number of unique gifts that receive this model for every `1000` gifts upgraded.

name

Name of the symbol.

Type
str

sticker

The sticker that represents the unique gift.

Type
telegram.Sticker

rarity_per_mille

The number of unique gifts that receive this model for every `1000` gifts upgraded.

Type
int

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

Update

class telegram.**Update**(*update_id*, *message=None*, *edited_message=None*, *channel_post=None*, *edited_channel_post=None*, *inline_query=None*, *chosen_inline_result=None*, *callback_query=None*, *shipping_query=None*, *pre_checkout_query=None*, *poll=None*, *poll_answer=None*, *my_chat_member=None*, *chat_member=None*, *chat_join_request=None*, *chat_boost=None*, *removed_chat_boost=None*, *message_reaction=None*, *message_reaction_count=None*, *business_connection=None*, *business_message=None*, *edited_business_message=None*, *deleted_business_messages=None*, *purchased_paid_media=None*, *managed_bot=None*, *guest_message=None*, ***, *api_kwargs=None*)

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.

See also

Your First Bot

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

Returned In

telegram.Bot.get_updates()

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. This update may at times be triggered by changes to message fields that are either unavailable or not actively used by your bot.

- ***channel_post*** (*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. This update may at times be triggered by changes to message fields that are either unavailable or not actively used by your bot.
- ***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 manually 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.

  Added 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()*).

  Added 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.

  Added in version 13.8.
- ***chat_boost*** (*telegram.ChatBoostUpdated*, optional) – A chat boost was added or changed. The bot must be an administrator in the chat to receive these updates.

  Added in version 20.8.
- ***removed_chat_boost*** (*telegram.ChatBoostRemoved*, optional) – A boost was removed from a chat. The bot must be an administrator in the chat to receive these updates.

  Added in version 20.8.
- ***message_reaction*** (*telegram.MessageReactionUpdated*, optional) – A reaction to a message was changed by a user. The bot must be an administrator in the chat and must explicitly specify *MESSAGE_REACTION* in the list of *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()*). The update isn't received for reactions set by bots.

  Added in version 20.8.

- ***message_reaction_count*** (*telegram.MessageReactionCountUpdated*, optional) – Reactions to a message with anonymous reactions were changed. The bot must be an administrator in the chat and must explicitly specify *MESSAGE_REACTION_COUNT* in the list of *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()*). The updates are grouped and can be sent with delay up to a few minutes.

  Added in version 20.8.

- ***business_connection*** (*telegram.BusinessConnection*, optional) – The bot was connected to or disconnected from a business account, or a user edited an existing connection with the bot.

  Added in version 21.1.

- ***business_message*** (*telegram.Message*, optional) – New message from a connected business account.

  Added in version 21.1.

- ***edited_business_message*** (*telegram.Message*, optional) – New version of a message from a connected business account.

  Added in version 21.1.

- ***deleted_business_messages*** (*telegram.BusinessMessagesDeleted*, optional) – Messages were deleted from a connected business account.

  Added in version 21.1.

- ***purchased_paid_media*** (*telegram.PaidMediaPurchased*, optional) – A user purchased paid media with a non-empty payload sent by the bot in a non-channel chat.

  Added in version 21.6.

- ***managed_bot*** (*telegram.ManagedBotUpdated*, optional) – A new bot was created to be managed by the bot, or token or owner of a managed bot was changed.

  Added in version 22.8.

- ***guest_message*** (*telegram.Message*, optional) – New guest message. The bot can use the field *telegram.Message.guest_query_id* and the method *telegram.Bot.answer_guest_query()* to send a message in response.

  Added in version 22.8.

update_id

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.

Type
int

message

Optional. New incoming message of any kind - text, photo, sticker, etc.

Type
telegram.Message

edited_message

Optional. New version of a message that is known to the bot and was edited. This update may at times be triggered by changes to message fields that are either unavailable or not actively used by your bot.

Type
telegram.Message

channel_post

Optional. New incoming channel post of any kind - text, photo, sticker, etc.

Type
telegram.Message

edited_channel_post

Optional. New version of a channel post that is known to the bot and was edited. This update may at times be triggered by changes to message fields that are either unavailable or not actively used by your bot.

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 and sent to their chat partner.

Type
telegram.ChosenInlineResult

callback_query

Optional. New incoming callback query.

Examples

Arbitrary Callback Data Bot

Type
telegram.CallbackQuery

shipping_query

Optional. New incoming shipping query. Only for invoices with flexible price.

Type
telegram.ShippingQuery

pre_checkout_query

Optional. New incoming pre-checkout query. Contains full information about checkout.

Type
telegram.PreCheckoutQuery

poll

Optional. New poll state. Bots receive only updates about manually 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.

Added 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()*).

Added 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.

Added in version 13.8.

Type
telegram.ChatJoinRequest

chat_boost

Optional. A chat boost was added or changed. The bot must be an administrator in the chat to receive these updates.

Added in version 20.8.

Type
telegram.ChatBoostUpdated

removed_chat_boost

Optional. A boost was removed from a chat. The bot must be an administrator in the chat to receive these updates.

Added in version 20.8.

Type
telegram.ChatBoostRemoved

message_reaction

Optional. A reaction to a message was changed by a user. The bot must be an administrator in the chat and must explicitly specify *MESSAGE_REACTION* in the list of *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()*). The update isn't received for reactions set by bots.

Added in version 20.8.

Type
telegram.MessageReactionUpdated

message_reaction_count

Optional. Reactions to a message with anonymous reactions were changed. The bot must be an administrator in the chat and must explicitly specify *MESSAGE_REACTION_COUNT* in the list of *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()*). The updates are grouped and can be sent with delay up to a few minutes.

Added in version 20.8.

Type
telegram.MessageReactionCountUpdated

business_connection

Optional. The bot was connected to or disconnected from a business account, or a user edited an existing connection with the bot.

Added in version 21.1.

Type
telegram.BusinessConnection

business_message

Optional. New message from a connected business account.

Added in version 21.1.

Type
telegram.Message

edited_business_message

Optional. New version of a message from a connected business account.

Added in version 21.1.

Type
telegram.Message

deleted_business_messages

Optional. Messages were deleted from a connected business account.

Added in version 21.1.

Type
telegram.BusinessMessagesDeleted

purchased_paid_media

Optional. A user purchased paid media with a non-empty payload sent by the bot in a non-channel chat.

Added in version 21.6.

Type
telegram.PaidMediaPurchased

managed_bot

Optional. A new bot was created to be managed by the bot, or token or owner of a managed bot was changed.

Added in version 22.8.

Type
telegram.ManagedBotUpdated

guest_message

Optional. New guest message. The bot can use the field *telegram.Message.guest_query_id* and the method *telegram.Bot.answerGuestQuery()* to send a message in response.

Added in version 22.8.

Type
telegram.Message

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>, <UpdateType.CHAT_BOOST>, <UpdateType.REMOVED_CHAT_BOOST>, <UpdateType.MESSAGE_REACTION>, <UpdateType.MESSAGE_REACTION_COUNT>, <UpdateType.BUSINESS_CONNECTION>, <UpdateType.BUSINESS_MESSAGE>, <UpdateType.EDITED_BUSINESS_MESSAGE>, <UpdateType.DELETED_BUSINESS_MESSAGES>, <UpdateType.PURCHASED_PAID_MEDIA>, <UpdateType.MANAGED_BOT>, <UpdateType.GUEST_MESSAGE>]

A list of all available update types.

Added in version 13.5.

Type
list[str]

BUSINESS_CONNECTION = 'business_connection'

telegram.constants.UpdateType.BUSINESS_CONNECTION

Added in version 21.1.

BUSINESS_MESSAGE = 'business_message'

telegram.constants.UpdateType.BUSINESS_MESSAGE

Added in version 21.1.

CALLBACK_QUERY = 'callback_query'

telegram.constants.UpdateType.CALLBACK_QUERY

Added in version 13.5.

CHANNEL_POST = 'channel_post'

telegram.constants.UpdateType.CHANNEL_POST

Added in version 13.5.

CHAT_BOOST = 'chat_boost'

telegram.constants.UpdateType.CHAT_BOOST

Added in version 20.8.

CHAT_JOIN_REQUEST = 'chat_join_request'

telegram.constants.UpdateType.CHAT_JOIN_REQUEST

Added in version 13.8.

CHAT_MEMBER = 'chat_member'

telegram.constants.UpdateType.CHAT_MEMBER

Added in version 13.5.

CHOSEN_INLINE_RESULT = 'chosen_inline_result'

telegram.constants.UpdateType.CHOSEN_INLINE_RESULT

Added in version 13.5.

DELETED_BUSINESS_MESSAGES = 'deleted_business_messages'

telegram.constants.UpdateType.DELETED_BUSINESS_MESSAGES

Added in version 21.1.

EDITED_BUSINESS_MESSAGE = 'edited_business_message'

telegram.constants.UpdateType.EDITED_BUSINESS_MESSAGE

Added in version 21.1.

EDITED_CHANNEL_POST = 'edited_channel_post'

telegram.constants.UpdateType.EDITED_CHANNEL_POST

Added in version 13.5.

EDITED_MESSAGE = 'edited_message'

telegram.constants.UpdateType.EDITED_MESSAGE

Added in version 13.5.

GUEST_MESSAGE = 'guest_message'

telegram.constants.UpdateType.GUEST_MESSAGE

Added in version 22.8.

INLINE_QUERY = 'inline_query'

telegram.constants.UpdateType.INLINE_QUERY

Added in version 13.5.

MANAGED_BOT = 'managed_bot'

telegram.constants.UpdateType.MANAGED_BOT

Added in version 22.8.

MESSAGE = 'message'

telegram.constants.UpdateType.MESSAGE

Added in version 13.5.

MESSAGE_REACTION = 'message_reaction'

telegram.constants.UpdateType.MESSAGE_REACTION

Added in version 20.8.

MESSAGE_REACTION_COUNT = 'message_reaction_count'

telegram.constants.UpdateType.MESSAGE_REACTION_COUNT

Added in version 20.8.

MY_CHAT_MEMBER = 'my_chat_member'

telegram.constants.UpdateType.MY_CHAT_MEMBER

Added in version 13.5.

POLL = 'poll'

telegram.constants.UpdateType.POLL

Added in version 13.5.

POLL_ANSWER = 'poll_answer'

telegram.constants.UpdateType.POLL_ANSWER

Added in version 13.5.

PRE_CHECKOUT_QUERY = 'pre_checkout_query'

telegram.constants.UpdateType.PRE_CHECKOUT_QUERY

Added in version 13.5.

PURCHASED_PAID_MEDIA = 'purchased_paid_media'

telegram.constants.UpdateType.PURCHASED_PAID_MEDIA

Added in version 21.6.

REMOVED_CHAT_BOOST = 'removed_chat_boost'

telegram.constants.UpdateType.REMOVED_CHAT_BOOST

Added in version 20.8.

SHIPPING_QUERY = 'shipping_query'

telegram.constants.UpdateType.SHIPPING_QUERY

Added in version 13.5.

classmethod de_json(*data*, *bot=None*)

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*, *poll_answer*, *business_connection*, *purchased_paid_media*, or *managed_bot* is present.

Changed in version 21.1: This property now also considers *business_message*, *edited_business_message*, and *deleted_business_messages*.

Changed in version 22.8: This property now also considers *guest_message*.

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.

Changed in version 21.1: This property now also considers *business_message*, and *edited_business_message*.

Changed in version 22.8: This property now also considers *guest_message*.

Tip

This property will only ever return objects of type *telegram.Message* or None, never *telegram.MaybeInaccessibleMessage* or *telegram.InaccessibleMessage*. Currently, this is only relevant for *callback_query*, as *telegram.CallbackQuery.message* is the only attribute considered by this property that can be an object of these types.

Type
telegram.Message

property effective_sender

The user or chat that sent this update, no matter what kind of update this is.

Note

- Depending on the type of update and the user's 'Remain anonymous' setting, this could either be *telegram.User*, *telegram.Chat* or None.

If no user whatsoever is associated with this update, this gives None. This is the case if any of

- *poll*
- *chat_boost*
- *removed_chat_boost*
- *message_reaction_count*
- *deleted_business_messages*

is present.

Changed in version 22.8: This property now also considers *guest_message*.

Example

- If *message* is present, this will give either *telegram.Message.from_user* or *telegram.Message.sender_chat*.
- If *poll_answer* is present, this will give either *telegram.PollAnswer.user* or *telegram.PollAnswer.voter_chat*.
- If *channel_post* is present, this will give *telegram.Message.sender_chat*.

Added in version 21.1.

Type
telegram.User or *telegram.Chat*

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 any of

- *poll*
- *chat_boost*
- *removed_chat_boost*
- *message_reaction_count*
- *deleted_business_messages*

is present.

Changed in version 21.1: This property now also considers *business_connection*, *business_message* and *edited_business_message*.

Changed in version 21.6: This property now also considers *purchased_paid_media*.

Changed in version 22.8: This property now also considers *managed_bot*, *guest_message*, *channel_post*, and *edited_channel_post*.

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

User

class telegram.**User**(*id*, *first_name*, *is_bot*, *last_name=None*, *username=None*, *language_code=None*, *can_join_groups=None*, *can_read_all_group_messages=None*, *supports_inline_queries=None*, *is_premium=None*, *added_to_attachment_menu=None*, *can_connect_to_business=None*, *has_main_web_app=None*, *has_topics_enabled=None*, *allows_users_to_create_topics=None*, *can_manage_bots=None*, *supports_guest_queries=None*, ***, *api_kwargs=None*)

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.

Available In

- *telegram.AffiliateInfo.affiliate_user*
- *telegram.Bot.bot*
- *telegram.BotAccessSettings.added_users*
- *telegram.BusinessConnection.user*
- *telegram.CallbackQuery.from_user*
- *telegram.ChatBoostSourceGiftCode.user*
- *telegram.ChatBoostSourceGiveaway.user*
- *telegram.ChatBoostSourcePremium.user*
- *telegram.ChatInviteLink.creator*
- *telegram.ChatJoinRequest.from_user*
- *telegram.ChatMember.user*
- *telegram.ChatMemberAdministrator.user*
- *telegram.ChatMemberBanned.user*
- *telegram.ChatMemberLeft.user*

- *telegram.ChatMemberMember.user*
- *telegram.ChatMemberOwner.user*
- *telegram.ChatMemberRestricted.user*
- *telegram.ChatMemberUpdated.from_user*
- *telegram.ChatOwnerChanged.new_owner*
- *telegram.ChatOwnerLeft.new_owner*
- *telegram.ChecklistTask.completed_by_user*
- *telegram.ChosenInlineResult.from_user*
- *telegram.DirectMessagesTopic.user*
- *telegram.GameHighScore.user*
- *telegram.GiveawayWinners.winners*
- *telegram.InlineQuery.from_user*
- *telegram.ManagedBotCreated.bot*
- *telegram.ManagedBotUpdated.bot*
- *telegram.ManagedBotUpdated.user*
- *telegram.Message.from_user*
- *telegram.Message.guest_bot_caller_user*
- *telegram.Message.left_chat_member*
- *telegram.Message.new_chat_members*
- *telegram.Message.sender_business_bot*
- *telegram.Message.via_bot*
- *telegram.MessageEntity.user*
- *telegram.MessageOriginUser.sender_user*
- *telegram.MessageReactionUpdated.user*
- *telegram.OwnedGiftRegular.sender_user*
- *telegram.OwnedGiftUnique.sender_user*
- *telegram.PaidMediaPurchased.from_user*
- *telegram.PollAnswer.user*
- *telegram.PollOption.added_by_user*
- *telegram.PreCheckoutQuery.from_user*
- *telegram.ProximityAlertTriggered.traveler*
- *telegram.ProximityAlertTriggered.watcher*
- *telegram.ShippingQuery.from_user*
- *telegram.TransactionPartnerAffiliateProgram.sponsor_user*
- *telegram.TransactionPartnerUser.user*
- *telegram.Update.effective_sender*
- *telegram.Update.effective_user*
- *telegram.VideoChatParticipantsInvited.users*

Returned In

telegram.Bot.get_me()

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 bot's first name.
- ***last_name*** (`str`, optional) – User's or bot's last name.
- ***username*** (`str`, optional) – User's or bot's 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()*.
- ***can_read_all_group_messages*** (`str`, optional) – `True`, if privacy mode is disabled for the bot. Returned only in *telegram.Bot.get_me()*.
- ***supports_inline_queries*** (`str`, optional) – `True`, if the bot supports inline queries. Returned only in *telegram.Bot.get_me()*.
- ***is_premium*** (`bool`, optional) – `True`, if this user is a Telegram Premium user.

  Added in version 20.0.
- ***added_to_attachment_menu*** (`bool`, optional) – `True`, if this user added the bot to the attachment menu.

  Added in version 20.0.
- ***can_connect_to_business*** (`bool`, optional) – `True`, if the bot can be connected to a user account to manage it. Returned only in *telegram.Bot.get_me()*.

  Added in version 21.1.
- ***has_main_web_app*** (`bool`, optional) – `True`, if the bot has the main Web App. Returned only in *telegram.Bot.get_me()*.

  Added in version 21.5.
- ***has_topics_enabled*** (`bool`, optional) – `True`, if the bot has forum topic mode enabled in private chats. Returned only in *telegram.Bot.get_me()*.

  Added in version 22.6.
- ***allows_users_to_create_topics*** (`bool`, optional) – `True`, if the bot allows users to create and delete topics in private chats. Returned only in *telegram.Bot.get_me()*.

  Added in version 22.7.
- ***can_manage_bots*** (`bool`, optional) – `True`, if other bots can be created to be controlled by the bot. Returned only in *telegram.Bot.get_me()*.

  Added in version 22.8.
- ***supports_guest_queries*** (`bool`, optional) – `True`, if the bot supports guest queries from chats it is not a member of. Returned only in *telegram.Bot.get_me()*.

  Added in version 22.8.

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

is_premium

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

Added in version 20.0.

Type
bool

added_to_attachment_menu

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

Added in version 20.0.

Type
bool

can_connect_to_business

Optional. True, if the bot can be connected to a Telegram Business account to receive its messages. Returned only in *telegram.Bot.get_me()*.

Added in version 21.1.

Type
bool

has_main_web_app

Returned only in *telegram.Bot.get_me()*.

Added in version 21.5.

Type
bool

has_topics_enabled

Optional. True, if the bot has forum topic mode enabled in private chats. Returned only in *telegram.Bot.get_me()*.

Added in version 22.6.

Type
bool

allows_users_to_create_topics

Optional. True, if the bot allows users to create and delete topics in private chats. Returned only in *telegram.Bot.get_me()*.

Added in version 22.7.

Type
bool

can_manage_bots

Optional. True, if other bots can be created to be controlled by the bot. Returned only in *telegram.Bot.get_me()*.

Added in version 22.8.

Type
bool

supports_guest_queries

Optional. True, if the bot supports guest queries from chats it is not a member of. Returned only in *telegram.Bot.get_me()*.

Added in version 22.8.

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()*.

> **Note**
>
> This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Added 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_markup=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *show_caption_above_media=None*, *allow_paid_broadcast=None*, *video_start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_effect_id=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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()*.

> **Note**
>
> This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

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

Return type
telegram.MessageId

async copy_messages(*chat_id*, *message_ids*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *remove_caption=None*, *direct_messages_topic_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.copy_messages(from_chat_id=update.effective_user.id, *argss,␣
↪**kwargs)
```

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

> **See also**
>
> *copy_message()*, *send_copy()*, *send_copies()*.

Added in version 20.8.

Returns
On success, a tuple of *MessageId* of the sent messages is returned.

Return type
tuple[*telegram.MessageId*]

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()*.

> **Note**
>
> This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Added in version 13.8.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.delete_all_message_reactions(
    user_id=update.effective_user.id,
    *args,
    **kwargs
)
```

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

Added in version 22.8.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.delete_message(update.effective_user.id, *argss, **kwargs)
```

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

Added in version 20.8.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.delete_messages(update.effective_user.id, *argss, **kwargs)
```

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

Added in version 20.8.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.delete_message_reaction(
    user_id=update.effective_user.id,
    *args,
    **kwargs
)
```

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

Added in version 22.8.

Returns
On success, `True` is returned.

Return type
`bool`

async forward_from(*from_chat_id*, *message_id*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *video_start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_effect_id=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_user.id, *argss, **kwargs)
```

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

> **See also**
>
> *forward_to()*, *forward_messages_from()*, *forward_messages_to()*

Added in version 20.0.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async forward_messages_from(*from_chat_id*, *message_ids*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *direct_messages_topic_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.forward_messages(chat_id=update.effective_user.id, *argss,␣
↪**kwargs)
```

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

> **See also**
>
> *forward_to()*, *forward_from()*, *forward_messages_to()*.

Added in version 20.8.

Returns
On success, a tuple of *MessageId* of sent messages is returned.

Return type
tuple[*telegram.MessageId*]

async forward_messages_to(*chat_id*, *message_ids*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *direct_messages_topic_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.forward_messages(from_chat_id=update.effective_user.id, *argss,␣
↪**kwargs)
```

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

> **See also**
>
> *forward_from()*, *forward_to()*, *forward_messages_from()*.

Added in version 20.8.

Returns
On success, a tuple of *MessageId* of sent messages is returned.

Return type
tuple[*telegram.MessageId*]

async forward_to(*chat_id*, *message_id*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *video_start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_effect_id=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_user.id, *argss,␣
↪**kwargs)
```

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

See also

forward_from(), *forward_messages_from()*, *forward_messages_to()*

Added in version 20.0.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

property full_name

Convenience property. The user's *first_name*, followed by (if available) *last_name*.

Type
str

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

Shortcut for:

```
await bot.get_user_chat_boosts(user_id=update.effective_user.id, *args,␣
↪**kwargs)
```

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

Added in version 20.8.

Returns
On success, returns the boosts applied by the user.

Return type
telegram.UserChatBoosts

async get_gifts(*exclude_unlimited=None*, *exclude_limited_upgradable=None*, *exclude_limited_non_upgradable=None*, *exclude_from_blockchain=None*, *exclude_unique=None*, *sort_by_price=None*, *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_gifts(user_id=update.effective_user.id)
```

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

Added in version 22.6.

Returns
On success, returns the gifts owned by the user.

Return type
telegram.OwnedGifts

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

Shortcut for:

```
await bot.get_managed_bot_access_settings(
   user_id=update.effective_user.id,
   *args, **kwargs
)
```

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

Added in version 22.8.

Returns

On success, returns the access settings of the bot managed by the user.

Return type
telegram.BotAccessSettings

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()

Note

This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Added in version 20.0.

Returns
On success, the current menu button is returned.

Return type
telegram.MenuButton

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

Shortcut for:

```
await bot.get_user_personal_chat_messages(
    user_id=update.effective_user.id,
    *args, **kwargs
)
```

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

Added in version 22.8.

Returns

On success, a tuple of messages from the personal channel chat is returned.

Return type

tuple[*telegram.Message*]

async get_profile_audios(*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_audios(update.effective_user.id, *args, **kwargs)
```

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

Added in version 22.7.

Returns

telegram.UserProfileAudios

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()*.

Returns

telegram.UserProfilePhotos

async gift_premium_subscription(*month_count*, *star_count*, *text=None*, *text_parse_mode=None*, *text_entities=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.gift_premium_subscription(user_id=update.effective_user.id, *args,␣
↪**kwargs)
```

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

Added in version 22.1.

Returns

On success, `True` is returned.

Return type

`bool`

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}
↪")
```

Added 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*, *business_connection_id=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()*.

Note

This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns
On success, True is returned.

Return type
bool

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

Shortcut for:

```
await bot.refund_star_payment(user_id=update.effective_user.id, *args,␣
↪**kwargs)
```

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

Added in version 21.3.

Returns
On success, True is returned.

Return type
bool

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

Shortcut for:

```
await bot.remove_user_verification(user_id=update.effective_user.id, *args,␣
↪**kwargs)
```

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

Added in version 21.10.

Returns
On success, True is returned.

Return type
bool

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

Shortcut for:

```
await bot.replace_managed_bot_token(user_id=update.effective_user.id, *args,␣
↪**kwargs)
```

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

Added in version 22.8.

Returns
On success, `str` is returned.

Return type
`str`

async repost_story(*business_connection_id*, *from_story_id*, *active_period*, *post_to_chat_page=None*, *protect_content=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.repost_story(
    from_chat_id=update.effective_user.id,
    *args, **kwargs
)
```

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

Added in version 22.6.

Returns
On success, *Story* is returned.

Return type
Story

async send_action(*action*, *message_thread_id=None*, *business_connection_id=None*, *, *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*, *caption=None*, *parse_mode=None*, *disable_notification=None*, *reply_markup=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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()*.

> **Note**
>
> This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

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_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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()*.

> **Note**
>
> This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_chat_action(*action*, *message_thread_id=None*, *business_connection_id=None*, ***, *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()*.

> **Note**
>
> This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns
On success.

Return type
True

async **send_contact**(*phone_number=None*, *first_name=None*, *last_name=None*, *disable_notification=None*, *reply_markup=None*, *vcard=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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()*.

Note

This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async **send_copies**(*from_chat_id*, *message_ids*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *remove_caption=None*, *direct_messages_topic_id=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.copy_messages(chat_id=update.effective_user.id, *argss, **kwargs)
```

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

See also

copy_message(), *send_copy()*, *copy_messages()*.

Added in version 20.8.

Returns
On success, a tuple of *MessageId* of the sent messages is returned.

Return type
tuple[*telegram.MessageId*]

async send_copy(*from_chat_id*, *message_id*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *disable_notification=None*, *reply_markup=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *show_caption_above_media=None*, *allow_paid_broadcast=None*, *video_start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, *message_effect_id=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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()*.

Note

This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns

On success, instance representing the message posted.

Return type

telegram.Message

async send_dice(*disable_notification=None*, *reply_markup=None*, *emoji=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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()*.

Note

This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns

On success, instance representing the message posted.

Return type

telegram.Message

async send_document(*document*, *caption=None*, *disable_notification=None*, *reply_markup=None*, *parse_mode=None*, *disable_content_type_detection=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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()*.

Note

This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns

On success, instance representing the message posted.

Return type

telegram.Message

async send_game(*game_short_name*, *disable_notification=None*, *reply_markup=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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()*.

Note

This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns

On success, instance representing the message posted.

Return type

telegram.Message

async send_gift(*gift_id*, *text=None*, *text_parse_mode=None*, *text_entities=None*, *pay_for_upgrade=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_gift(user_id=update.effective_user.id, *args, **kwargs )
```

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

Added in version 21.8.

Returns
On success, `True` is returned.

Return type
`bool`

async send_invoice(*title*, *description*, *payload*, *currency*, *prices*, *provider_token=None*, *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_markup=None*, *provider_data=None*, *send_phone_number_to_provider=None*, *send_email_to_provider=None*, *max_tip_amount=None*, *suggested_tip_amounts=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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.

Note

This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

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_live_photo(*live_photo*, *photo*, *business_connection_id=None*, *message_thread_id=None*, *direct_messages_topic_id=None*, *caption=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, *has_spoiler=None*, *disable_notification=None*, *protect_content=None*, *allow_paid_broadcast=None*, *message_effect_id=None*, *suggested_post_parameters=None*, *reply_parameters=None*, *reply_markup=None*, ***, *allow_sending_without_reply=None*, *reply_to_message_id=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_live_photo(update.effective_user.id, *args, **kwargs)
```

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

Added in version 22.8.

Note

This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_location(*latitude=None*, *longitude=None*, *disable_notification=None*, *reply_markup=None*, *live_period=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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()*.

Note

This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_media_group(*media*, *disable_notification=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*, *caption=None*, *parse_mode=None*, *caption_entities=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()*.

Note

This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns

] On success, a tuple of *Message* instances that were sent is returned.

Return type

tuple[*telegram.Message*

async send_message(*text*, *parse_mode=None*, *disable_notification=None*, *reply_markup=None*, *entities=None*, *protect_content=None*, *message_thread_id=None*, *link_preview_options=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *disable_web_page_preview=None*, *allow_sending_without_reply=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()*.

Note

This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns

On success, instance representing the message posted.

Return type

telegram.Message

async send_message_draft(*draft_id*, *text=None*, *message_thread_id=None*, *parse_mode=None*, *entities=None*, ***, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.send_message_draft(update.effective_user.id, *args, **kwargs)
```

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

> **Note**
>
> This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Added in version 22.6.

Changed in version 22.8: Bot API 10.0 makes the `text` argument optional.

Returns
On success, `True` is returned.

Return type
`bool`

async send_photo(*photo*, *caption=None*, *disable_notification=None*, *reply_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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()*.

> **Note**
>
> This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

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_markup=None*, *explanation=None*, *explanation_parse_mode=None*, *open_period=None*, *close_date=None*, *explanation_entities=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *question_parse_mode=None*, *question_entities=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *shuffle_options=None*, *allows_revoting=None*, *correct_option_ids=None*, *allow_adding_options=None*, *hide_results_until_closes=None*, *description=None*, *description_parse_mode=None*, *description_entities=None*, *members_only=None*, *country_codes=None*, *explanation_media=None*, *media=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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()*.

Note

This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_sticker(*sticker*, *disable_notification=None*, *reply_markup=None*, *protect_content=None*, *message_thread_id=None*, *emoji=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *read_timeout=None*, *write_timeout=None*, *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()*.

Note

This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

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_markup=None*, *foursquare_type=None*, *google_place_id=None*, *google_place_type=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=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()*.

> **Note**
>
> This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_video(*video*, *duration=None*, *caption=None*, *disable_notification=None*, *reply_markup=None*, *width=None*, *height=None*, *parse_mode=None*, *supports_streaming=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *show_caption_above_media=None*, *cover=None*, *start_timestamp=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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()*.

> **Note**
>
> This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

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_markup=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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()*.

> **Note**
>
> This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

async send_voice(*voice*, *duration=None*, *caption=None*, *disable_notification=None*, *reply_markup=None*, *parse_mode=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=None*, *business_connection_id=None*, *message_effect_id=None*, *allow_paid_broadcast=None*, *direct_messages_topic_id=None*, *suggested_post_parameters=None*, ***, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *filename=None*, *read_timeout=None*, *write_timeout=None*, *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()*.

> **Note**
>
> This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns
On success, instance representing the message posted.

Return type
telegram.Message

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

Shortcut for:

```
await bot.set_chat_member_tag(user_id=update.effective_user.id, *args,␣
↪**kwargs)
```

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

Added in version 22.7.

Returns
On success, `True` is returned.

Return type
`bool`

async set_managed_bot_access_settings(*is_access_restricted*, *added_user_ids=None*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.set_managed_bot_access_settings(
   user_id=update.effective_user.id,
   *args, **kwargs
)
```

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

Added in version 22.8.

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_user.id, *argss,␣
↪**kwargs)
```

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

See also

get_menu_button()

Note

This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Added 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()*.

> **Note**
>
> This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns
On success, `True` is returned.

Return type
`bool`

async unpin_message(*message_id=None*, *business_connection_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()*.

> **Note**
>
> This shortcuts build on the assumption that *User.id* coincides with the *Chat.id* of the private chat with the user. This has been the case so far, but Telegram does not guarantee that this stays this way.

Returns
On success, `True` is returned.

Return type
`bool`

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

Shortcut for:

```
await bot.verify_user(user_id=update.effective_user.id, *args, **kwargs)
```

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

Added in version 21.10.

Returns
On success, `True` is returned.

Return type
`bool`

UserChatBoosts

Added in version 20.8.

class telegram.**UserChatBoosts**(*boosts*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a list of boosts added to a chat by a user.

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

Returned In

telegram.Bot.get_user_chat_boosts()

Added in version 20.8.

Parameters

boosts (Sequence[*telegram.ChatBoost*]) – List of boosts added to the chat by the user.

boosts

List of boosts added to the chat by the user.

Type

tuple[*telegram.ChatBoost*]

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

UserProfileAudios

class telegram.**UserProfileAudios**(*total_count*, *audios*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents the audios displayed on a user's profile.

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

Returned In

telegram.Bot.get_user_profile_audios()

Added in version 22.7.

Parameters

- ***total_count*** (int) – Total number of profile audios for the target user.
- ***audios*** (Sequence[*telegram.Audio*]) – Requested profile audios.

total_count

Total number of profile audios for the target user.

Type

int

audios

Requested profile audios.

Type

tuple[*telegram.Audio*]

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

UserProfilePhotos

class telegram.**UserProfilePhotos**(*total_count*, *photos*, ***, *api_kwargs=None*)

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.

Returned In

telegram.Bot.get_user_profile_photos()

Parameters

- ***total_count*** (int) – Total number of profile pictures the target user has.
- ***photos*** (Sequence[Sequence[*telegram.PhotoSize*]]) – Requested profile pictures (in up to 4 sizes each).

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.

total_count

Total number of profile pictures.

Type
int

photos

Requested profile pictures (in up to 4 sizes each).

Changed in version 20.0: This attribute is now an immutable tuple.

Type
tuple[tuple[*telegram.PhotoSize*]]

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

UserRating

class telegram.**UserRating**(*level*, *rating*, *current_level_rating*, *next_level_rating=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object describes the rating of a user based on their Telegram Star spendings.

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

Available In

telegram.ChatFullInfo.rating

Added in version 22.6.

Parameters

- ***level*** (int) – Current level of the user, indicating their reliability when purchasing digital goods and services. A higher level suggests a more trustworthy customer; a negative level is likely reason for concern.
- ***rating*** (int) – Numerical value of the user's rating; the higher the rating, the better
- ***current_level_rating*** (int) – The rating value required to get the current level
- ***next_level_rating*** (int, optional) – The rating value required to get to the next level; omitted if the maximum level was reached

level

Current level of the user, indicating their reliability when purchasing digital goods and services. A higher level suggests a more trustworthy customer; a negative level is likely reason for concern.

Type
int

rating

Numerical value of the user's rating; the higher the rating, the better

Type
int

current_level_rating

The rating value required to get the current level

Type
int

next_level_rating

Optional. The rating value required to get to the next level; omitted if the maximum level was reached

Type
int

UsersShared

class telegram.**UsersShared**(*request_id*, *users*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object contains information about the user whose identifier was shared with the bot using a *telegram.KeyboardButtonRequestUsers* button.

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

Available In

telegram.Message.users_shared

Added in version 20.8: Bot API 7.0 replaces `UserShared` with this class. The only difference is that now the `user_ids` is a sequence instead of a single integer.

Changed in version 21.1: The argument *users* is now considered for the equality comparison instead of `user_ids`.

Removed in version 21.2: Removed the deprecated argument and attribute `user_ids`.

Parameters

- ***request_id*** (int) – Identifier of the request.

- **users** (Sequence[*telegram.SharedUser*]) – Information about users shared with the bot.

  Added in version 21.1.

  Changed in version 21.2: This argument is now required.

request_id

Identifier of the request.

Type
int

users

Information about users shared with the bot.

Added in version 21.1.

Type
tuple[*telegram.SharedUser*]

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

Venue

class telegram.**Venue**(*location*, *title*, *address*, *foursquare_id=None*, *foursquare_type=None*, *google_place_id=None*, *google_place_type=None*, ***, *api_kwargs=None*)

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 Place details are mutually exclusive. However, this behaviour is undocumented and might be changed by Telegram.

Use In

telegram.Bot.send_venue()

Available In

- *telegram.ExternalReplyInfo.venue*
- *telegram.Message.effective_attachment*
- *telegram.Message.venue*
- *telegram.PollMedia.venue*

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.)

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. (For example, "arts_entertainment/default", "arts_entertainment/aquarium" or "food/icecream".)

Type
str

google_place_id

Optional. Google Places identifier of the venue.

Type
str

google_place_type

Optional. Google Places type of the venue. (See supported types.)

Type
str

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

Video

class telegram.**Video**(*file_id*, *file_unique_id*, *width*, *height*, *duration*, *mime_type=None*, *file_size=None*, *file_name=None*, *thumbnail=None*, *cover=None*, *start_timestamp=None*, *qualities=None*, *, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.get_file()*
- *telegram.Bot.send_video()*

Available In

- *telegram.ExternalReplyInfo.video*
- *telegram.InputMediaLivePhoto.media*
- *telegram.InputMediaVideo.media*
- *telegram.InputPaidMediaLivePhoto.media*
- *telegram.InputPaidMediaVideo.media*
- *telegram.Message.effective_attachment*
- *telegram.Message.video*
- *telegram.PaidMediaVideo.video*
- *telegram.PollMedia.video*

Changed in version 20.5: Removed the deprecated argument and attribute `thumb`.

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 the sender.
- ***height*** (`int`) – Video height as defined by the sender.
- ***duration*** (`int` | `datetime.timedelta`) – Duration of the video in seconds as defined by the sender.

  Changed in version v22.2: `datetime.timedelta` objects are accepted in addition to plain `int` values.
- ***file_name*** (`str`, optional) – Original filename as defined by the sender.
- ***mime_type*** (`str`, optional) – MIME type of a file as defined by the sender.
- ***file_size*** (`int`, optional) – File size in bytes.
- ***thumbnail*** (*telegram.PhotoSize*, optional) – Video thumbnail.

  Added in version 20.2.
- ***cover*** (Sequence[*telegram.PhotoSize*], optional) – Available sizes of the cover of the video in the message.

  Added in version 21.11.
- ***start_timestamp*** (`int` | `datetime.timedelta`, optional) – Timestamp in seconds from which the video will play in the message .. versionadded:: 21.11

Changed in version v22.2: datetime.timedelta objects are accepted in addition to plain int values.

- ***qualities*** (Sequence[*telegram.VideoQuality*], optional) – List of available qualities of the video

  Added in version 22.7.

file_id

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

Type
int

height

Video height as defined by the sender.

Type
int

duration

Duration of the video in seconds as defined by the sender.

Deprecated since version v22.2: In a future major version this attribute will be of type datetime.timedelta. You can opt-in early by setting *PTB_TIMEDELTA=true* or PTB_TIMEDELTA=1 as an environment variable.

Type
int | datetime.timedelta

file_name

Optional. Original filename as defined by the sender.

Type
str

mime_type

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

Type
str

file_size

Optional. File size in bytes.

Type
int

thumbnail

Optional. Video thumbnail.

Added in version 20.2.

Type
telegram.PhotoSize

cover

Optional, Available sizes of the cover of the video in the message.

Added in version 21.11.

Type
tuple[*telegram.PhotoSize*]

start_timestamp

Optional. Timestamp in seconds from which the video will play in the message .. versionadded:: 21.11

Deprecated since version v22.2: In a future major version this attribute will be of type `datetime.timedelta`. You can opt-in early by setting *PTB_TIMEDELTA=true* or `PTB_TIMEDELTA=1` as an environment variable.

Type
`int` | `datetime.timedelta`

qualities

Optional. List of available qualities of the video

Added in version 22.7.

Type
tuple[*telegram.VideoQuality*]

classmethod de_json(*data*, *bot=None*)

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 –

VideoChatEnded

class telegram.**VideoChatEnded**(*duration*, *, *api_kwargs=None*)

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.

Available In

telegram.Message.video_chat_ended

Added in version 13.4.

Changed in version 20.0: This class was renamed from `VoiceChatEnded` in accordance to Bot API 6.0.

Changed in version v22.2: As part of the migration to representing time periods using `datetime.timedelta`, equality comparison now considers integer durations and equivalent timedeltas as equal.

Parameters

duration (int | datetime.timedelta) – Voice chat duration in seconds.

Changed in version v22.2: datetime.timedelta objects are accepted in addition to plain int values.

duration

Voice chat duration in seconds.

Deprecated since version v22.2: In a future major version this attribute will be of type datetime.timedelta. You can opt-in early by setting *PTB_TIMEDELTA=true* or PTB_TIMEDELTA=1 as an environment variable.

Type

int | datetime.timedelta

VideoChatParticipantsInvited

class telegram.**VideoChatParticipantsInvited**(*users*, *, *api_kwargs=None*)

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.

Available In

telegram.Message.video_chat_participants_invited

Added in version 13.4.

Changed in version 20.0: This class was renamed from VoiceChatParticipantsInvited in accordance to Bot API 6.0.

Parameters

users (Sequence[*telegram.User*]) – New members that were invited to the video chat.

Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.

users

New members that were invited to the video chat.

Changed in version 20.0: This attribute is now an immutable tuple.

Type

tuple[*telegram.User*]

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

VideoChatScheduled

class telegram.**VideoChatScheduled**(*start_date*, *, *api_kwargs=None*)

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.

Available In

telegram.Message.video_chat_scheduled

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

Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

start_date

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

Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type

datetime.datetime

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

VideoChatStarted

class telegram.**VideoChatStarted**(***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a service message about a video chat started in the chat. Currently holds no information.

Available In

telegram.Message.video_chat_started

Added in version 13.4.

Changed in version 20.0: This class was renamed from `VoiceChatStarted` in accordance to Bot API 6.0.

VideoNote

class telegram.**VideoNote**(*file_id*, *file_unique_id*, *length*, *duration*, *file_size=None*, *thumbnail=None*, ***, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.get_file()*
- *telegram.Bot.send_video_note()*

Available In

- *telegram.ExternalReplyInfo.video_note*
- *telegram.Message.effective_attachment*
- *telegram.Message.video_note*

Changed in version 20.5: Removed the deprecated argument and attribute `thumb`.

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` | `datetime.timedelta`) – Duration of the video in seconds as defined by the sender.

  Changed in version v22.2: `datetime.timedelta` objects are accepted in addition to plain `int` values.
- ***file_size*** (`int`, optional) – File size in bytes.
- ***thumbnail*** (*telegram.PhotoSize*, optional) – Video thumbnail.

  Added in version 20.2.

file_id

Identifier for this file, which can be used to download or reuse the 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 (diameter of the video message) as defined by sender.

Type
`int`

duration

Duration of the video in seconds as defined by the sender.

Deprecated since version v22.2: In a future major version this attribute will be of type `datetime.timedelta`. You can opt-in early by setting *PTB_TIMEDELTA=true* or `PTB_TIMEDELTA=1` as an environment variable.

Type
`int` | `datetime.timedelta`

file_size

Optional. File size in bytes.

Type
`int`

thumbnail

Optional. Video thumbnail.

Added in version 20.2.

Type
telegram.PhotoSize

classmethod de_json(*data*, *bot=None*)

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 –

VideoQuality

class telegram.**VideoQuality**(*file_id*, *file_unique_id*, *width*, *height*, *codec*, *file_size=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a video file of a specific quality.

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.

Available In

telegram.Video.qualities

Added in version 22.7.

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.
- ***height*** (int) – Video height.
- ***codec*** (str) – Codec that was used to encode the video, for example, `h264`, `h265`, or `av01`
- ***file_size*** (int, optional) – File size in bytes.

file_id

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

Type
int

height

Video height.

Type
int

codec

Codec that was used to encode the video, for example, `h264`, `h265`, or `av01`

Type
str

file_size

Optional. File size in bytes.

Type
int

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 –

Voice

class telegram.**Voice**(*file_id*, *file_unique_id*, *duration*, *mime_type=None*, *file_size=None*, *, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.get_file()*
- *telegram.Bot.send_voice()*

Available In

- *telegram.ExternalReplyInfo.voice*
- *telegram.Message.effective_attachment*

- *telegram.Message.voice*

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 | datetime.timedelta) – Duration of the audio in seconds as defined by the sender.

  Changed in version v22.2: datetime.timedelta objects are accepted in addition to plain int values.
- ***mime_type*** (str, optional) – MIME type of the file as defined by the sender.
- ***file_size*** (int, optional) – File size in bytes.

file_id

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

Deprecated since version v22.2: In a future major version this attribute will be of type datetime.timedelta. You can opt-in early by setting *PTB_TIMEDELTA=true* or PTB_TIMEDELTA=1 as an environment variable.

Type
int | datetime.timedelta

mime_type

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

Type
str

file_size

Optional. File size in bytes.

Type
int

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 –

WebAppData

class telegram.**WebAppData**(*data*, *button_text*, *, *api_kwargs=None*)

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.

Examples

Webapp Bot

Available In

telegram.Message.web_app_data

Added 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

WebAppInfo

class telegram.**WebAppInfo**(*url*, *, *api_kwargs=None*)

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.

Examples

Webapp Bot

Available In

- *telegram.InlineKeyboardButton.web_app*
- *telegram.InlineQueryResultsButton.web_app*
- *telegram.KeyboardButton.web_app*
- *telegram.MenuButtonWebApp.web_app*

Added 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

WebhookInfo

class telegram.**WebhookInfo**(*url*, *has_custom_certificate*, *pending_update_count*, *last_error_date=None*, *last_error_message=None*, *max_connections=None*, *allowed_updates=None*, *ip_address=None*, *last_synchronization_error_date=None*, ***, *api_kwargs=None*)

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.

Returned In

telegram.Bot.get_webhook_info()

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*** (datetime.datetime) – Optional. Datetime for the most recent error that happened when trying to deliver an update via webhook.

  Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

- ***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*** (Sequence[str], optional) – A sequence of update types the bot is subscribed to. Defaults to all update types, except *chat_member*, *message_reaction*, and *message_reaction_count*.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.
- ***last_synchronization_error_date*** (datetime.datetime, optional) – Datetime of the most recent error that happened when trying to synchronize available updates with Telegram datacenters.

  Added in version 20.0.

  Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

url

Webhook URL, may be empty if webhook is not set up.

Type
str

has_custom_certificate

True, if a custom certificate was provided for webhook certificate checks.

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. Datetime for the most recent error that happened when trying to deliver an update via webhook.

Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

last_error_message

Optional. Error message in human-readable format for the most recent error that happened when trying to deliver an update via webhook.

Type
str

max_connections

Optional. Maximum allowed number of simultaneous HTTPS connections to the webhook for update delivery.

Type
int

allowed_updates

Optional. A tuple of update types the bot is subscribed to. Defaults to all update types, except *chat_member*, *message_reaction*, and *message_reaction_count*.

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[str]

last_synchronization_error_date

Datetime of the most recent error that happened when trying to synchronize available updates with Telegram datacenters.

Added in version 20.0.

Changed in version 20.3: The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime, optional

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

WriteAccessAllowed

class telegram.**WriteAccessAllowed**(*web_app_name=None*, *from_request=None*, *from_attachment_menu=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a service message about a user allowing a bot to write messages after adding it to the attachment menu, launching a Web App from a link, or accepting an explicit request from a Web App sent by the method requestWriteAccess.

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

Available In

telegram.Message.write_access_allowed

Added in version 20.0.

Changed in version 20.6: Added custom equality comparison for objects of this class.

Parameters

- ***web_app_name*** (str, optional) – Name of the Web App, if the access was granted when the Web App was launched from a link.

  Added in version 20.3.

- ***from_request*** (bool, optional) – True, if the access was granted after the user accepted an explicit request from a Web App sent by the method requestWriteAccess.

  Added in version 20.6.

- *from_attachment_menu* (bool, optional) – True, if the access was granted when the bot was added to the attachment or side menu.

  Added in version 20.6.

web_app_name

Optional. Name of the Web App, if the access was granted when the Web App was launched from a link.

Added in version 20.3.

Type
str

from_request

Optional. True, if the access was granted after the user accepted an explicit request from a Web App.

Added in version 20.6.

Type
bool

from_attachment_menu

Optional. True, if the access was granted when the bot was added to the attachment or side menu.

Added in version 20.6.

Type
bool

Stickers

The following methods and objects allow your bot to handle stickers and sticker sets.

Gift

class telegram.**Gift**(*id*, *sticker*, *star_count*, *total_count=None*, *remaining_count=None*, *upgrade_star_count=None*, *publisher_chat=None*, *personal_total_count=None*, *personal_remaining_count=None*, *background=None*, *is_premium=None*, *has_colors=None*, *unique_gift_variant_count=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a gift that can be sent by the bot.

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

Use In

telegram.Bot.send_gift()

Available In

- *telegram.GiftInfo.gift*
- *telegram.Gifts.gifts*
- *telegram.OwnedGiftRegular.gift*
- *telegram.TransactionPartnerChat.gift*
- *telegram.TransactionPartnerUser.gift*

Added in version 21.8.

Parameters

- ***id*** (*str*) – Unique identifier of the gift.
- ***sticker*** (*Sticker*) – The sticker that represents the gift.
- ***star_count*** (*int*) – The number of Telegram Stars that must be paid to send the sticker.
- ***total_count*** (*int*, optional) – The total number of the gifts of this type that can be sent by all users; for limited gifts only.
- ***remaining_count*** (*int*, optional) – The number of remaining gifts of this type that can be sent by all users; for limited gifts only.
- ***upgrade_star_count*** (*int*, optional) – The number of Telegram Stars that must be paid to upgrade the gift to a unique one.

  Added in version 21.10.
- ***publisher_chat*** (*telegram.Chat*, optional) – Information about the chat that published the gift.

  Added in version 22.4.
- ***personal_total_count*** (*int*, optional) – The total number of gifts of this type that can be sent by the bot; for limited gifts only.

  Added in version 22.6.
- ***personal_remaining_count*** (*int*, optional) – The number of remaining gifts of this type that can be sent by the bot; for limited gifts only.

  Added in version 22.6.
- ***background*** (*GiftBackground*, optional) – Background of the gift.

  Added in version 22.6.
- ***is_premium*** (*bool*, optional) – *True*, if the gift can only be purchased by Telegram Premium subscribers.

  Added in version 22.6.
- ***has_colors*** (*bool*, optional) – *True*, if the gift can be used (after being upgraded) to customize a user's appearance.

  Added in version 22.6.
- ***unique_gift_variant_count*** (*int*, optional) – The total number of different unique gifts that can be obtained by upgrading the gift.

  Added in version 22.6.

id

Unique identifier of the gift.

Type
str

sticker

The sticker that represents the gift.

Type
Sticker

star_count

The number of Telegram Stars that must be paid to send the sticker.

Type
int

total_count

Optional. The total number of the gifts of this type that can be sent by all users; for limited gifts only.

Type
int

remaining_count

Optional. The number of remaining gifts of this type that can be sent by all users; for limited gifts only.

Type
int

upgrade_star_count

Optional. The number of Telegram Stars that must be paid to upgrade the gift to a unique one.

Added in version 21.10.

Type
int

publisher_chat

Optional. Information about the chat that published the gift.

Added in version 22.4.

Type
telegram.Chat

personal_total_count

Optional. The total number of gifts of this type that can be sent by the bot; for limited gifts only.

Added in version 22.6.

Type
int

personal_remaining_count

Optional. The number of remaining gifts of this type that can be sent by the bot; for limited gifts only.

Added in version 22.6.

Type
int

background

Optional. Background of the gift.

Added in version 22.6.

Type
GiftBackground

is_premium

Optional. True, if the gift can only be purchased by Telegram Premium subscribers.

Added in version 22.6.

Type
bool

has_colors

Optional. True, if the gift can be used (after being upgraded) to customize a user's appearance.

Added in version 22.6.

Type
bool

unique_gift_variant_count
Optional. The total number of different unique gifts that can be obtained by upgrading the gift.

Added in version 22.6.

Type
int

classmethod de_json(*data*, *bot=None*)
See *telegram.TelegramObject.de_json()*.

Gifts

class telegram.**Gifts**(*gifts*, *, *api_kwargs=None*)
Bases: *telegram.TelegramObject*

This object represent a list of gifts.

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

Returned In
telegram.Bot.get_available_gifts()

Added in version 21.8.

Parameters
gifts (Sequence[*Gift*]) – The sequence of gifts.

gifts
The sequence of gifts.

Type
tuple[*Gift*]

classmethod de_json(*data*, *bot=None*)
See *telegram.TelegramObject.de_json()*.

InputSticker

class telegram.**InputSticker**(*sticker*, *emoji_list*, *format*, *mask_position=None*, *keywords=None*, *, *api_kwargs=None*)
Bases: *telegram.TelegramObject*

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

Use In
- *telegram.Bot.add_sticker_to_set()*
- *telegram.Bot.create_new_sticker_set()*
- *telegram.Bot.replace_sticker_in_set()*

Added in version 20.2.

Changed in version 21.1: As of Bot API 7.2, the new argument *format* is a required argument, and thus the order of the arguments has changed.

Parameters

- ***sticker*** (str | file object | *InputFile* | bytes | pathlib.Path) – The added sticker. To upload a file, you can either pass a file object (e.g. `open("filename", "rb")`) or the file contents as bytes. If the bot is running in *local_mode*, passing the path of the file (as string or pathlib.Path object) is supported as well. Animated and video stickers can't be uploaded via HTTP URL.
- ***emoji_list*** (Sequence[str]) – Sequence of *1* - *20* emoji associated with the sticker.
- ***mask_position*** (*telegram.MaskPosition*, optional) – Position where the mask should be placed on faces. For "*'mask'*" stickers only.
- ***keywords*** (Sequence[str], optional) – Sequence of 0-*20* search keywords for the sticker with the total length of up to *64* characters. For "*'regular'*" and "*'custom_emoji'*" stickers only.
- ***format*** (str) – Format of the added sticker, must be one of *'static'* for a `.WEBP` or `.PNG` image, *'animated'* for a `.TGS` animation, *'video'* for a `.WEBM` video.

  Added in version 21.1.

sticker

The added sticker.

Type
str | *telegram.InputFile*

emoji_list

Tuple of *1* - *20* emoji associated with the sticker.

Type
tuple[str]

mask_position

Optional. Position where the mask should be placed on faces. For "*'mask'*" stickers only.

Type
telegram.MaskPosition

keywords

Optional. Tuple of 0-*20* search keywords for the sticker with the total length of up to *64* characters. For "*'regular'*" and "*'custom_emoji'*" stickers only. "*'custom_emoji'*" stickers only.

Type
tuple[str]

format

Format of the added sticker, must be one of *'static'* for a `.WEBP` or `.PNG` image, *'animated'* for a `.TGS` animation, *'video'* for a `.WEBM` video.

Added in version 21.1.

Type
str

MaskPosition

class telegram.**MaskPosition**(*point*, *x_shift*, *y_shift*, *scale*, ***, *api_kwargs=None*)

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.

Use In

telegram.Bot.set_sticker_mask_position()

Available In

- *telegram.InputSticker.mask_position*
- *telegram.Sticker.mask_position*

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. For example, choosing `-1.0` will place mask just to the left of the default mask position.

Type
float

y_shift

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.

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

Sticker

class telegram.**Sticker**(*file_id*, *file_unique_id*, *width*, *height*, *is_animated*, *is_video*, *type*, *emoji=None*, *file_size=None*, *set_name=None*, *mask_position=None*, *premium_animation=None*, *custom_emoji_id=None*, *thumbnail=None*, *needs_repainting=None*, ***, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.delete_sticker_from_set()*
- *telegram.Bot.get_file()*
- *telegram.Bot.replace_sticker_in_set()*
- *telegram.Bot.send_sticker()*
- *telegram.Bot.set_sticker_emoji_list()*
- *telegram.Bot.set_sticker_keywords()*
- *telegram.Bot.set_sticker_mask_position()*
- *telegram.Bot.set_sticker_position_in_set()*

Available In

- *telegram.BusinessIntro.sticker*
- *telegram.ExternalReplyInfo.sticker*
- *telegram.Gift.sticker*
- *telegram.InputMediaSticker.media*
- *telegram.Message.effective_attachment*
- *telegram.Message.sticker*
- *telegram.PollMedia.sticker*
- *telegram.StickerSet.stickers*
- *telegram.UniqueGiftModel.sticker*
- *telegram.UniqueGiftSymbol.sticker*

Returned In

- *telegram.Bot.get_custom_emoji_stickers()*
- *telegram.Bot.get_forum_topic_icon_stickers()*

Changed in version 20.5: Removed the deprecated argument and attribute `thumb`.

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.

  Added 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*.

  Added in version 20.0.
- ***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.
- ***premium_animation*** (*telegram.File*, optional) – For premium regular stickers, premium animation for the sticker.

  Added in version 20.0.
- ***custom_emoji_id*** (`str`, optional) – For custom emoji stickers, unique identifier of the custom emoji.

  Added in version 20.0.
- ***thumbnail*** (*telegram.PhotoSize*, optional) – Sticker thumbnail in the `.WEBP` or `.JPG` format.

  Added in version 20.2.
- ***needs_repainting*** (`bool`, optional) – `True`, if the sticker must be repainted to a text color in messages, the color of the Telegram Premium badge in emoji status, white color on chat photos, or another appropriate color in other places.

  Added in version 20.2.

file_id

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

Added 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*.

Added in version 20.0.

Type
str

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

premium_animation

Optional. For premium regular stickers, premium animation for the sticker.

Added in version 20.0.

Type
telegram.File

custom_emoji_id

Optional. For custom emoji stickers, unique identifier of the custom emoji.

Added in version 20.0.

Type
str

thumbnail

Optional. Sticker thumbnail in the `.WEBP` or `.JPG` format.

Added in version 20.2.

Type
telegram.PhotoSize

needs_repainting

Optional. `True`, if the sticker must be repainted to a text color in messages, the color of the Telegram Premium badge in emoji status, white color on chat photos, or another appropriate color in other places.

Added in version 20.2.

Type
bool

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=None*)

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 –

StickerSet

class telegram.**StickerSet**(*name*, *title*, *stickers*, *sticker_type*, *thumbnail=None*, *, *api_kwargs=None*)

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.

Returned In

telegram.Bot.get_sticker_set()

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

Changed in version 21.1: The parameters `is_video` and `is_animated` are deprecated and now made optional. Thus, the order of the arguments had to be changed.

Changed in version 20.5: Removed the deprecated argument and attribute `thumb`.

Removed in version 21.2: Removed the deprecated arguments and attributes `is_animated` and `is_video`.

Parameters

- ***name*** (`str`) – Sticker set name.
- ***title*** (`str`) – Sticker set title.
- ***stickers*** (Sequence[*telegram.Sticker*]) – List of all set stickers.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list. The input is converted to a tuple.
- ***sticker_type*** (`str`) – Type of stickers in the set, currently one of *telegram.Sticker.REGULAR*, *telegram.Sticker.MASK*, *telegram.Sticker.CUSTOM_EMOJI*.

  Added in version 20.0.
- ***thumbnail*** (*telegram.PhotoSize*, optional) – Sticker set thumbnail in the `.WEBP`, `.TGS`, or `.WEBM` format.

  Added in version 20.2.

name

Sticker set name.

Type
`str`

title

Sticker set title.

Type
`str`

stickers

List of all set stickers.

Changed in version 20.0: This attribute is now an immutable tuple.

Type
tuple[*telegram.Sticker*]

sticker_type

Type of stickers in the set, currently one of *telegram.Sticker.REGULAR*, *telegram.Sticker.MASK*, *telegram.Sticker.CUSTOM_EMOJI*.

Added in version 20.0.

Type
str

thumbnail

Optional. Sticker set thumbnail in the `.WEBP`, `.TGS`, or `.WEBM` format.

Added in version 20.2.

Type
telegram.PhotoSize

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

Inline Mode

The following methods and objects allow your bot to work in inline mode. Please see Telegrams Introduction to Inline bots for more details.

To enable this option, send the `/setinline` command to @BotFather and provide the placeholder text that the user will see in the input field after typing your bot's name.

ChosenInlineResult

class telegram.**ChosenInlineResult**(*result_id*, *from_user*, *query*, *location=None*, *inline_message_id=None*, ***, *api_kwargs=None*)

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.

Available In

telegram.Update.chosen_inline_result

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.

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, only for bots that require user location.

Type
telegram.Location

inline_message_id

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.

Type
str

query

The query that was used to obtain the result.

Type
str

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

InlineQuery

class telegram.**InlineQuery**(*id*, *from_user*, *query*, *offset*, *location=None*, *chat_type=None*, ***, *api_kwargs=None*)

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.

See also

The *telegram.InlineQueryResult* classes represent the media the user can choose from (see above figure).

Note

In Python `from` is a reserved word. Use *from_user* instead.

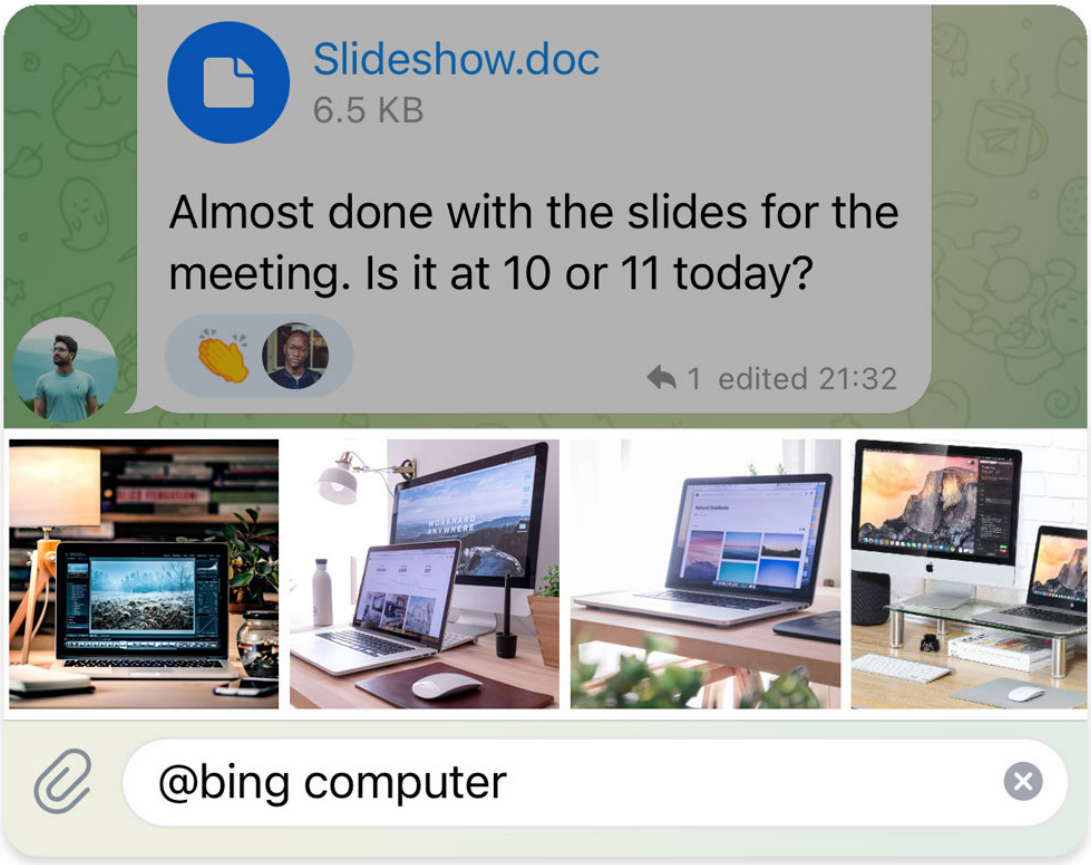

Fig. 3: Inline queries on Telegram

Available In

telegram.Update.inline_query

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.

Changed in version 22.0: Removed constants `MIN_START_PARAMETER_LENGTH` and `MAX_START_PARAMETER_LENGTH`. Use *telegram.constants.InlineQueryResultsButtonLimit.MIN_START_PARAMETER_LENGTH* and *telegram.constants.InlineQueryResultsButtonLimit.MAX_START_PARAMETER_LENGTH* instead.

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.

  Added in version 13.5.
- ***location*** (*telegram.Location*, optional) – Sender location, only for bots that request user location.

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`

chat_type

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.

Added in version 13.5.

Type
str

location

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

Type
telegram.Location

MAX_OFFSET_LENGTH = 64

telegram.constants.InlineQueryLimit.MAX_OFFSET_LENGTH

Added in version 20.0.

MAX_QUERY_LENGTH = 256

telegram.constants.InlineQueryLimit.MAX_QUERY_LENGTH

Added in version 20.0.

MAX_RESULTS = 50

telegram.constants.InlineQueryLimit.RESULTS

Added in version 13.2.

async answer(*results*, *cache_time=None*, *is_personal=None*, *next_offset=None*, *button=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=None*)

See *telegram.TelegramObject.de_json()*.

InlineQueryResult

class telegram.**InlineQueryResult**(*type*, *id*, ***, *api_kwargs=None*)

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*.

Examples

Inline Bot

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

Parameters

- ***type*** (str) – Type of the result.
- ***id*** (str) – Unique identifier for this result, *1- 64* Bytes.

type

Type of the result.

Type
str

id

Unique identifier for this result, *1- 64* Bytes.

Type
str

MAX_ID_LENGTH = 64

telegram.constants.InlineQueryResultLimit.MAX_ID_LENGTH

Added in version 20.0.

MIN_ID_LENGTH = 1

telegram.constants.InlineQueryResultLimit.MIN_ID_LENGTH

Added in version 20.0.

InlineQueryResultArticle

class telegram.**InlineQueryResultArticle**(*id*, *title*, *input_message_content*, *reply_markup=None*, *url=None*, *description=None*, *thumbnail_url=None*, *thumbnail_width=None*, *thumbnail_height=None*, ***, *api_kwargs=None*)

Bases: *telegram.InlineQueryResult*

This object represents a Telegram InlineQueryResultArticle.

Examples

Inline Bot

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

Changed in version 20.5: Removed the deprecated arguments and attributes `thumb_*`.

Changed in version 21.11: Removed the deprecated argument and attribute `hide_url`.

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.

  Tip

  Pass an empty string as URL if you don't want the URL to be shown in the message.

- ***description*** (str, optional) – Short description of the result.
- ***thumbnail_url*** (str, optional) – Url of the thumbnail for the result.

  Added in version 20.2.
- ***thumbnail_width*** (int, optional) – Thumbnail width.

  Added in version 20.2.
- ***thumbnail_height*** (int, optional) – Thumbnail height.

  Added in version 20.2.

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

description

Optional. Short description of the result.

Type
str

thumbnail_url

Optional. Url of the thumbnail for the result.

Added in version 20.2.

Type
str

thumbnail_width

Optional. Thumbnail width.

Added in version 20.2.

Type
int

thumbnail_height

Optional. Thumbnail height.

Added in version 20.2.

Type
int

InlineQueryResultAudio

class telegram.**InlineQueryResultAudio**(*id*, *audio_url*, *title*, *performer=None*, *audio_duration=None*, *caption=None*, *reply_markup=None*, *input_message_content=None*, *parse_mode=None*, *caption_entities=None*, ***, *api_kwargs=None*)

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.

See also

Working with Files and Media

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

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*** (int | datetime.timedelta, optional) – Audio duration in seconds.

  Changed in version v22.2: datetime.timedelta objects are accepted in addition to plain int values.
- ***caption*** (str, optional) – Caption, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of parse_mode.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.
- ***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.

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.

Deprecated since version v22.2: In a future major version this attribute will be of type datetime.timedelta. You can opt-in early by setting *PTB_TIMEDELTA=true* or PTB_TIMEDELTA=1 as an environment variable.

Type
int | datetime.timedelta

caption

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

Type
str

parse_mode

Optional. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*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

InlineQueryResultCachedAudio

class telegram.**InlineQueryResultCachedAudio**(*id*, *audio_file_id*, *caption=None*, *reply_markup=None*, *input_message_content=None*, *parse_mode=None*, *caption_entities=None*, *, *api_kwargs=None*)

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.

See also

Working with Files and Media

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of parse_mode.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.
- ***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.

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. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*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

InlineQueryResultCachedDocument

class telegram.**InlineQueryResultCachedDocument**(*id*, *title*, *document_file_id*, *description=None*, *caption=None*, *reply_markup=None*, *input_message_content=None*, *parse_mode=None*, *caption_entities=None*, *, *api_kwargs=None*)

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.

See also

Working with Files and Media

Use In

- *telegram.Bot.answer_guest_query()*

- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of parse_mode.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.
- ***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.

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. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*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

InlineQueryResultCachedGif

class telegram.**InlineQueryResultCachedGif**(*id*, *gif_file_id*, *title=None*, *caption=None*, *reply_markup=None*, *input_message_content=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, ***, *api_kwargs=None*)

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.

See also

Working with Files and Media

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*

- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

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 of the GIF file to be sent, 0-*1024* characters after entities parsing.
- ***parse_mode*** (str, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of `parse_mode`.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list. The input is converted to a tuple.
- ***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.
- ***show_caption_above_media*** (bool, optional) – Pass `True`, if the caption must be shown above the message media.

  Added in version 21.3.

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. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*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

show_caption_above_media

Optional. True, if the caption must be shown above the message media.

Added in version 21.3.

Type
bool

InlineQueryResultCachedMpeg4Gif

class telegram.**InlineQueryResultCachedMpeg4Gif**(*id*, *mpeg4_file_id*, *title=None*, *caption=None*, *reply_markup=None*, *input_message_content=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, ***, *api_kwargs=None*)

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.

See also

Working with Files and Media

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of parse_mode.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.
- ***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.
- ***show_caption_above_media*** (bool, optional) – Pass True, if the caption must be shown above the message media.

  Added in version 21.3.

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. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*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

show_caption_above_media

Optional. `True`, if the caption must be shown above the message media.

Added in version 21.3.

Type
bool

InlineQueryResultCachedPhoto

class telegram.**InlineQueryResultCachedPhoto**(*id*, *photo_file_id*, *title=None*, *description=None*, *caption=None*, *reply_markup=None*, *input_message_content=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, ***, *api_kwargs=None*)

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.

See also

Working with Files and Media

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of parse_mode.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.
- ***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.
- ***show_caption_above_media*** (bool, optional) – Pass True, if the caption must be shown above the message media.

  Added in version 21.3.

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. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*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

show_caption_above_media

Optional. `True`, if the caption must be shown above the message media.

Added in version 21.3.

Type
bool

InlineQueryResultCachedSticker

class telegram.**InlineQueryResultCachedSticker**(*id*, *sticker_file_id*, *reply_markup=None*, *input_message_content=None*, ***, *api_kwargs=None*)

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.

See also

Working with Files and Media

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

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.

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

InlineQueryResultCachedVideo

class telegram.**InlineQueryResultCachedVideo**(*id*, *video_file_id*, *title*, *description=None*, *caption=None*, *reply_markup=None*, *input_message_content=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, ***, *api_kwargs=None*)

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.

See also

Working with Files and Media

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence 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.
- ***show_caption_above_media*** (bool, optional) – Pass `True`, if the caption must be shown above the message media.

  Added in version 21.3.

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. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*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

show_caption_above_media

Optional. `True`, if the caption must be shown above the message media.

Added in version 21.3.

Type
bool

InlineQueryResultCachedVoice

class telegram.**InlineQueryResultCachedVoice**(*id*, *voice_file_id*, *title*, *caption=None*, *reply_markup=None*, *input_message_content=None*, *parse_mode=None*, *caption_entities=None*, *, *api_kwargs=None*)

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.

See also

Working with Files and Media

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Tuple of special entities that appear in the caption, which can be specified instead of parse_mode.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.
- ***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.

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. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*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

InlineQueryResultContact

class telegram.**InlineQueryResultContact**(*id*, *phone_number*, *first_name*, *last_name=None*, *reply_markup=None*, *input_message_content=None*, *vcard=None*, *thumbnail_url=None*, *thumbnail_width=None*, *thumbnail_height=None*, ***, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

Changed in version 20.5: Removed the deprecated arguments and attributes `thumb_*`.

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.
- ***thumbnail_url*** (`str`, optional) – Url of the thumbnail for the result.

  Added in version 20.2.
- ***thumbnail_width*** (`int`, optional) – Thumbnail width.

  Added in version 20.2.
- ***thumbnail_height*** (`int`, optional) – Thumbnail height.

  Added in version 20.2.

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

thumbnail_url

Optional. Url of the thumbnail for the result.

Added in version 20.2.

Type
str

thumbnail_width

Optional. Thumbnail width.

Added in version 20.2.

Type
int

thumbnail_height

Optional. Thumbnail height.

Added in version 20.2.

Type
int

InlineQueryResultDocument

class telegram.**InlineQueryResultDocument**(*id*, *document_url*, *title*, *mime_type*, *caption=None*, *description=None*, *reply_markup=None*, *input_message_content=None*, *parse_mode=None*, *caption_entities=None*, *thumbnail_url=None*, *thumbnail_width=None*, *thumbnail_height=None*, ***, *api_kwargs=None*)

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.

See also

Working with Files and Media

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

Changed in version 20.5: Removed the deprecated arguments and attributes `thumb_*`.

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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of `parse_mode`.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list. The input is converted to a tuple.
- ***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.
- ***thumbnail_url*** (`str`, optional) – URL of the thumbnail (JPEG only) for the file.

  Added in version 20.2.
- ***thumbnail_width*** (`int`, optional) – Thumbnail width.

  Added in version 20.2.
- ***thumbnail_height*** (`int`, optional) – Thumbnail height.

  Added in version 20.2.

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. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*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

thumbnail_url

Optional. URL of the thumbnail (JPEG only) for the file.

Added in version 20.2.

Type
str

thumbnail_width

Optional. Thumbnail width.

Added in version 20.2.

Type
int

thumbnail_height

Optional. Thumbnail height.

Added in version 20.2.

Type
int

InlineQueryResultGame

class telegram.**InlineQueryResultGame**(*id*, *game_short_name*, *reply_markup=None*, ***, *api_kwargs=None*)

Bases: *telegram.InlineQueryResult*

Represents a *telegram.Game*.

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

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.

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

InlineQueryResultGif

class telegram.**InlineQueryResultGif**(*id*, *gif_url*, *thumbnail_url*, *gif_width=None*, *gif_height=None*, *title=None*, *caption=None*, *reply_markup=None*, *input_message_content=None*, *gif_duration=None*, *parse_mode=None*, *caption_entities=None*, *thumbnail_mime_type=None*, *show_caption_above_media=None*, ***, *api_kwargs=None*)

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.

See also

Working with Files and Media

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

Changed in version 20.5: Removed the deprecated arguments and attributes `thumb_*`.

Parameters

- ***id*** (str) – Unique identifier for this result, *1- 64* Bytes.
- ***gif_url*** (str) – A valid URL for the GIF file.
- ***gif_width*** (int, optional) – Width of the GIF.
- ***gif_height*** (int, optional) – Height of the GIF.
- ***gif_duration*** (int | datetime.timedelta, optional) – Duration of the GIF in seconds.

Changed in version v22.2: datetime.timedelta objects are accepted in addition to plain int values.

- ***thumbnail_url*** (str) – URL of the static (JPEG or GIF) or animated (MPEG4) thumbnail for the result.

  Added in version 20.2.

  Changed in version 20.5: Removal of the deprecated argument thumb_url made thumbnail_url mandatory.

- ***thumbnail_mime_type*** (str, optional) – MIME type of the thumbnail, must be one of 'image/jpeg', 'image/gif', or 'video/mp4'. Defaults to 'image/jpeg'.

  Added in version 20.2.

- ***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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of parse_mode.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.

- ***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.
- ***show_caption_above_media*** (bool, optional) – Pass True, if the caption must be shown above the message media.

  Added in version 21.3.

type

'gif'.

Type
str

id

Unique identifier for this result, *1*- *64* Bytes.

Type
str

gif_url

A valid URL for the GIF file.

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.

Deprecated since version v22.2: In a future major version this attribute will be of type datetime.timedelta. You can opt-in early by setting *PTB_TIMEDELTA=true* or PTB_TIMEDELTA=1 as an environment variable.

Type
int | datetime.timedelta

thumbnail_url

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

Added in version 20.2.

Type
str

thumbnail_mime_type

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

Added in version 20.2.

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. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*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

show_caption_above_media

Optional. `True`, if the caption must be shown above the message media.

Added in version 21.3.

Type
`bool`

InlineQueryResultLocation

class telegram.**InlineQueryResultLocation**(*id*, *latitude*, *longitude*, *title*, *live_period=None*, *reply_markup=None*, *input_message_content=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, *thumbnail_url=None*, *thumbnail_width=None*, *thumbnail_height=None*, ***, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

Changed in version 20.5: Removed the deprecated arguments and attributes `thumb_*`.

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` | `datetime.timedelta`, optional) – Period in seconds for which the location will be updated, should be between *60* and *86400*.

  Changed in version v22.2: `datetime.timedelta` objects are accepted in addition to plain `int` values.
- ***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 *100000* 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.
- ***thumbnail_url*** (str, optional) – Url of the thumbnail for the result.

  Added in version 20.2.
- ***thumbnail_width*** (int, optional) – Thumbnail width.

  Added in version 20.2.
- ***thumbnail_height*** (int, optional) – Thumbnail height.

  Added in version 20.2.

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; 0- *1500*.

Type
float

live_period

Optional. Period in seconds for which the location will be updated, should be between *60* and *86400* or *2147483647* for live locations that can be edited indefinitely.

Deprecated since version v22.2: In a future major version this attribute will be of type datetime.timedelta. You can opt-in early by setting *PTB_TIMEDELTA=true* or PTB_TIMEDELTA=1 as an environment variable.

Type
int | datetime.timedelta

heading

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

Type
int

proximity_alert_radius

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

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

thumbnail_url

Optional. Url of the thumbnail for the result.

Added in version 20.2.

Type
str

thumbnail_width

Optional. Thumbnail width.

Added in version 20.2.

Type
int

thumbnail_height

Optional. Thumbnail height.

Added in version 20.2.

Type
int

HORIZONTAL_ACCURACY = 1500

telegram.constants.LocationLimit.HORIZONTAL_ACCURACY

Added in version 20.0.

MAX_HEADING = 360

telegram.constants.LocationLimit.MAX_HEADING

Added in version 20.0.

MAX_LIVE_PERIOD = 86400

telegram.constants.LocationLimit.MAX_LIVE_PERIOD

Added in version 20.0.

MAX_PROXIMITY_ALERT_RADIUS = 100000

telegram.constants.LocationLimit.MAX_PROXIMITY_ALERT_RADIUS

Added in version 20.0.

MIN_HEADING = 1

telegram.constants.LocationLimit.MIN_HEADING

Added in version 20.0.

MIN_LIVE_PERIOD = 60

telegram.constants.LocationLimit.MIN_LIVE_PERIOD

Added in version 20.0.

MIN_PROXIMITY_ALERT_RADIUS = 1

telegram.constants.LocationLimit.MIN_PROXIMITY_ALERT_RADIUS

Added in version 20.0.

InlineQueryResultMpeg4Gif

class telegram.**InlineQueryResultMpeg4Gif**(*id*, *mpeg4_url*, *thumbnail_url*, *mpeg4_width=None*, *mpeg4_height=None*, *title=None*, *caption=None*, *reply_markup=None*, *input_message_content=None*, *mpeg4_duration=None*, *parse_mode=None*, *caption_entities=None*, *thumbnail_mime_type=None*, *show_caption_above_media=None*, ***, *api_kwargs=None*)

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.

See also

Working with Files and Media

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

Changed in version 20.5: Removed the deprecated arguments and attributes `thumb_*`.

Parameters

- ***id*** (`str`) – Unique identifier for this result, *1- 64* Bytes.
- ***mpeg4_url*** (`str`) – A valid URL for the MP4 file.
- ***mpeg4_width*** (`int`, optional) – Video width.
- ***mpeg4_height*** (`int`, optional) – Video height.
- ***mpeg4_duration*** (`int` | `datetime.timedelta`, optional) – Video duration in seconds.

Changed in version v22.2: datetime.timedelta objects are accepted in addition to plain int values.

- ***thumbnail_url*** (str) – URL of the static (JPEG or GIF) or animated (MPEG4) thumbnail for the result.

  Added in version 20.2.

  Changed in version 20.5: Removal of the deprecated argument thumb_url made thumbnail_url mandatory.

- ***thumbnail_mime_type*** (str, optional) – MIME type of the thumbnail, must be one of 'image/jpeg', 'image/gif', or 'video/mp4'. Defaults to 'image/jpeg'.

  Added in version 20.2.

- ***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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Tuple of special entities that appear in the caption, which can be specified instead of parse_mode.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.

- ***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.
- ***show_caption_above_media*** (bool, optional) – Pass True, if the caption must be shown above the message media.

  Added in version 21.3.

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.

Type
str

mpeg4_width

Optional. Video width.

Type
int

mpeg4_height

Optional. Video height.

Type
int

mpeg4_duration

Optional. Video duration in seconds.

Deprecated since version v22.2: In a future major version this attribute will be of type datetime.timedelta. You can opt-in early by setting *PTB_TIMEDELTA=true* or PTB_TIMEDELTA=1 as an environment variable.

Type
int | datetime.timedelta

thumbnail_url

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

Added in version 20.2.

Type
str

thumbnail_mime_type

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

Added in version 20.2.

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. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*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

show_caption_above_media

Optional. `True`, if the caption must be shown above the message media.

Added in version 21.3.

Type
`bool`

InlineQueryResultPhoto

class telegram.**InlineQueryResultPhoto**(*id*, *photo_url*, *thumbnail_url*, *photo_width=None*, *photo_height=None*, *title=None*, *description=None*, *caption=None*, *reply_markup=None*, *input_message_content=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, ***, *api_kwargs=None*)

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.

See also

Working with Files and Media

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

Changed in version 20.5: Removed the deprecated argument and attribute `thumb_url` which made thumbnail_url mandatory.

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.
- ***thumbnail_url*** (`str`) – URL of the thumbnail for the photo.

  Added in version 20.2.

  Changed in version 20.5: Removal of the deprecated argument `thumb_url` made `thumbnail_url` mandatory.

- ***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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of parse_mode.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.
- ***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.
- ***show_caption_above_media*** (bool, optional) – Pass True, if the caption must be shown above the message media.

  Added in version 21.3.

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

thumbnail_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. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*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

show_caption_above_media

Optional. `True`, if the caption must be shown above the message media.

Added in version 21.3.

Type
bool

InlineQueryResultsButton

class telegram.**InlineQueryResultsButton**(*text*, *web_app=None*, *start_parameter=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *text*, *web_app* and *start_parameter* are equal.

Use In

telegram.Bot.answer_inline_query()

Parameters

- ***text*** (str) – Label text on the button.
- ***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 switch back to the inline mode using the method switchInlineQuery inside the Web App.
- ***start_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.

  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.

text

Label text on the button.

Type
str

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 switch back to the inline mode using the method web_app_switch_inline_query inside the Web App.

Type
telegram.WebAppInfo

start_parameter

Optional. Deep-linking parameter for the *1* - *64* characters, only A-Z, a-z, 0-9, _ and - are allowed.

Type
str

MAX_START_PARAMETER_LENGTH = 64

telegram.constants.InlineQueryResultsButtonLimit.MAX_START_PARAMETER_LENGTH

MIN_START_PARAMETER_LENGTH = 1

telegram.constants.InlineQueryResultsButtonLimit.MIN_START_PARAMETER_LENGTH

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

InlineQueryResultVenue

class telegram.**InlineQueryResultVenue**(*id*, *latitude*, *longitude*, *title*, *address*, *foursquare_id=None*, *foursquare_type=None*, *reply_markup=None*, *input_message_content=None*, *google_place_id=None*, *google_place_type=None*, *thumbnail_url=None*, *thumbnail_width=None*, *thumbnail_height=None*, ***, *api_kwargs=None*)

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.

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

Changed in version 20.5: Removed the deprecated arguments and attributes `thumb_*`.

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 venue.
- ***thumbnail_url*** (str, optional) – Url of the thumbnail for the result.

  Added in version 20.2.

- ***thumbnail_width*** (int, optional) – Thumbnail width.

  Added in version 20.2.

- ***thumbnail_height*** (int, optional) – Thumbnail height.

  Added in version 20.2.

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. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”.)

Type
str

google_place_id

Optional. Google Places identifier of the venue.

Type
str

google_place_type

Optional. Google Places type of the venue. (See supported types.)

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

thumbnail_url

Optional. Url of the thumbnail for the result.

Added in version 20.2.

Type
str

thumbnail_width

Optional. Thumbnail width.

Added in version 20.2.

Type
int

thumbnail_height

Optional. Thumbnail height.

Added in version 20.2.

Type
int

InlineQueryResultVideo

class telegram.**InlineQueryResultVideo**(*id*, *video_url*, *mime_type*, *thumbnail_url*, *title*, *caption=None*, *video_width=None*, *video_height=None*, *video_duration=None*, *description=None*, *reply_markup=None*, *input_message_content=None*, *parse_mode=None*, *caption_entities=None*, *show_caption_above_media=None*, ***, *api_kwargs=None*)

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*.

See also

Working with Files and Media

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_message()*

Changed in version 20.5: Removed the deprecated argument and attribute `thumb_url` which made thumbnail_url mandatory.

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".
- ***thumbnail_url*** (`str`, optional) – URL of the thumbnail (JPEG only) for the video.

  Added in version 20.2.

  Changed in version 20.5: Removal of the deprecated argument `thumb_url` made `thumbnail_url` mandatory.
- ***title*** (`str`) – Title for the result.
- ***caption*** (`str`, optional) – Caption of the video to be sent, 0-*1024* characters after entities parsing.
- ***parse_mode*** (`str`, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of `parse_mode`.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list. The input is converted to a tuple.
- ***video_width*** (`int`, optional) – Video width.
- ***video_height*** (`int`, optional) – Video height.
- ***video_duration*** (`int` | `datetime.timedelta`, optional) – Video duration in seconds.

  Changed in version v22.2: `datetime.timedelta` objects are accepted in addition to plain `int` values.
- ***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).
- ***show_caption_above_media*** (`bool`, optional) – Pass `True`, if the caption must be shown above the message media.

  Added in version 21.3.

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

thumbnail_url

URL of the thumbnail (JPEG only) for the video.

Added in version 20.2.

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. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*telegram.MessageEntity*]

video_width

Optional. Video width.

Type
int

video_height

Optional. Video height.

Type
int

video_duration

Optional. Video duration in seconds.

Deprecated since version v22.2: In a future major version this attribute will be of type datetime.timedelta. You can opt-in early by setting *PTB_TIMEDELTA=true* or PTB_TIMEDELTA=1 as an environment variable.

Type
int | datetime.timedelta

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

show_caption_above_media

Optional. True, if the caption must be shown above the message media.

Added in version 21.3.

Type
bool

InlineQueryResultVoice

class telegram.**InlineQueryResultVoice**(*id*, *voice_url*, *title*, *voice_duration=None*, *caption=None*, *reply_markup=None*, *input_message_content=None*, *parse_mode=None*, *caption_entities=None*, ***, *api_kwargs=None*)

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 voice message.

See also

Working with Files and Media

Use In

- *telegram.Bot.answer_guest_query()*
- *telegram.Bot.answer_inline_query()*
- *telegram.Bot.answer_web_app_query()*
- *telegram.Bot.save_prepared_inline_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) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***caption_entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of parse_mode.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.
- ***voice_duration*** (int | datetime.timedelta, optional) – Recording duration in seconds.

  Changed in version v22.2: datetime.timedelta objects are accepted in addition to plain int values.
- ***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.

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. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

caption_entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*telegram.MessageEntity*]

voice_duration

Optional. Recording duration in seconds.

Deprecated since version v22.2: In a future major version this attribute will be of type `datetime.timedelta`. You can opt-in early by setting *PTB_TIMEDELTA=true* or `PTB_TIMEDELTA=1` as an environment variable.

Type
int | datetime.timedelta

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

InputMessageContent

class telegram.**InputMessageContent**(*, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Base class for Telegram InputMessageContent Objects.

See: *telegram.InputContactMessageContent*, *telegram.InputInvoiceMessageContent*, *telegram.InputLocationMessageContent*, *telegram.InputTextMessageContent* and *telegram.InputVenueMessageContent* for more details.

Available In

- *telegram.InlineQueryResultArticle.input_message_content*
- *telegram.InlineQueryResultAudio.input_message_content*
- *telegram.InlineQueryResultCachedAudio.input_message_content*
- *telegram.InlineQueryResultCachedDocument.input_message_content*
- *telegram.InlineQueryResultCachedGif.input_message_content*
- *telegram.InlineQueryResultCachedMpeg4Gif.input_message_content*
- *telegram.InlineQueryResultCachedPhoto.input_message_content*
- *telegram.InlineQueryResultCachedSticker.input_message_content*
- *telegram.InlineQueryResultCachedVideo.input_message_content*
- *telegram.InlineQueryResultCachedVoice.input_message_content*
- *telegram.InlineQueryResultContact.input_message_content*
- *telegram.InlineQueryResultDocument.input_message_content*
- *telegram.InlineQueryResultGif.input_message_content*
- *telegram.InlineQueryResultLocation.input_message_content*
- *telegram.InlineQueryResultMpeg4Gif.input_message_content*
- *telegram.InlineQueryResultPhoto.input_message_content*
- *telegram.InlineQueryResultVenue.input_message_content*
- *telegram.InlineQueryResultVideo.input_message_content*
- *telegram.InlineQueryResultVoice.input_message_content*

InputTextMessageContent

class telegram.**InputTextMessageContent**(*message_text*, *parse_mode=None*, *entities=None*, *link_preview_options=None*, ***, *disable_web_page_preview=None*, *api_kwargs=None*)

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.

Examples

Inline Bot

Available In

- *telegram.InlineQueryResultArticle.input_message_content*
- *telegram.InlineQueryResultAudio.input_message_content*
- *telegram.InlineQueryResultCachedAudio.input_message_content*
- *telegram.InlineQueryResultCachedDocument.input_message_content*

- *telegram.InlineQueryResultCachedGif.input_message_content*
- *telegram.InlineQueryResultCachedMpeg4Gif.input_message_content*
- *telegram.InlineQueryResultCachedPhoto.input_message_content*
- *telegram.InlineQueryResultCachedSticker.input_message_content*
- *telegram.InlineQueryResultCachedVideo.input_message_content*
- *telegram.InlineQueryResultCachedVoice.input_message_content*
- *telegram.InlineQueryResultContact.input_message_content*
- *telegram.InlineQueryResultDocument.input_message_content*
- *telegram.InlineQueryResultGif.input_message_content*
- *telegram.InlineQueryResultLocation.input_message_content*
- *telegram.InlineQueryResultMpeg4Gif.input_message_content*
- *telegram.InlineQueryResultPhoto.input_message_content*
- *telegram.InlineQueryResultVenue.input_message_content*
- *telegram.InlineQueryResultVideo.input_message_content*
- *telegram.InlineQueryResultVoice.input_message_content*

Parameters

- ***message_text*** (str) – Text of the message to be sent, *1- 4096* characters after entities parsing.
- ***parse_mode*** (str, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***entities*** (Sequence[*telegram.MessageEntity*], optional) – Sequence of special entities that appear in the caption, which can be specified instead of parse_mode.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.
- ***link_preview_options*** (*LinkPreviewOptions*, optional) – Link preview generation options for the message. Mutually exclusive with *disable_web_page_preview*.

  Added in version 20.8.

Keyword Arguments

disable_web_page_preview (bool, optional) – Disables link previews for links in the sent message. Convenience parameter for setting *link_preview_options*. Mutually exclusive with *link_preview_options*.

Changed in version 20.8: Bot API 7.0 introduced *link_preview_options* replacing this argument. PTB will automatically convert this argument to that one, but for advanced options, please use *link_preview_options* directly.

Changed in version 21.0: This argument is now a keyword-only argument.

message_text

Text of the message to be sent, *1- 4096* characters after entities parsing.

Type

str

parse_mode

Optional. Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.

Type
str

entities

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

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*telegram.MessageEntity*]

link_preview_options

Optional. Link preview generation options for the message.

Added in version 20.8.

Type
LinkPreviewOptions

InputLocationMessageContent

class telegram.**InputLocationMessageContent**(*latitude*, *longitude*, *live_period=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, ***, *api_kwargs=None*)

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.

Available In

- *telegram.InlineQueryResultArticle.input_message_content*
- *telegram.InlineQueryResultAudio.input_message_content*
- *telegram.InlineQueryResultCachedAudio.input_message_content*
- *telegram.InlineQueryResultCachedDocument.input_message_content*
- *telegram.InlineQueryResultCachedGif.input_message_content*
- *telegram.InlineQueryResultCachedMpeg4Gif.input_message_content*
- *telegram.InlineQueryResultCachedPhoto.input_message_content*
- *telegram.InlineQueryResultCachedSticker.input_message_content*
- *telegram.InlineQueryResultCachedVideo.input_message_content*
- *telegram.InlineQueryResultCachedVoice.input_message_content*
- *telegram.InlineQueryResultContact.input_message_content*
- *telegram.InlineQueryResultDocument.input_message_content*
- *telegram.InlineQueryResultGif.input_message_content*
- *telegram.InlineQueryResultLocation.input_message_content*
- *telegram.InlineQueryResultMpeg4Gif.input_message_content*

- *telegram.InlineQueryResultPhoto.input_message_content*
- *telegram.InlineQueryResultVenue.input_message_content*
- *telegram.InlineQueryResultVideo.input_message_content*
- *telegram.InlineQueryResultVoice.input_message_content*

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 | datetime.timedelta, optional) – Period in seconds for which the location will be updated, should be between *60* and *86400* or *2147483647* for live locations that can be edited indefinitely.

  Changed in version v22.2: datetime.timedelta objects are accepted in addition to plain int values.
- ***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 *100000* if specified.

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; 0- *1500*.

Type
float

live_period

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

Deprecated since version v22.2: In a future major version this attribute will be of type datetime.timedelta. You can opt-in early by setting *PTB_TIMEDELTA=true* or PTB_TIMEDELTA=1 as an environment variable.

Type
int | datetime.timedelta

heading

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

Type
int

proximity_alert_radius

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

Type
int

HORIZONTAL_ACCURACY = 1500

telegram.constants.LocationLimit.HORIZONTAL_ACCURACY

Added in version 20.0.

MAX_HEADING = 360

telegram.constants.LocationLimit.MAX_HEADING

Added in version 20.0.

MAX_LIVE_PERIOD = 86400

telegram.constants.LocationLimit.MAX_LIVE_PERIOD

Added in version 20.0.

MAX_PROXIMITY_ALERT_RADIUS = 100000

telegram.constants.LocationLimit.MAX_PROXIMITY_ALERT_RADIUS

Added in version 20.0.

MIN_HEADING = 1

telegram.constants.LocationLimit.MIN_HEADING

Added in version 20.0.

MIN_LIVE_PERIOD = 60

telegram.constants.LocationLimit.MIN_LIVE_PERIOD

Added in version 20.0.

MIN_PROXIMITY_ALERT_RADIUS = 1

telegram.constants.LocationLimit.MIN_PROXIMITY_ALERT_RADIUS

Added in version 20.0.

InputVenueMessageContent

class telegram.**InputVenueMessageContent**(*latitude*, *longitude*, *title*, *address*, *foursquare_id=None*, *foursquare_type=None*, *google_place_id=None*, *google_place_type=None*, ***, *api_kwargs=None*)

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.

Available In

- *telegram.InlineQueryResultArticle.input_message_content*

- *telegram.InlineQueryResultAudio.input_message_content*
- *telegram.InlineQueryResultCachedAudio.input_message_content*
- *telegram.InlineQueryResultCachedDocument.input_message_content*
- *telegram.InlineQueryResultCachedGif.input_message_content*
- *telegram.InlineQueryResultCachedMpeg4Gif.input_message_content*
- *telegram.InlineQueryResultCachedPhoto.input_message_content*
- *telegram.InlineQueryResultCachedSticker.input_message_content*
- *telegram.InlineQueryResultCachedVideo.input_message_content*
- *telegram.InlineQueryResultCachedVoice.input_message_content*
- *telegram.InlineQueryResultContact.input_message_content*
- *telegram.InlineQueryResultDocument.input_message_content*
- *telegram.InlineQueryResultGif.input_message_content*
- *telegram.InlineQueryResultLocation.input_message_content*
- *telegram.InlineQueryResultMpeg4Gif.input_message_content*
- *telegram.InlineQueryResultPhoto.input_message_content*
- *telegram.InlineQueryResultVenue.input_message_content*
- *telegram.InlineQueryResultVideo.input_message_content*
- *telegram.InlineQueryResultVoice.input_message_content*

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.)

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. (For example, "arts_entertainment/default", "arts_entertainment/aquarium" or "food/icecream".)

Type
str

google_place_id

Optional. Google Places identifier of the venue.

Type
str

google_place_type

Optional. Google Places type of the venue. (See supported types.)

Type
str

InputContactMessageContent

class telegram.**InputContactMessageContent**(*phone_number*, *first_name*, *last_name=None*, *vcard=None*, *, *api_kwargs=None*)

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.

Available In

- *telegram.InlineQueryResultArticle.input_message_content*
- *telegram.InlineQueryResultAudio.input_message_content*
- *telegram.InlineQueryResultCachedAudio.input_message_content*
- *telegram.InlineQueryResultCachedDocument.input_message_content*
- *telegram.InlineQueryResultCachedGif.input_message_content*
- *telegram.InlineQueryResultCachedMpeg4Gif.input_message_content*
- *telegram.InlineQueryResultCachedPhoto.input_message_content*
- *telegram.InlineQueryResultCachedSticker.input_message_content*
- *telegram.InlineQueryResultCachedVideo.input_message_content*
- *telegram.InlineQueryResultCachedVoice.input_message_content*

- *telegram.InlineQueryResultContact.input_message_content*
- *telegram.InlineQueryResultDocument.input_message_content*
- *telegram.InlineQueryResultGif.input_message_content*
- *telegram.InlineQueryResultLocation.input_message_content*
- *telegram.InlineQueryResultMpeg4Gif.input_message_content*
- *telegram.InlineQueryResultPhoto.input_message_content*
- *telegram.InlineQueryResultVenue.input_message_content*
- *telegram.InlineQueryResultVideo.input_message_content*
- *telegram.InlineQueryResultVoice.input_message_content*

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.

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

InputInvoiceMessageContent

class telegram.**InputInvoiceMessageContent**(*title*, *description*, *payload*, *currency*, *prices*, *provider_token=None*, *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*, ***, *api_kwargs=None*)

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*, *currency* and *prices* are equal.

Available In

- *telegram.InlineQueryResultArticle.input_message_content*
- *telegram.InlineQueryResultAudio.input_message_content*
- *telegram.InlineQueryResultCachedAudio.input_message_content*
- *telegram.InlineQueryResultCachedDocument.input_message_content*
- *telegram.InlineQueryResultCachedGif.input_message_content*
- *telegram.InlineQueryResultCachedMpeg4Gif.input_message_content*
- *telegram.InlineQueryResultCachedPhoto.input_message_content*
- *telegram.InlineQueryResultCachedSticker.input_message_content*
- *telegram.InlineQueryResultCachedVideo.input_message_content*
- *telegram.InlineQueryResultCachedVoice.input_message_content*
- *telegram.InlineQueryResultContact.input_message_content*
- *telegram.InlineQueryResultDocument.input_message_content*
- *telegram.InlineQueryResultGif.input_message_content*
- *telegram.InlineQueryResultLocation.input_message_content*
- *telegram.InlineQueryResultMpeg4Gif.input_message_content*
- *telegram.InlineQueryResultPhoto.input_message_content*
- *telegram.InlineQueryResultVenue.input_message_content*
- *telegram.InlineQueryResultVideo.input_message_content*
- *telegram.InlineQueryResultVoice.input_message_content*

Added in version 13.5.

Changed in version 21.11: *provider_token* is no longer considered for equality comparison.

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 it for your internal processes.
- ***provider_token*** (str, optional) – Payment provider token, obtained via @Botfather. Pass an empty string for payments in Telegram Stars.

  Changed in version 21.11: Bot API 7.4 made this parameter is optional and this is now reflected in the class signature.
- ***currency*** (str) – Three-letter ISO 4217 currency code, see more on currencies. Pass XTR for payments in Telegram Stars.

- ***prices*** (Sequence[*telegram.LabeledPrice*]) – Price breakdown, a list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.). Must contain exactly one item for payments in Telegram Stars.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list. The input is converted to a tuple.

- ***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`. Not supported for payments in Telegram Stars.

- ***suggested_tip_amounts*** (Sequence[`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*.

  Changed in version 20.0:

  – This attribute is now an immutable tuple.

  – This attribute is now always a tuple, that may be empty.

- ***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. Ignored for payments in Telegram Stars.

- ***need_phone_number*** (`bool`, optional) – Pass `True`, if you require the user's phone number to complete the order. Ignored for payments in Telegram Stars.

- ***need_email*** (`bool`, optional) – Pass `True`, if you require the user's email address to complete the order. Ignored for payments in Telegram Stars.

- ***need_shipping_address*** (`bool`, optional) – Pass `True`, if you require the user's shipping address to complete the order. Ignored for payments in Telegram Stars

- ***send_phone_number_to_provider*** (`bool`, optional) – Pass `True`, if user's phone number should be sent to provider. Ignored for payments in Telegram Stars.

- ***send_email_to_provider*** (`bool`, optional) – Pass `True`, if user's email address should be sent to provider. Ignored for payments in Telegram Stars.

- ***is_flexible*** (`bool`, optional) – Pass `True`, if the final price depends on the shipping method. Ignored for payments in Telegram Stars.

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 it for your internal processes.

Type
str

provider_token

Payment provider token, obtained via @Botfather. Pass an empty string for payments in Telegram Stars.

Type
str

currency

Three-letter ISO 4217 currency code, see more on currencies. Pass `XTR` for payments in Telegram Stars.

Type
str

prices

Price breakdown, a list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.). Must contain exactly one item for payments in Telegram Stars.

Changed in version 20.0: This attribute is now an immutable tuple.

Type
tuple[*telegram.LabeledPrice*]

max_tip_amount

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` `max_tip_amount` is `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`. Not supported for payments in Telegram Stars.

Type
int

suggested_tip_amounts

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*.

Changed in version 20.0: This attribute is now an immutable tuple.

Type
tuple[int]

provider_data

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.

Type
str

photo_url

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.

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. Ignored for payments in Telegram Stars.

Type
bool

need_phone_number

Optional. Pass True, if you require the user's phone number to complete the order. Ignored for payments in Telegram Stars.

Type
bool

need_email

Optional. Pass True, if you require the user's email address to complete the order. Ignored for payments in Telegram Stars.

Type
bool

need_shipping_address

Optional. Pass True, if you require the user's shipping address to complete the order. Ignored for payments in Telegram Stars.

Type
bool

send_phone_number_to_provider

Optional. Pass True, if user's phone number should be sent to provider. Ignored for payments in Telegram Stars.

Type
bool

send_email_to_provider

Optional. Pass True, if user's email address should be sent to provider. Ignored for payments in Telegram Stars.

Type
bool

is_flexible

Optional. Pass `True`, if the final price depends on the shipping method. Ignored for payments in Telegram Stars.

Type
`bool`

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

PreparedInlineMessage

class telegram.**PreparedInlineMessage**(*id*, *expiration_date*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes an inline message to be sent by a user of a Mini App.

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

Returned In

telegram.Bot.save_prepared_inline_message()

Added in version 21.8.

Parameters

- ***id*** (`str`) – Unique identifier of the prepared message
- ***expiration_date*** (`datetime.datetime`) – Expiration date of the prepared message. Expired prepared messages can no longer be used. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

id

Unique identifier of the prepared message

Type
`str`

expiration_date

Expiration date of the prepared message. Expired prepared messages can no longer be used. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
`datetime.datetime`

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

Payments

Your bot can accept payments from Telegram users. Please see the introduction to payments for more details on the process and how to set up payments for your bot.

AffiliateInfo

class telegram.**AffiliateInfo**(*commission_per_mille*, *amount*, *affiliate_user=None*, *affiliate_chat=None*, *nanostar_amount=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Contains information about the affiliate that received a commission via this transaction.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their *affiliate_user*, *affiliate_chat*, *commission_per_mille*, *amount*, and *nanostar_amount* are equal.

Available In

telegram.TransactionPartnerUser.affiliate

Added in version 21.9.

Parameters

- ***affiliate_user*** (*telegram.User*, optional) – The bot or the user that received an affiliate commission if it was received by a bot or a user
- ***affiliate_chat*** (*telegram.Chat*, optional) – The chat that received an affiliate commission if it was received by a chat
- ***commission_per_mille*** (int) – The number of Telegram Stars received by the affiliate for each 1000 Telegram Stars received by the bot from referred users
- ***amount*** (int) – Integer amount of Telegram Stars received by the affiliate from the transaction, rounded to 0; can be negative for refunds
- ***nanostar_amount*** (int, optional) – The number of *1e-09* shares of Telegram Stars received by the affiliate; from *-999999999* to *999999999*; can be negative for refunds

affiliate_user

Optional. The bot or the user that received an affiliate commission if it was received by a bot or a user

Type
telegram.User

affiliate_chat

Optional. The chat that received an affiliate commission if it was received by a chat

Type
telegram.Chat

commission_per_mille

The number of Telegram Stars received by the affiliate for each 1000 Telegram Stars received by the bot from referred users

Type
int

amount

Integer amount of Telegram Stars received by the affiliate from the transaction, rounded to 0; can be negative for refunds

Type
int

nanostar_amount

Optional. The number of *1e-09* shares of Telegram Stars received by the affiliate; from *-999999999* to *999999999*; can be negative for refunds

Type
int

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

Invoice

class telegram.**Invoice**(*title*, *description*, *start_parameter*, *currency*, *total_amount*, *, *api_kwargs=None*)

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.

Available In

- *telegram.ExternalReplyInfo.invoice*
- *telegram.Message.effective_attachment*
- *telegram.Message.invoice*

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, or XTR for payments in Telegram Stars.
- ***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).

title

Product name.

Type
str

description

Product description.

Type
str

start_parameter

Unique bot deep-linking parameter that can be used to generate this invoice.

Type
str

currency

Three-letter ISO 4217 currency code, or XTR for payments in Telegram Stars.

Type
str

total_amount

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).

Type
int

MAX_DESCRIPTION_LENGTH = 255

telegram.constants.InvoiceLimit.MAX_DESCRIPTION_LENGTH

Added in version 20.0.

MAX_PAYLOAD_LENGTH = 128

telegram.constants.InvoiceLimit.MAX_PAYLOAD_LENGTH

Added in version 20.0.

MAX_TIP_AMOUNTS = 4

telegram.constants.InvoiceLimit.MAX_TIP_AMOUNTS

Added in version 20.0.

MAX_TITLE_LENGTH = 32

telegram.constants.InvoiceLimit.MAX_TITLE_LENGTH

Added in version 20.0.

MIN_DESCRIPTION_LENGTH = 1

telegram.constants.InvoiceLimit.MIN_DESCRIPTION_LENGTH

Added in version 20.0.

MIN_PAYLOAD_LENGTH = 1

telegram.constants.InvoiceLimit.MIN_PAYLOAD_LENGTH

Added in version 20.0.

MIN_TITLE_LENGTH = 1

telegram.constants.InvoiceLimit.MIN_TITLE_LENGTH

Added in version 20.0.

LabeledPrice

class telegram.**LabeledPrice**(*label*, *amount*, *, *api_kwargs=None*)

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.

Examples

Payment Bot

Use In

- *telegram.Bot.create_invoice_link()*
- *telegram.Bot.send_invoice()*

Available In

- *telegram.InputInvoiceMessageContent.prices*
- *telegram.ShippingOption.prices*

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).

label

Portion label.

Type
str

amount

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).

Type
int

OrderInfo

class telegram.**OrderInfo**(*name=None*, *phone_number=None*, *email=None*, *shipping_address=None*, ***, *api_kwargs=None*)

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.

Available In

- *telegram.PreCheckoutQuery.order_info*
- *telegram.SuccessfulPayment.order_info*

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.

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=None*)

See *telegram.TelegramObject.de_json()*.

PreCheckoutQuery

class telegram.**PreCheckoutQuery**(*id*, *from_user*, *currency*, *total_amount*, *invoice_payload*, *shipping_option_id=None*, *order_info=None*, ***, *api_kwargs=None*)

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.

Available In

telegram.Update.pre_checkout_query

Parameters

- ***id*** (`str`) – Unique query identifier.
- ***from_user*** (*telegram.User*) – User who sent the query.
- ***currency*** (`str`) – Three-letter ISO 4217 currency code, or `XTR` for payments in Telegram Stars.
- ***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.

id

Unique query identifier.

Type
str

from_user

User who sent the query.

Type
telegram.User

currency

Three-letter ISO 4217 currency code, or `XTR` for payments in Telegram Stars.

Type
str

total_amount

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).

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

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=None*)

See *telegram.TelegramObject.de_json()*.

RefundedPayment

class telegram.**RefundedPayment**(*currency*, *total_amount*, *invoice_payload*, *telegram_payment_charge_id*, *provider_payment_charge_id=None*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object contains basic information about a refunded 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* is equal.

Available In

telegram.Message.refunded_payment

Added in version 21.4.

Parameters

- ***currency*** (str) – Three-letter ISO 4217 currency code, or XTR for payments in Telegram Stars. Currently, always XTR.
- ***total_amount*** (int) – Total refunded price in the *smallest units* of the currency (integer, **not** float/double). For example, for a price of US$ 1.45, total_amount = 145. See the *exp* parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- ***invoice_payload*** (str) – Bot-specified invoice payload.
- ***telegram_payment_charge_id*** (str) – Telegram payment identifier.
- ***provider_payment_charge_id*** (str, optional) – Provider payment identifier.

currency

Three-letter ISO 4217 currency code, or XTR for payments in Telegram Stars. Currently, always XTR.

Type
str

total_amount

Total refunded price in the *smallest units* of the currency (integer, **not** float/double). For example, for a price of US$ 1.45, total_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).

Type
int

invoice_payload

Bot-specified invoice payload.

Type
str

telegram_payment_charge_id

Telegram payment identifier.

Type
str

provider_payment_charge_id

Optional. Provider payment identifier.

Type
str

RevenueWithdrawalState

class telegram.**RevenueWithdrawalState**(*type*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object describes the state of a revenue withdrawal operation. Currently, it can be one of:

- *telegram.RevenueWithdrawalStatePending*
- *telegram.RevenueWithdrawalStateSucceeded*
- *telegram.RevenueWithdrawalStateFailed*

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

Available In

telegram.TransactionPartnerFragment.withdrawal_state

Added in version 21.4.

Parameters

type (str) – The type of the state.

type

The type of the state.

Type

str

FAILED = 'failed'

telegram.constants.RevenueWithdrawalStateType.FAILED

PENDING = 'pending'

telegram.constants.RevenueWithdrawalStateType.PENDING

SUCCEEDED = 'succeeded'

telegram.constants.RevenueWithdrawalStateType.SUCCEEDED

classmethod de_json(*data*, *bot=None*)

Converts JSON data to the appropriate *RevenueWithdrawalState* 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.

RevenueWithdrawalStateFailed

class telegram.**RevenueWithdrawalStateFailed**(***, *api_kwargs=None*)

Bases: *telegram.RevenueWithdrawalState*

The withdrawal failed and the transaction was refunded.

Available In

telegram.TransactionPartnerFragment.withdrawal_state

Added in version 21.4.

type

The type of the state, always *'failed'*.

Type

str

RevenueWithdrawalStatePending

class telegram.**RevenueWithdrawalStatePending**(*, *api_kwargs=None*)

Bases: *telegram.RevenueWithdrawalState*

The withdrawal is in progress.

Available In

telegram.TransactionPartnerFragment.withdrawal_state

Added in version 21.4.

type

The type of the state, always *'pending'*.

Type

str

RevenueWithdrawalStateSucceeded

class telegram.**RevenueWithdrawalStateSucceeded**(*date*, *url*, *, *api_kwargs=None*)

Bases: *telegram.RevenueWithdrawalState*

The withdrawal succeeded.

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

Available In

telegram.TransactionPartnerFragment.withdrawal_state

Added in version 21.4.

Parameters

- ***date*** (datetime.datetime) – Date the withdrawal was completed as a datetime object.
- ***url*** (str) – An HTTPS URL that can be used to see transaction details.

type

The type of the state, always *'succeeded'*.

Type

str

date

Date the withdrawal was completed as a datetime object.

Type
datetime.datetime

url

An HTTPS URL that can be used to see transaction details.

Type
str

classmethod de_json(*data*, *bot=None*)

See *telegram.RevenueWithdrawalState.de_json()*.

ShippingAddress

class telegram.**ShippingAddress**(*country_code*, *state*, *city*, *street_line1*, *street_line2*, *post_code*, ***, *api_kwargs=None*)

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.

Available In

- *telegram.OrderInfo.shipping_address*
- *telegram.ShippingQuery.shipping_address*

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.

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

ShippingOption

class telegram.**ShippingOption**(*id*, *title*, *prices*, *, *api_kwargs=None*)

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.

Examples

Payment Bot

Use In

telegram.Bot.answer_shipping_query()

Parameters

- ***id*** (str) – Shipping option identifier.
- ***title*** (str) – Option title.
- ***prices*** (Sequence[*telegram.LabeledPrice*]) – List of price portions.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.

id

Shipping option identifier.

Type
str

title

Option title.

Type
str

prices

List of price portions.

Changed in version 20.0: This attribute is now an immutable tuple.

Type
tuple[*telegram.LabeledPrice*]

ShippingQuery

class telegram.**ShippingQuery**(*id*, *from_user*, *invoice_payload*, *shipping_address*, *, *api_kwargs=None*)

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.

Available In

telegram.Update.shipping_query

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.

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

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=None*)

See *telegram.TelegramObject.de_json()*.

StarAmount

class telegram.**StarAmount**(*amount*, *nanostar_amount=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes an amount of Telegram Stars.

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

Available In

telegram.SuggestedPostPaid.star_amount

Returned In

- *telegram.Bot.get_business_account_star_balance()*
- *telegram.Bot.get_my_star_balance()*

Parameters

- ***amount*** (int) – Integer amount of Telegram Stars, rounded to 0; can be negative.
- ***nanostar_amount*** (int, optional) – The number of *1e-09* shares of Telegram Stars; from *-999999999* to *999999999*; can be negative if and only if *amount* is non-positive.

amount

Integer amount of Telegram Stars, rounded to 0; can be negative.

Type
int

nanostar_amount

Optional. The number of *1e-09* shares of Telegram Stars; from *-999999999* to *999999999*; can be negative if and only if *amount* is non-positive.

Type
int

StarTransaction

class telegram.**StarTransaction**(*id*, *amount*, *date*, *source=None*, *receiver=None*, *nanostar_amount=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Describes a Telegram Star transaction. Note that if the buyer initiates a chargeback with the payment provider from whom they acquired Stars (e.g., Apple, Google) following this transaction, the refunded Stars will be deducted from the bot's balance. This is outside of Telegram's control.

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

Available In

telegram.StarTransactions.transactions

Added in version 21.4.

Parameters

- ***id*** (str) – Unique identifier of the transaction. Coincides with the identifer of the original transaction for refund transactions. Coincides with *SuccessfulPayment.telegram_payment_charge_id* for successful incoming payments from users.
- ***amount*** (int) – Integer amount of Telegram Stars transferred by the transaction.
- ***nanostar_amount*** (int, optional) – The number of *1e-09* shares of Telegram Stars transferred by the transaction; from 0 to *999999999*

  Added in version 21.9.
- ***date*** (datetime.datetime) – Date the transaction was created as a datetime object.
- ***source*** (*telegram.TransactionPartner*, optional) – Source of an incoming transaction (e.g., a user purchasing goods or services, Fragment refunding a failed withdrawal). Only for incoming transactions.
- ***receiver*** (*telegram.TransactionPartner*, optional) – Receiver of an outgoing transaction (e.g., a user for a purchase refund, Fragment for a withdrawal). Only for outgoing transactions.

id

Unique identifier of the transaction. Coincides with the identifer of the original transaction for refund transactions. Coincides with *SuccessfulPayment.telegram_payment_charge_id* for successful incoming payments from users.

Type
str

amount

Integer amount of Telegram Stars transferred by the transaction.

Type
int

nanostar_amount

Optional. The number of *1e-09* shares of Telegram Stars transferred by the transaction; from 0 to *999999999*

Added in version 21.9.

Type
int

date

Date the transaction was created as a datetime object.

Type
datetime.datetime

source

Optional. Source of an incoming transaction (e.g., a user purchasing goods or services, Fragment refunding a failed withdrawal). Only for incoming transactions.

Type
telegram.TransactionPartner

receiver

Optional. Receiver of an outgoing transaction (e.g., a user for a purchase refund, Fragment for a withdrawal). Only for outgoing transactions.

Type
telegram.TransactionPartner

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

StarTransactions

class telegram.**StarTransactions**(*transactions*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

Contains a list of Telegram Star transactions.

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

Returned In

telegram.Bot.get_star_transactions()

Added in version 21.4.

Parameters
transactions (Sequence[*telegram.StarTransaction*]) – The list of transactions.

transactions

The list of transactions.

Type
tuple[*telegram.StarTransaction*]

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

SuccessfulPayment

class telegram.**SuccessfulPayment**(*currency*, *total_amount*, *invoice_payload*, *telegram_payment_charge_id*, *provider_payment_charge_id*, *shipping_option_id=None*, *order_info=None*, *subscription_expiration_date=None*, *is_recurring=None*, *is_first_recurring=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object contains basic information about a successful payment. Note that if the buyer initiates a chargeback with the relevant payment provider following this transaction, the funds may be debited from your balance. This is outside of Telegram's control.

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.

Available In

- *telegram.Message.effective_attachment*
- *telegram.Message.successful_payment*

Parameters

- ***currency*** (str) – Three-letter ISO 4217 currency code, or XTR for payments in Telegram Stars.
- ***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.
- ***subscription_expiration_date*** (datetime.datetime, optional) – Expiration date of the subscription; for recurring payments only.

  Added in version 21.8.
- ***is_recurring*** (bool, optional) – True, if the payment is for a subscription.

  Added in version 21.8.
- ***is_first_recurring*** (bool, optional) – True, if the payment is the first payment of a subscription.

  Added in version 21.8.
- ***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.

currency

Three-letter ISO 4217 currency code, or XTR for payments in Telegram Stars.

Type
str

total_amount

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).

Type
int

invoice_payload

Bot-specified invoice payload.

Type
str

subscription_expiration_date

Optional. Expiration date of the subscription; for recurring payments only.

Added in version 21.8.

Type
datetime.datetime

is_recurring

Optional. True, if the payment is for a subscription.

Added in version 21.8.

Type
bool

is_first_recurring

Optional. True, if the payment is the first payment of a subscription.

Added in version 21.8.

Type
bool

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=None*)

See *telegram.TelegramObject.de_json()*.

TransactionPartner

class telegram.**TransactionPartner**(*type*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object describes the source of a transaction, or its recipient for outgoing transactions. Currently, it can be one of:

- *TransactionPartnerUser*
- *TransactionPartnerChat*
- *TransactionPartnerAffiliateProgram*
- *TransactionPartnerFragment*
- *TransactionPartnerTelegramAds*
- *TransactionPartnerTelegramApi*
- *TransactionPartnerOther*

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

Available In

- *telegram.StarTransaction.receiver*
- *telegram.StarTransaction.source*

Added in version 21.4.

Changed in version 21.11: Added *TransactionPartnerChat*

Parameters

type (str) – The type of the transaction partner.

type

The type of the transaction partner.

Type

str

AFFILIATE_PROGRAM = 'affiliate_program'

telegram.constants.TransactionPartnerType.AFFILIATE_PROGRAM

Added in version 21.9.

CHAT = 'chat'

telegram.constants.TransactionPartnerType.CHAT

Added in version 21.11.

FRAGMENT = 'fragment'

telegram.constants.TransactionPartnerType.FRAGMENT

OTHER = 'other'

telegram.constants.TransactionPartnerType.OTHER

TELEGRAM_ADS = 'telegram_ads'

telegram.constants.TransactionPartnerType.TELEGRAM_ADS

TELEGRAM_API = 'telegram_api'

telegram.constants.TransactionPartnerType.TELEGRAM_API

USER = 'user'

telegram.constants.TransactionPartnerType.USER

classmethod de_json(*data*, *bot=None*)

Converts JSON data to the appropriate *TransactionPartner* 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.

TransactionPartnerAffiliateProgram

class telegram.**TransactionPartnerAffiliateProgram**(*commission_per_mille*, *sponsor_user=None*, ***, *api_kwargs=None*)

Bases: *telegram.TransactionPartner*

Describes the affiliate program that issued the affiliate commission received via this transaction.

This object is comparable in terms of equality. Two objects of this class are considered equal, if their *commission_per_mille* are equal.

Available In

- *telegram.StarTransaction.receiver*
- *telegram.StarTransaction.source*

Added in version 21.9.

Parameters

- ***sponsor_user*** (*telegram.User*, optional) – Information about the bot that sponsored the affiliate program
- ***commission_per_mille*** (int) – The number of Telegram Stars received by the bot for each 1000 Telegram Stars received by the affiliate program sponsor from referred users.

type

The type of the transaction partner, always *'affiliate_program'*.

Type
str

sponsor_user

Optional. Information about the bot that sponsored the affiliate program

Type
telegram.User

commission_per_mille

The number of Telegram Stars received by the bot for each 1000 Telegram Stars received by the affiliate program sponsor from referred users.

Type
int

classmethod de_json(*data*, *bot=None*)

See *telegram.TransactionPartner.de_json()*.

TransactionPartnerChat

class telegram.**TransactionPartnerChat**(*chat*, *gift=None*, ***, *api_kwargs=None*)

Bases: *telegram.TransactionPartner*

Describes a transaction with a chat.

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

Available In

- *telegram.StarTransaction.receiver*

- *telegram.StarTransaction.source*

Added in version 21.11.

Parameters

- ***chat*** (*telegram.Chat*) – Information about the chat.
- ***gift*** (*telegram.Gift*, optional) – The gift sent to the chat by the bot.

type

The type of the transaction partner, always *'chat'*.

Type
str

chat

Information about the chat.

Type
telegram.Chat

gift

Optional. The gift sent to the user by the bot.

Type
telegram.Gift

classmethod de_json(*data*, *bot=None*)

See *telegram.TransactionPartner.de_json()*.

TransactionPartnerFragment

class telegram.**TransactionPartnerFragment**(*withdrawal_state=None*, ***, *api_kwargs=None*)

Bases: *telegram.TransactionPartner*

Describes a withdrawal transaction with Fragment.

Available In

- *telegram.StarTransaction.receiver*
- *telegram.StarTransaction.source*

Added in version 21.4.

Parameters

withdrawal_state (*telegram.RevenueWithdrawalState*, optional) – State of the transaction if the transaction is outgoing.

type

The type of the transaction partner, always *'fragment'*.

Type
str

withdrawal_state

Optional. State of the transaction if the transaction is outgoing.

Type
telegram.RevenueWithdrawalState

classmethod de_json(*data*, *bot=None*)

See *telegram.TransactionPartner.de_json()*.

TransactionPartnerOther

class telegram.**TransactionPartnerOther**(*, *api_kwargs=None*)

Bases: *telegram.TransactionPartner*

Describes a transaction with an unknown partner.

Available In

- *telegram.StarTransaction.receiver*
- *telegram.StarTransaction.source*

Added in version 21.4.

type

The type of the transaction partner, always *'other'*.

Type

str

TransactionPartnerTelegramAds

class telegram.**TransactionPartnerTelegramAds**(*, *api_kwargs=None*)

Bases: *telegram.TransactionPartner*

Describes a withdrawal transaction to the Telegram Ads platform.

Available In

- *telegram.StarTransaction.receiver*
- *telegram.StarTransaction.source*

Added in version 21.4.

type

The type of the transaction partner, always *'telegram_ads'*.

Type

str

TransactionPartnerTelegramApi

class telegram.**TransactionPartnerTelegramApi**(*request_count*, *, *api_kwargs=None*)

Bases: *telegram.TransactionPartner*

Describes a transaction with payment for paid broadcasting.

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

Available In

- *telegram.StarTransaction.receiver*

- *telegram.StarTransaction.source*

Added in version 21.7.

Parameters

request_count (int) – The number of successful requests that exceeded regular limits and were therefore billed.

type

The type of the transaction partner, always *'telegram_api'*.

Type

str

request_count

The number of successful requests that exceeded regular limits and were therefore billed.

Type

int

TransactionPartnerUser

class telegram.**TransactionPartnerUser**(*transaction_type*, *user*, *invoice_payload=None*, *paid_media=None*, *paid_media_payload=None*, *subscription_period=None*, *gift=None*, *affiliate=None*, *premium_subscription_duration=None*, *, *api_kwargs=None*)

Bases: *telegram.TransactionPartner*

Describes a transaction with a user.

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

Available In

- *telegram.StarTransaction.receiver*
- *telegram.StarTransaction.source*

Added in version 21.4.

Changed in version 22.1: Equality comparison now includes the new required argument *transaction_type*, introduced in Bot API 9.0.

Parameters

- ***transaction_type*** (str) – Type of the transaction, currently one of *'invoice_payment'* for payments via invoices, *'paid_media_payment'* for payments for paid media, *'gift_purchase'* for gifts sent by the bot, *'premium_purchase'* for Telegram Premium subscriptions gifted by the bot, *'business_account_transfer'* for direct transfers from managed business accounts.

  Added in version 22.1.

- ***user*** (*telegram.User*) – Information about the user.

- ***affiliate*** (*telegram.AffiliateInfo*, optional) – Information about the affiliate that received a commission via this transaction. Can be available only for *'invoice_payment'* and *'paid_media_payment'* transactions.

  Added in version 21.9.

- ***invoice_payload*** (str, optional) – Bot-specified invoice payload. Can be available only for *'invoice_payment'* transactions.
- ***subscription_period*** (int | datetime.timedelta, optional) – The duration of the paid subscription. Can be available only for *'invoice_payment'* transactions.

  Added in version 21.8.

  Changed in version v22.2: Accepts int objects as well as datetime.timedelta.
- ***paid_media*** (Sequence[*telegram.PaidMedia*], optional) – Information about the paid media bought by the user. for *'paid_media_payment'* transactions only.

  Added in version 21.5.
- ***paid_media_payload*** (str, optional) – Bot-specified paid media payload. Can be available only for *'paid_media_payment'* transactions.

  Added in version 21.6.
- ***gift*** (*telegram.Gift*, optional) – The gift sent to the user by the bot; for *'gift_purchase'* transactions only.

  Added in version 21.8.
- ***premium_subscription_duration*** (int, optional) – Number of months the gifted Telegram Premium subscription will be active for; for *'premium_purchase'* transactions only.

  Added in version 22.1.

type

The type of the transaction partner, always *'user'*.

Type
str

transaction_type

Type of the transaction, currently one of *'invoice_payment'* for payments via invoices, *'paid_media_payment'* for payments for paid media, *'gift_purchase'* for gifts sent by the bot, *'premium_purchase'* for Telegram Premium subscriptions gifted by the bot, *'business_account_transfer'* for direct transfers from managed business accounts.

Added in version 22.1.

Type
str

user

Information about the user.

Type
telegram.User

affiliate

Optional. Information about the affiliate that received a commission via this transaction. Can be available only for *'invoice_payment'* and *'paid_media_payment'* transactions.

Added in version 21.9.

Type
telegram.AffiliateInfo

invoice_payload

Optional. Bot-specified invoice payload. Can be available only for *'invoice_payment'* transactions.

Type
str

subscription_period

Optional. The duration of the paid subscription. Can be available only for *'invoice_payment'* transactions.

Added in version 21.8.

Type
datetime.timedelta

paid_media

Optional. Information about the paid media bought by the user. for *'paid_media_payment'* transactions only.

Added in version 21.5.

Type
tuple[*telegram.PaidMedia*]

paid_media_payload

Optional. Bot-specified paid media payload. Can be available only for *'paid_media_payment'* transactions.

Added in version 21.6.

Type
str

gift

Optional. The gift sent to the user by the bot; for *'gift_purchase'* transactions only.

Added in version 21.8.

Type
telegram.Gift

premium_subscription_duration

Optional. Number of months the gifted Telegram Premium subscription will be active for; for *'premium_purchase'* transactions only.

Added in version 22.1.

Type
int

classmethod de_json(*data*, *bot=None*)

See *telegram.TransactionPartner.de_json()*.

Games

Your bot can offer users **HTML5 games** to play solo or to compete against each other in groups and one-on-one chats. Create games via @BotFather using the `/newgame` command. Please note that this kind of power requires responsibility: you will need to accept the terms for each game that your bots will be offering.

- Games are a new type of content on Telegram, represented by the *telegram.Game* and *telegram.InlineQueryResultGame* objects.
- Once you've created a game via BotFather, you can send games to chats as regular messages using the *sendGame()* method, or use *inline mode* with *telegram.InlineQueryResultGame*.
- If you send the game message without any buttons, it will automatically have a 'Play `GameName`' button. When this button is pressed, your bot gets a *telegram.CallbackQuery* with the `game_short_name` of the requested game. You provide the correct URL for this particular user and the app opens the game in the in-app browser.
- You can manually add multiple buttons to your game message. Please note that the first button in the first row **must always** launch the game, using the field `callback_game` in *telegram.InlineKeyboardButton*.

You can add extra buttons according to taste: e.g., for a description of the rules, or to open the game's official community.

- To make your game more attractive, you can upload a GIF animation that demonstrates the game to the users via BotFather (see Lumberjack for example).
- A game message will also display high scores for the current chat. Use *setGameScore()* to post high scores to the chat with the game, add the *disable_edit_message* parameter to disable automatic update of the message with the current scoreboard.
- Use *getGameHighScores()* to get data for in-game high score tables.
- You can also add an extra sharing button for users to share their best score to different chats.
- For examples of what can be done using this new stuff, check the @gamebot and @gamee bots.

Callbackgame

class telegram.**CallbackGame**(*, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

A placeholder, currently holds no information. Use BotFather to set up your game.

Available In

telegram.InlineKeyboardButton.callback_game

Game

class telegram.**Game**(*title*, *description*, *photo*, *text=None*, *text_entities=None*, *animation=None*, *, *api_kwargs=None*)

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.

Available In

- *telegram.ExternalReplyInfo.game*
- *telegram.Message.effective_attachment*
- *telegram.Message.game*

Parameters

- ***title*** (str) – Title of the game.
- ***description*** (str) – Description of the game.
- ***photo*** (Sequence[*telegram.PhotoSize*]) – Photo that will be displayed in the game message in chats.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.
- ***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*** (Sequence[*telegram.MessageEntity*], optional) – Special entities that appear in text, such as usernames, URLs, bot commands, etc.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list. The input is converted to a tuple.

- ***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.

Changed in version 20.0: This attribute is now an immutable tuple.

Type
tuple[*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()*. 0-*4096* characters.

Type
`str`

text_entities

Optional. Special entities that appear in text, such as usernames, URLs, bot commands, etc. This tuple is empty if the message does not contain text entities.

Changed in version 20.0: This attribute is now an immutable tuple.

Type
tuple[*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=None*)

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.

GameHighScore

class telegram.**GameHighScore**(*position*, *user*, *score*, *, *api_kwargs=None*)

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.

> **Returned In**
>
> *telegram.Bot.get_game_high_scores()*

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=None*)

See *telegram.TelegramObject.de_json()*.

Passport

Passport is a unified authorization method for services that require personal identification. Users can upload their documents once, then instantly share their data with services that require real-world ID (finance, ICOs, etc.). Please see the manual for details.

Credentials

class telegram.**Credentials**(*secure_data*, *nonce*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

secure_data

Credentials for encrypted data

Type
telegram.SecureData

nonce

Bot-specified nonce

Type
str

Available In

- *telegram.EncryptedCredentials.decrypted_data*
- *telegram.PassportData.decrypted_credentials*

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

DataCredentials

class telegram.**DataCredentials**(*data_hash*, *secret*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

These credentials can be used to decrypt encrypted data from the data field in EncryptedPassportData.

Available In

telegram.SecureValue.data

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

EncryptedCredentials

class telegram.**EncryptedCredentials**(*data*, *hash*, *secret*, *, *api_kwargs=None*)

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*.

Available In

telegram.PassportData.credentials

Parameters

- ***data*** (*telegram.Credentials* | 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.

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 | 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
bytes

EncryptedPassportElement

class telegram.**EncryptedPassportElement**(*type*, *hash*, *data=None*, *phone_number=None*, *email=None*, *files=None*, *front_side=None*, *reverse_side=None*, *selfie=None*, *translation=None*, ***, *api_kwargs=None*)

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*.

Available In

- *telegram.PassportData.data*
- *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 only for “personal_details”, “passport”, “driver_license”, “identity_card”, “internal_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*** (Sequence[*telegram.PassportFile*], optional) – Array of encrypted/decrypted files with documents provided by the user; available only for “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.
- ***front_side*** (*telegram.PassportFile*, optional) – Encrypted/decrypted file with the front side of the document, provided by the user; Available only 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 only 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 if requested for “passport”, “driver_license”, “identity_card” and “internal_passport”.
- ***translation*** (Sequence[*telegram.PassportFile*], optional) – Array of encrypted/decrypted files with translated versions of documents provided by the user; available if requested requested for “passport”, “driver_license”, “identity_card”, “internal_passport”, “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.

  Changed in version 20.0: Accepts any collections.abc.Sequence as input instead of just a list. The input is converted to a tuple.

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 only for “personal_details”, “passport”, “driver_license”, “identity_card”, “internal_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 only for "utility_bill", "bank_statement", "rental_agreement", "passport_registration" and "temporary_registration" types.

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*telegram.PassportFile*]

front_side

Optional. Encrypted/decrypted file with the front side of the document, provided by the user; available only 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 only 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 if requested 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.

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*telegram.PassportFile*]

classmethod de_json(*data*, *bot=None*)

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* | None) – The bot associated with these object. May be None, in which case shortcut methods will not be available.

  Changed in version 21.4: *bot* is now optional and defaults to None

  Deprecated since version 21.4: This argument will be converted to an optional argument in future versions.
- ***credentials*** (*telegram.FileCredentials*) – The credentials

Return type

telegram.EncryptedPassportElement

FileCredentials

class telegram.**FileCredentials**(*file_hash*, *secret*, ***, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

These credentials can be used to decrypt encrypted files from the front_side, reverse_side, selfie and files fields in EncryptedPassportData.

Available In

- *telegram.SecureValue.files*
- *telegram.SecureValue.front_side*
- *telegram.SecureValue.reverse_side*
- *telegram.SecureValue.selfie*
- *telegram.SecureValue.translation*

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

IdDocumentData

class telegram.**IdDocumentData**(*document_no*, *expiry_date*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents the data of an identity document.

Available In

telegram.EncryptedPassportElement.data

Parameters

- ***document_no*** (str) – Document number.
- ***expiry_date*** (str) – Optional. Date of expiry, in DD.MM.YYYY format.

document_no

Document number.

Type

str

expiry_date

Optional. Date of expiry, in DD.MM.YYYY format.

Type

str

PassportData

class telegram.**PassportData**(*data*, *credentials*, *, *api_kwargs=None*)

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*.

Available In

- *telegram.Message.effective_attachment*
- *telegram.Message.passport_data*

Parameters

- ***data*** (Sequence[*telegram.EncryptedPassportElement*]) – Array with encrypted information about documents and other Telegram Passport elements that was shared with the bot.

  Changed in version 20.0: Accepts any `collections.abc.Sequence` as input instead of just a list. The input is converted to a tuple.

- ***credentials*** (*telegram.EncryptedCredentials*)) – Encrypted credentials.

data

Array with encrypted information about documents and other Telegram Passport elements that was shared with the bot.

Changed in version 20.0: This attribute is now an immutable tuple.

Type
tuple[*telegram.EncryptedPassportElement*]

credentials

Encrypted credentials.

Type
telegram.EncryptedCredentials

classmethod de_json(*data*, *bot=None*)

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.

Changed in version 20.0: Returns a tuple instead of a list.

Raises
telegram.error.PassportDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type
tuple[*telegram.EncryptedPassportElement*]

PassportElementError

class telegram.**PassportElementError**(*source*, *type*, *message*, ***, *api_kwargs=None*)

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.

Use In

telegram.Bot.set_passport_data_errors()

Parameters

- ***source*** (str) – Error source.

- ***type*** (str) – The section of the user's Telegram Passport which has the error.
- ***message*** (str) – Error message.

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

PassportElementErrorDataField

class telegram.**PassportElementErrorDataField**(*type*, *field_name*, *data_hash*, *message*, *, *api_kwargs=None*)

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.

Use In

telegram.Bot.set_passport_data_errors()

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.

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

PassportElementErrorFile

class telegram.**PassportElementErrorFile**(*type*, *file_hash*, *message*, *, *api_kwargs=None*)

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.

Use In

telegram.Bot.set_passport_data_errors()

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.

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

PassportElementErrorFiles

class telegram.**PassportElementErrorFiles**(*type*, *file_hashes*, *message*, *, *api_kwargs=None*)

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.

Use In

telegram.Bot.set_passport_data_errors()

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*** (Sequence[str]) – List of base64-encoded file hashes.

  Changed in version 22.0: Accepts any collections.abc.Sequence as input instead of just a list.
- ***message*** (str) – Error message.

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.

Changed in version 22.0: This attribute is now an immutable tuple.

Type
tuple[str]

message

Error message.

Type
str

PassportElementErrorFrontSide

class telegram.**PassportElementErrorFrontSide**(*type*, *file_hash*, *message*, *, *api_kwargs=None*)

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.

Use In

telegram.Bot.set_passport_data_errors()

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.

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

PassportElementErrorReverseSide

class telegram.**PassportElementErrorReverseSide**(*type*, *file_hash*, *message*, *, *api_kwargs=None*)

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.

Use In

telegram.Bot.set_passport_data_errors()

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.

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

PassportElementErrorSelfie

class telegram.**PassportElementErrorSelfie**(*type*, *file_hash*, *message*, *, *api_kwargs=None*)

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.

Use In

telegram.Bot.set_passport_data_errors()

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.

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

PassportElementErrorTranslationFile

class telegram.**PassportElementErrorTranslationFile**(*type*, *file_hash*, *message*, ***, *api_kwargs=None*)

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.

Use In

telegram.Bot.set_passport_data_errors()

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.

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

PassportElementErrorTranslationFiles

class telegram.**PassportElementErrorTranslationFiles**(*type*, *file_hashes*, *message*, ***, *api_kwargs=None*)

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.

Use In

telegram.Bot.set_passport_data_errors()

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*** (Sequence[str]) – List of base64-encoded file hashes.

  Changed in version 22.0: Accepts any collections.abc.Sequence as input instead of just a list.
- ***message*** (str) – Error message.

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.

Changed in version 22.0: This attribute is now an immutable tuple.

Type
tuple[str]

message

Error message.

Type
str

PassportElementErrorUnspecified

class telegram.**PassportElementErrorUnspecified**(*type*, *element_hash*, *message*, *, *api_kwargs=None*)

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.

Use In

telegram.Bot.set_passport_data_errors()

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.

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

PassportFile

class telegram.**PassportFile**(*file_id*, *file_unique_id*, *file_date*, *file_size*, *credentials=None*, ***, *api_kwargs=None*)

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.

Available In

- *telegram.EncryptedPassportElement.files*
- *telegram.EncryptedPassportElement.front_side*
- *telegram.EncryptedPassportElement.reverse_side*
- *telegram.EncryptedPassportElement.selfie*
- *telegram.EncryptedPassportElement.translation*

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*** (datetime.datetime) – Time when the file was uploaded.

  Changed in version 22.0: Accepts only datetime.datetime instead of int. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

file_id

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

Time when the file was uploaded.

Changed in version 22.0: Returns datetime.datetime instead of int. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

Type
datetime.datetime

classmethod de_json(*data*, *bot=None*)

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* | None) – The bot associated with these object. May be None, in which case shortcut methods will not be available.

  Changed in version 21.4: *bot* is now optional and defaults to None

  Deprecated since version 21.4: This argument will be converted to an optional argument in future versions.
- ***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.

Changed in version 20.0:

- Returns a tuple instead of a list.
- Filters out any None values

Parameters

- ***data*** (list[dict[str, ...]]) – The JSON data.
- ***bot*** (*telegram.Bot* | None) – The bot associated with these object. May be None, in which case shortcut methods will not be available.

  Changed in version 21.4: *bot* is now optional and defaults to None

  Deprecated since version 21.4: This argument will be converted to an optional argument in future versions.
- ***credentials*** (*telegram.FileCredentials*) – The credentials

Return type
tuple[*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 –

PersonalDetails

class telegram.**PersonalDetails**(*first_name*, *last_name*, *birth_date*, *gender*, *country_code*, *residence_country_code*, *first_name_native=None*, *last_name_native=None*, *middle_name=None*, *middle_name_native=None*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents personal details.

Available In
telegram.EncryptedPassportElement.data

Parameters

- ***first_name*** (str) – First Name.
- ***middle_name*** (str) – Optional. First Name.
- ***last_name*** (str) – Last Name.
- ***birth_date*** (str) – Date of birth in DD.MM.YYYY format.
- ***gender*** (str) – Gender, male or female.
- ***country_code*** (str) – Citizenship (ISO 3166-1 alpha-2 country code).
- ***residence_country_code*** (str) – Country of residence (ISO 3166-1 alpha-2 country code).
- ***first_name_native*** (str) – First Name in the language of the user's country of residence.
- ***middle_name_native*** (str) – Optional. Middle Name in the language of the user's country of residence.
- ***last_name_native*** (str) – Last Name in the language of the user's country of residence.

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

ResidentialAddress

class telegram.**ResidentialAddress**(*street_line1*, *street_line2*, *city*, *state*, *country_code*, *post_code*, *, *api_kwargs=None*)

Bases: *telegram.TelegramObject*

This object represents a residential address.

Available In

telegram.EncryptedPassportElement.data

Parameters

- ***street_line1*** (str) – First line for the address.
- ***street_line2*** (str) – Optional. Second line for the address.
- ***city*** (str) – City.
- ***state*** (str) – Optional. State.
- ***country_code*** (str) – ISO 3166-1 alpha-2 country code.
- ***post_code*** (str) – Address post code.

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

SecureData

class telegram.**SecureData**(*personal_details=None*, *passport=None*, *internal_passport=None*, *driver_license=None*, *identity_card=None*, *address=None*, *utility_bill=None*, *bank_statement=None*, *rental_agreement=None*, *passport_registration=None*, *temporary_registration=None*, ***, *api_kwargs=None*)

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.

Available In

telegram.Credentials.secure_data

Parameters

- ***personal_details*** (*telegram.SecureValue*, optional) – Credentials for encrypted personal details.
- ***passport*** (*telegram.SecureValue*, optional) – Credentials for encrypted passport.
- ***internal_passport*** (*telegram.SecureValue*, optional) – Credentials for encrypted internal passport.
- ***driver_license*** (*telegram.SecureValue*, optional) – Credentials for encrypted driver license.
- ***identity_card*** (*telegram.SecureValue*, optional) – Credentials for encrypted ID card
- ***address*** (*telegram.SecureValue*, optional) – Credentials for encrypted residential address.
- ***utility_bill*** (*telegram.SecureValue*, optional) – Credentials for encrypted utility bill.
- ***bank_statement*** (*telegram.SecureValue*, optional) – Credentials for encrypted bank statement.
- ***rental_agreement*** (*telegram.SecureValue*, optional) – Credentials for encrypted rental agreement.
- ***passport_registration*** (*telegram.SecureValue*, optional) – Credentials for encrypted registration from internal passport.
- ***temporary_registration*** (*telegram.SecureValue*, optional) – Credentials for encrypted temporary registration.

personal_details

Optional. Credentials for encrypted personal details.

Type
telegram.SecureValue

passport

Optional. Credentials for encrypted passport.

Type
telegram.SecureValue

internal_passport

Optional. Credentials for encrypted internal passport.

Type
telegram.SecureValue

driver_license

Optional. Credentials for encrypted driver license.

Type
telegram.SecureValue

identity_card

Optional. Credentials for encrypted ID card

Type
telegram.SecureValue

address
Optional. Credentials for encrypted residential address.

Type
telegram.SecureValue

utility_bill
Optional. Credentials for encrypted utility bill.

Type
telegram.SecureValue

bank_statement
Optional. Credentials for encrypted bank statement.

Type
telegram.SecureValue

rental_agreement
Optional. Credentials for encrypted rental agreement.

Type
telegram.SecureValue

passport_registration
Optional. Credentials for encrypted registration from internal passport.

Type
telegram.SecureValue

temporary_registration
Optional. Credentials for encrypted temporary registration.

Type
telegram.SecureValue

classmethod de_json(*data*, *bot=None*)
See *telegram.TelegramObject.de_json()*.

SecureValue

class telegram.**SecureValue**(*data=None*, *front_side=None*, *reverse_side=None*, *selfie=None*, *files=None*, *translation=None*, ***, *api_kwargs=None*)

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.

Available In

- *telegram.SecureData.address*
- *telegram.SecureData.bank_statement*
- *telegram.SecureData.driver_license*
- *telegram.SecureData.identity_card*
- *telegram.SecureData.internal_passport*
- *telegram.SecureData.passport_registration*

- *telegram.SecureData.passport*
- *telegram.SecureData.personal_details*
- *telegram.SecureData.rental_agreement*
- *telegram.SecureData.temporary_registration*
- *telegram.SecureData.utility_bill*

Parameters

- ***data*** (*telegram.DataCredentials*, optional) – Credentials for encrypted Telegram Passport data. Available for "personal_details", "passport", "driver_license", "identity_card", "identity_passport" and "address" types.
- ***front_side*** (*telegram.FileCredentials*, optional) – Credentials for encrypted document's front side. Available for "passport", "driver_license", "identity_card" and "internal_passport".
- ***reverse_side*** (*telegram.FileCredentials*, optional) – Credentials for encrypted document's reverse side. Available for "driver_license" and "identity_card".
- ***selfie*** (*telegram.FileCredentials*, optional) – Credentials for encrypted selfie of the user with a document. Can be available for "passport", "driver_license", "identity_card" and "internal_passport".
- ***translation*** (list[*telegram.FileCredentials*], optional) – 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".
- ***files*** (list[*telegram.FileCredentials*], optional) – Credentials for encrypted files. Available for "utility_bill", "bank_statement", "rental_agreement", "passport_registration" and "temporary_registration" types.

data

Optional. Credentials for encrypted Telegram Passport data. Available for "personal_details", "passport", "driver_license", "identity_card", "identity_passport" and "address" types.

Type
telegram.DataCredentials

front_side

Optional. Credentials for encrypted document's front side. Available for "passport", "driver_license", "identity_card" and "internal_passport".

Type
telegram.FileCredentials

reverse_side

Optional. Credentials for encrypted document's reverse side. Available for "driver_license" and "identity_card".

Type
telegram.FileCredentials

selfie

Optional. 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

translation

Optional. 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".

Changed in version 20.0: This attribute is now an immutable tuple.

Type
tuple[*telegram.FileCredentials*]

files

Optional. Credentials for encrypted files. Available for "utility_bill", "bank_statement", "rental_agreement", "passport_registration" and "temporary_registration" types.

Changed in version 20.0:

- This attribute is now an immutable tuple.
- This attribute is now always a tuple, that may be empty.

Type
tuple[*telegram.FileCredentials*]

classmethod de_json(*data*, *bot=None*)

See *telegram.TelegramObject.de_json()*.

6.2 telegram.ext package

Extensions over the Telegram Bot API to facilitate bot making

6.2.1 Application

class telegram.ext.**Application**(***, *bot*, *update_queue*, *updater*, *job_queue*, *update_processor*, *persistence*, *context_types*, *post_init*, *post_shutdown*, *post_stop*)

Bases: contextlib.AbstractAsyncContextManager, typing.Generic

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()
```

See also

__aenter__() and *__aexit__()*.

This class is a Generic class and accepts six type variables:

1. The type of *bot*. Must be *telegram.Bot* or a subclass of that class.
2. The type of the argument context of callback functions for (error) handlers and jobs. Must be *telegram.ext.CallbackContext* or a subclass of that class. This must be consistent with the following types.
3. The type of the values of *user_data*.
4. The type of the values of *chat_data*.
5. The type of *bot_data*.
6. The type of *job_queue*. Must either be *telegram.ext.JobQueue* or a subclass of that or None.

Examples

Echo Bot

See also

Your First Bot, Architecture Overview

Use In

telegram.ext.ApplicationBuilder.application_class()

Available In

- *telegram.ext.CallbackContext.application*
- *telegram.ext.JobQueue.application*

Returned In

telegram.ext.ApplicationBuilder.build()

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

chat_data

A dictionary handlers can use to store data for the chat. For each integer chat id, the corresponding value of this mapping is available as *telegram.ext.CallbackContext.chat_data* in handler callbacks for updates from that chat.

Changed in version 20.0: *chat_data* is now read-only. Note that the values of the mapping are still mutable, i.e. editing `context.chat_data` within a handler callback is possible (and encouraged), but editing the mapping `application.chat_data` itself is not.

Tip

- Manually modifying *chat_data* is almost never needed and unadvisable.
- Entries are never deleted automatically from this mapping. If you want to delete the data associated with a specific chat, e.g. if the bot got removed from that chat, please use *drop_chat_data()*.

Type
types.MappingProxyType

user_data

A dictionary handlers can use to store data for the user. For each integer user id, the corresponding value of this mapping is available as *telegram.ext.CallbackContext.user_data* in handler callbacks for updates from that user.

Changed in version 20.0: *user_data* is now read-only. Note that the values of the mapping are still mutable, i.e. editing `context.user_data` within a handler callback is possible (and encouraged), but editing the mapping `application.user_data` itself is not.

Tip

- Manually modifying *user_data* is almost never needed and unadvisable.
- Entries are never deleted automatically from this mapping. If you want to delete the data associated with a specific user, e.g. if that user blocked the bot, please use *drop_user_data()*.

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

post_stop

Optional. A callback that will be executed by *Application.run_polling()* and *Application.run_webhook()* after stopping the application via *stop()*.

Added in version 20.1.

Type
coroutine function

async __aenter__()

Asynchronous context manager which *initializes* the App.

Returns
The initialized App instance.

Raises
Exception – If an exception is raised during initialization, *shutdown()* is called in this case.

async __aexit__(*exc_type*, *exc_val*, *exc_tb*)

Asynchronous context manager which *shuts down* the App.

__repr__()

Give a string representation of the application in the form `Application[bot=...]`.

As this class doesn't implement `object.__str__()`, the default implementation will be used, which is equivalent to *__repr__()*.

Returns
`str`

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.

Examples

Errorhandler Bot

Hint

This method currently has no influence on calls to *process_error()* that are already in progress.

Warning

This behavior should currently be considered an implementation detail and not as guaranteed behavior.

See also

Exceptions, Warnings and Logging

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: object | None, 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.

Hint

This method currently has no influence on calls to *process_update()* that are already in progress.

Warning

This behavior should currently be considered an implementation detail and not as guaranteed behavior.

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.

Added in version 20.0.

Parameters

- ***handlers*** (Sequence[*telegram.ext.BaseHandler*] | dict[int, Sequence[*telegram.ext.BaseHandler*]]) – Specify a sequence of handlers *or* a dictionary where the keys are groups and values are handlers.

  Changed in version 21.7: Accepts any collections.abc.Sequence as input instead of just a list or tuple.

- ***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(...)]
}
```

Raises

TypeError – If the combination of arguments is invalid.

static builder()

Convenience method. Returns a new *telegram.ext.ApplicationBuilder*.

Added 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.

Changed in version 20.4: This is now just a shortcut to *update_processor.max_concurrent_updates*.

See also

Concurrency

Type

int

create_task(*coroutine*, *update=None*, ***, *name=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()*.

See also

Concurrency

Parameters

- ***coroutine*** (awaitable) – The awaitable to run as task.

  Changed in version 20.2: Accepts asyncio.Future and generator-based coroutine functions.

  Deprecated since version 20.4: Since Python 3.12, generator-based coroutine functions are no longer accepted.

- ***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.

Keyword Arguments
name (str, optional) – The name of the task.

Added in version 20.4.

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.

Added 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.

Added 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.
- The *update_processor* by calling *telegram.ext.BaseUpdateProcessor.initialize()*.

Does *not* call *post_init* - that is only done by *run_polling()* and *run_webhook()*.

> **See also**
>
> *shutdown()*

property job_queue

The *JobQueue* used by the
telegram.ext.Application.

> **See also**
>
> Job Queue

Type
telegram.ext.JobQueue

mark_data_for_update_persistence(*chat_ids=None*, *user_ids=None*)

Mark entries of *chat_data* and *user_data* to be updated on the next run of *update_persistence()*.

> **Tip**
>
> Use this method sparingly. If you have to use this method, it likely means that you access and modify `context.application.chat/user_data[some_id]` within a callback. Note that for data which should be available globally in all handler callbacks independent of the chat/user, it is recommended to use *bot_data* instead.

Added in version 20.3.

Parameters

- ***chat_ids*** (`int` | Collection[`int`], optional) – Chat IDs to mark.
- ***user_ids*** (`int` | Collection[`int`], optional) – User IDs to mark.

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*.
>
> 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.

> **See also**
>
> Storing Bot, User and Chat Related Data

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.

  Added 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_error()*.

> **See also**
>
> Concurrency

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.

> **Hint**
>
> This method currently has no influence on calls to *process_error()* that are already in progress.
>
> > **Warning**
> >
> > This behavior should currently be considered an implementation detail and not as guaranteed behavior.

Parameters

callback (coroutine function) – The error handler to remove.

remove_handler(*handler*, *group=0*)

Remove a handler from the specified group.

> **Hint**
>
> This method currently has no influence on calls to *process_update()* that are already in progress.
>
> > **Warning**
> >
> > This behavior should currently be considered an implementation detail and not as guaranteed behavior.

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=datetime.timedelta(seconds=10)*, *bootstrap_retries=0*, *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. This also works from within handlers, error handlers and jobs. However, using *stop_running()* will give a somewhat

cleaner shutdown behavior than manually raising those exceptions. On unix, the app will also shut down on receiving the signals specified by *stop_signals*.

The order of execution by *run_polling()* is roughly as follows:

- *initialize()*
- *post_init()*
- *telegram.ext.Updater.start_polling()*
- *start()*
- Run the application until the users stops it
- *telegram.ext.Updater.stop()*
- *stop()*
- *post_stop()*
- *shutdown()*
- *post_shutdown()*

A small wrapper is passed to *telegram.ext.Updater.start_polling.error_callback* which forwards errors occurring during polling to *registered error handlers*. The update parameter of the callback will be set to `None`.

Tip

- When combining `python-telegram-bot` with other `asyncio` based frameworks, using this method is likely not the best choice, as it blocks the event loop until it receives a stop signal as described above. Instead, you can manually call the methods listed below to start and shut down the application and the *updater*. Keeping the event loop running and listening for a stop signal is then up to you.
- To gracefully stop the execution of this method from within a handler, job or error callback, use *stop_running()*.

Changed in version Removed: the deprecated parameters `read_timeout`, `write_timeout`, `connect_timeout`, and `pool_timeout`. Use the corresponding methods in *telegram.ext.ApplicationBuilder* instead.

Parameters

- ***poll_interval*** (`float`, optional) – Time to wait between polling updates from Telegram in seconds. Default is `0.0`.
- ***timeout*** (`int | datetime.timedelta`, optional) – Passed to *telegram.Bot.get_updates.timeout*. Default is `timedelta(seconds=10)`.

  Changed in version v22.2: `datetime.timedelta` objects are accepted in addition to plain `int` values.
- ***bootstrap_retries*** (`int`, optional) – Whether the bootstrapping phase (calling *initialize()* and the boostrapping of *telegram.ext.Updater.start_polling()*) will retry on failures on the Telegram server.
  - < 0 - retry indefinitely
  - 0 - no retries (default)
  - > 0 - retry up to X times

  Changed in version 21.11: The default value will be changed to from `-1` to `0`. Indefinite retries during bootstrapping are not recommended.

- ***drop_pending_updates*** (bool, optional) – Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is False.
- ***allowed_updates*** (Sequence[str], optional) – Passed to *telegram.Bot.get_updates()*.

  Changed in version 21.9: Accepts any collections.abc.Sequence as input instead of just a list
- ***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 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*, *unix=None*)

Convenience method that takes care of initializing and starting the app, listening for 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. This also works from within handlers, error handlers and jobs. However, using *stop_running()* will give a somewhat cleaner shutdown behavior than manually raising those exceptions. 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.

The order of execution by *run_webhook()* is roughly as follows:

- *initialize()*
- *post_init()*
- *telegram.ext.Updater.start_webhook()*
- *start()*
- Run the application until the users stops it
- *telegram.ext.Updater.stop()*
- *stop()*

- *post_stop()*
- *shutdown()*
- *post_shutdown()*

Important

If you want to use this method, you must install PTB with the optional requirement `webhooks`, i.e.

```
pip install "python-telegram-bot[webhooks]"
```

Tip

- When combining `python-telegram-bot` with other `asyncio` based frameworks, using this method is likely not the best choice, as it blocks the event loop until it receives a stop signal as described above. Instead, you can manually call the methods listed below to start and shut down the application and the *updater*. Keeping the event loop running and listening for a stop signal is then up to you.
- To gracefully stop the execution of this method from within a handler, job or error callback, use *stop_running()*.

See also

Webhooks

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 (calling *initialize()* and the boostrapping of *telegram.ext.Updater.start_polling()*) 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*** (Sequence[`str`], optional) – Passed to *telegram.Bot.set_webhook()*.

  Changed in version 21.9: Accepts any `collections.abc.Sequence` as input instead of just a list

- ***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.

  Added in version 20.0.

- ***unix*** (pathlib.Path | str | socket.socket, optional) – Can be either:
  - the path to the unix socket file as pathlib.Path or str. This will be passed to tornado.netutil.bind_unix_socket to create the socket. If the Path does not exist, the file will be created.
  - or the socket itself. This option allows you to e.g. restrict the permissions of the socket for improved security. Note that you need to pass the correct family, type and socket options yourself.

  Caution

  This parameter is a replacement for the default TCP bind. Therefore, it is mutually exclusive with *listen* and *port*. When using this param, you must also run a reverse proxy to the unix socket and set the appropriate *webhook_url*.

  Added in version 20.8.

  Changed in version 21.1: Added support to pass a socket instance itself.

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()*
- *update_processor* by calling *telegram.ext.BaseUpdateProcessor.shutdown()*

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 via *process_update()*.
- *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()*.

Tip

When using a custom logic for startup and shutdown of the application, eventual cancellation of pending tasks should happen only *after* *stop()* has been called in order to ensure that the tasks mentioned above are not cancelled prematurely.

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()*.
- Does *not* call *post_stop* - that is only done by *run_polling()* and *run_webhook()*.

Raises

RuntimeError – If the application is not running.

stop_running()

This method can be used to stop the execution of *run_polling()* or *run_webhook()* from within a handler, job or error callback. This allows a graceful shutdown of the application, i.e. the methods listed in *run_polling* and *run_webhook* will still be executed.

This method can also be called within *post_init()*. This allows for a graceful, early shutdown of the application if some condition is met (e.g., a database connection could not be established).

Note

If the application is not running and this method is not called within *post_init()*, this method does nothing.

Warning

This method is designed to for use in combination with *run_polling()* or *run_webhook()*. Using this method in combination with a custom logic for starting and stopping the application is not guaranteed to work as expected. Use at your own risk.

Added in version 20.5.

Changed in version 21.2: Added support for calling within *post_init()*.

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* and *chat_data*, only those entries are updated which either were used or have been manually marked via *mark_data_for_update_persistence()* since the last run of this method.

> **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*, *mark_data_for_update_persistence()*

property update_processor

The update processor used by this application.

> **See also**
>
> Concurrency

Added in version 20.4.

Type

telegram.ext.BaseUpdateProcessor

6.2.2 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.

See also

Your First Bot, Builder Pattern

Changed in version 22.0: Removed deprecated methods `proxy_url` and `get_updates_proxy_url`.

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.

Important

If you want to use this feature, you must install PTB with the optional requirement `callback-data`, i.e.

```
pip install "python-telegram-bot[callback-data]"
```

Examples

Arbitrary callback_data Bot

See also

Arbitrary callback_data

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()*

Changed in version 21.11: Supports callable input and string formatting.

Parameters
base_file_url (`str` | Callable[[`str`], `str`]) – The URL or input for the URL as accepted by *telegram.Bot.base_file_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()*

Changed in version 21.11: Supports callable input and string formatting.

Parameters
base_url (`str` | Callable[[`str`], `str`]) – The URL or input for the URL as accepted by *telegram.Bot.base_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. If not called, updates will be processed 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 | *BaseUpdateProcessor*) – Passing `True` will allow for `256` updates to be processed concurrently using *telegram.ext.SimpleUpdateProcessor*. Pass an integer to specify a different number of updates that may be processed concurrently. Pass an instance of *telegram.ext.BaseUpdateProcessor* to use that instance for handling updates concurrently.

Changed in version 20.4: Now accepts *BaseUpdateProcessor* instances.

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`.

> **See also**
>
> *get_updates_connect_timeout()*

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.

See also

get_updates_connection_pool_size()

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*.

Examples

Context Types Bot

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 to Your Bot

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`.

> **See also**
>
> *connect_timeout()*

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.

> **See also**
>
> *connection_pool_size()*

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_http_version(*get_updates_http_version*)

Sets the HTTP protocol version which is used for the *http_version* parameter which is used in the *telegram.Bot.get_updates()* request. By default, HTTP/1.1 is used.

> **See also**
>
> *http_version()*

> **Note**
>
> Users have observed stability issues with HTTP/2, which happen due to how the h2 library handles cancellations of keepalive connections. See #3556 for a discussion.
>
> You will also need to install the http2 dependency. Keep in mind that the HTTP/1.1 implementation may be considered the "more robust option at this time".
>
> ```
> pip install httpx[http2]
> ```

Added in version 20.1.

Changed in version 20.2: Reset the default version to 1.1.

Parameters

get_updates_http_version (`str`) – Pass `"2"` or `"2.0"` if you'd like to use HTTP/2 for making requests to Telegram. Defaults to `"1.1"`, in which case HTTP/1.1 is used.

Changed in version 20.5: Accept `"2"` as a valid value.

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 `1.0`.

> **See also**
>
> *pool_timeout()*

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`(*get_updates_proxy*)

Sets the proxy for the *telegram.request.HTTPXRequest.proxy* parameter which is used for *telegram.Bot.get_updates()*. Defaults to `None`.

> **See also**
>
> *proxy()*

Added in version 20.7.

Parameters

proxy (`str` | `httpx.Proxy` | `httpx.URL`) – The URL to a proxy server, a `httpx.Proxy` object or a `httpx.URL` object. See *telegram.request.HTTPXRequest.proxy* 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`.

See also

read_timeout()

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_socket_options(*get_updates_socket_options*)

Sets the options for the *socket_options* parameter of *telegram.Bot.get_updates_request*. Defaults to `None`.

See also

socket_options()

Added in version 20.7.

Parameters
get_updates_socket_options (Collection[`tuple`], optional) – Socket options. See *telegram.request.HTTPXRequest.socket_options* for more information.

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`.

See also

write_timeout()

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

`http_version`(*http_version*)

Sets the HTTP protocol version which is used for the *http_version* parameter of *telegram.Bot.request*. By default, HTTP/1.1 is used.

See also

get_updates_http_version()

Note

Users have observed stability issues with HTTP/2, which happen due to how the h2 library handles cancellations of keepalive connections. See #3556 for a discussion.

If you want to use HTTP/2, you must install PTB with the optional requirement `http2`, i.e.

```
pip install "python-telegram-bot[http2]"
```

Keep in mind that the HTTP/1.1 implementation may be considered the "more robust option at this time".

Added in version 20.1.

Changed in version 20.2: Reset the default version to 1.1.

Parameters
http_version (`str`) – Pass `"2"` or `"2.0"` if you'd like to use HTTP/2 for making requests to Telegram. Defaults to `"1.1"`, in which case HTTP/1.1 is used.

Changed in version 20.5: Accept `"2"` as a valid value.

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 if the requirements of *telegram.ext.JobQueue* are installed.

Examples

Timer Bot

See also

Job Queue

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` or when the requirements of *telegram.ext.JobQueue* are not installed, *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

local_mode(*local_mode*)

Specifies the value for *local_mode* for the *telegram.ext.Application.bot*. If not called, will default to `False`.

See also

Local Bot API Server

Parameters
local_mode (`bool`) – Whether the bot should run in local mode.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

media_write_timeout(*media_write_timeout*)

Sets the media write operation timeout for the *media_write_timeout* parameter of *telegram.Bot.request*. Defaults to `20`.

Added in version 21.0.

Parameters
media_write_timeout (`float`) – See *telegram.request.HTTPXRequest.media_write_timeout* for more information.

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.

Examples

Persistent Conversation Bot

See also

Making Your Bot Persistent

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 `1.0`.

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

get_updates_pool_timeout()

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()
```

Note

If you implement custom logic that implies that you will **not** be using *Application*'s methods *run_polling()* or *run_webhook()* to run your application (like it's done in Custom Webhook Bot Example), the callback you set in this method **will not be called automatically**. So instead of setting a callback with this method, you have to explicitly `await` the function that you want to run at this stage of your application's life (in the example mentioned above, that would be in `async with application` context manager).

See also

post_stop(), *post_shutdown()*

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()
```

Note

If you implement custom logic that implies that you will **not** be using *Application*'s methods *run_polling()* or *run_webhook()* to run your application (like it's done in Custom Webhook Bot Example), the callback you set in this method **will not be called automatically**. So instead of setting a callback with this method, you have to explicitly `await` the function that you want to run at this stage of your application's life (in the example mentioned above, that would be in `async with application` context manager).

See also

post_init(), *post_stop()*

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

post_stop(*post_stop*)

Sets a callback to be executed by *Application.run_polling()* and *Application.run_webhook()* *after* executing *Updater.stop()* and *Application.stop()*.

Added in version 20.1.

Tip

This can be used for custom stop logic that requires to await coroutines, e.g. sending message to a chat before shutting down the bot.

Hint

The callback will be called only, if *Application.stop()* was indeed called successfully. For example, if the application is stopped early by calling *Application.stop_running()* within *post_init()*, then the set callback will *not* be called.

Example

```
async def post_stop(application: Application) -> None:
    await application.bot.send_message(123456, "Shutting down...")

application = Application.builder()
                         .token("TOKEN")
                         .post_stop(post_stop)
                         .build()
```

Note

If you implement custom logic that implies that you will **not** be using *Application*'s methods *run_polling()* or *run_webhook()* to run your application (like it's done in Custom Webhook Bot Example), the callback you set in this method **will not be called automatically**. So instead of setting a callback with this method, you have to explicitly `await` the function that you want to run at this stage of your application's life (in the example mentioned above, that would be in `async with application` context manager).

See also

post_init(), *post_shutdown()*

Parameters

post_stop (coroutine function) – The custom callback. Must be a coroutine function and must accept exactly one positional argument, which is the *Application*:

```
async def post_stop(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*.

Examples

Passport Bot

See also

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(*proxy*)

Sets the proxy for the *proxy* parameter of *telegram.Bot.request*. Defaults to None.

See also

get_updates_proxy()

Added in version 20.7.

Parameters
proxy (str | httpx.Proxy | httpx.URL) – The URL to a proxy server, a httpx.Proxy object or a httpx.URL object. See *telegram.request.HTTPXRequest.proxy* 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.

> **See also**
>
> *get_updates_read_timeout()*

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

socket_options(*socket_options*)

Sets the options for the *socket_options* parameter of *telegram.Bot.request*. Defaults to None.

> **See also**
>
> *get_updates_socket_options()*

Added in version 20.7.

Parameters
socket_options (Collection[tuple], optional) – Socket options. See *telegram.request.HTTPXRequest.socket_options* for more information.

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.

> **See also**
> *get_updates_write_timeout()*

Parameters
write_timeout (float) – See *telegram.request.HTTPXRequest.write_timeout* for more information.

Returns
The same builder with the updated argument.

Return type
ApplicationBuilder

6.2.3 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

6.2.4 BaseUpdateProcessor

class telegram.ext.**BaseUpdateProcessor**(*max_concurrent_updates*)

Bases: contextlib.AbstractAsyncContextManager, ABC

An abstract base class for update processors. You can use this class to implement your own update processor.

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

```
async with processor:
    # code
```

is roughly equivalent to

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

See also

__aenter__() and *__aexit__()*.

See also

Concurrency

Use In

telegram.ext.ApplicationBuilder.concurrent_updates()

Available In

telegram.ext.Application.update_processor

Added in version 20.4.

Parameters

max_concurrent_updates (int) – The maximum number of updates to be processed concurrently. If this number is exceeded, new updates will be queued until the number of currently processed updates decreases.

Raises

ValueError – If *max_concurrent_updates* is a non-positive integer.

async __aenter__()

Asynchronous context manager which *initializes* the Processor.

Returns

The initialized Processor instance.

Raises

Exception – If an exception is raised during initialization, *shutdown()* is called in this case.

async __aexit__(*exc_type*, *exc_val*, *exc_tb*)

Asynchronous context manager which *shuts down* the Processor.

property current_concurrent_updates

The number of updates currently being processed.

Caution

This value is a snapshot of the current number of updates being processed. It may change immediately after being read.

Added in version 21.11.

Type

int

abstractmethod async do_process_update(*update*, *coroutine*)

Custom implementation of how to process an update. Must be implemented by a subclass.

Warning

This method will be called by *process_update()*. It should *not* be called manually.

Parameters

- ***update*** (object) – The update to be processed.
- ***coroutine*** (Awaitable) – The coroutine that will be awaited to process the update.

abstractmethod async initialize()

Initializes the processor so resources can be allocated. Must be implemented by a subclass.

> **See also**
>
> *shutdown()*

property max_concurrent_updates

The maximum number of updates that can be processed concurrently.

Type

int

final async process_update(*update*, *coroutine*)

Calls *do_process_update()* with a semaphore to limit the number of concurrent updates.

Parameters

- ***update*** (object) – The update to be processed.
- ***coroutine*** (Awaitable) – The coroutine that will be awaited to process the update.

abstractmethod async shutdown()

Shutdown the processor so resources can be freed. Must be implemented by a subclass.

> **See also**
>
> *initialize()*

6.2.5 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).

Examples

- *Context Types Bot*
- *Custom Webhook Bot*

See also

telegram.ext.ContextTypes.DEFAULT_TYPE, Job Queue

Use In

telegram.ext.ApplicationBuilder.context_types()

Returned In

telegram.ext.Application.builder()

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*.

  Added in version 20.0.
- ***user_id*** (int, optional) – The ID of the user associated with this object. Used to provide *user_data*.

  Added in version 20.0.

coroutine

Optional. Only present in error handlers if the error was caused by an awaitable run with *Application.create_task()* or a handler callback with *block=False*.

Type

awaitable

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.

See also

Storing Bot, User and Chat Related Data

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.

See also

Storing Bot, User and Chat Related Data

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.

Added 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.

> **See also**
>
> Arbitrary callback_data

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.

  Added in version 20.0.
- ***coroutine*** (awaitable, optional) – The awaitable associated with this error if the error was caused by a coroutine run with *Application.create_task()* or a handler callback with *block=False*.

  Added in version 20.0.

  Changed in version 20.2: Accepts asyncio.Future and generator-based coroutine functions.

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*.

> **See also**
>
> Job Queue

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()*.

Added 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`.

See also

Storing Bot, User and Chat Related Data

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

6.2.6 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.

Examples

ContextTypes Bot

See also

Architecture Overview, Storing Bot, User and Chat Related Data

Use In

telegram.ext.ApplicationBuilder.context_types()

Available In

- *telegram.ext.Application.context_types*
- *telegram.ext.PicklePersistence.context_types*

Added 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*[None], dict[Any, Any], dict[Any, Any], dict[Any, Any]]

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.

6.2.7 Defaults

final class telegram.ext.**Defaults**(*parse_mode=None*, *disable_notification=None*, *tzinfo=datetime.timezone.utc*, *block=True*, *allow_sending_without_reply=None*, *protect_content=None*, *link_preview_options=None*, *do_quote=None*)

Bases: `object`

Convenience Class to gather all parameters with a (user defined) default value

See also

Architecture Overview, Adding Defaults to Your Bot

Use In

telegram.ext.ApplicationBuilder.defaults()

Available In

telegram.ext.ExtBot.defaults

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.

Changed in version 22.0: Removed deprecated arguments and properties `disable_web_page_preview` and `quote`. Use *link_preview_options* and *do_quote* instead.

Parameters

- ***parse_mode*** (`str`, optional) – Mode for parsing entities. See *telegram.constants.ParseMode* and formatting options for more details.
- ***disable_notification*** (`bool`, optional) – Sends the message silently. Users will receive a notification with no sound.
- ***allow_sending_without_reply*** (`bool`, optional) – Pass `True`, if the message should be sent even if the specified replied-to message is not found.. Will be used for *telegram.ReplyParameters.allow_sending_without_reply*.
- ***tzinfo*** (`datetime.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*. Defaults to `datetime.timezone.utc` otherwise.

  Deprecated since version 21.10: Support for `pytz` timezones is deprecated and will be removed in future versions.
- ***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.

  Added in version 20.0.
- ***link_preview_options*** (*telegram.LinkPreviewOptions*, optional) – Link preview generation options for all outgoing messages. Mutually exclusive with *disable_web_page_preview*. This object is used for the corresponding parameter of *telegram.Bot.send_message()*, *telegram.Bot.edit_message_text()*, and *telegram.InputTextMessageContent* if not specified. If a value is specified for the corresponding parameter, only those parameters of *telegram.LinkPreviewOptions* will be overridden that are not explicitly set.

  Example

  ```
  from telegram import LinkPreviewOptions
  from telegram.ext import Defaults, ExtBot

  defaults = Defaults(
      link_preview_options=LinkPreviewOptions(show_above_
  ↪text=True)
  )
  ```

```
chat_id = 123

async def main():
    async with ExtBot("Token", defaults=defaults) as bot:
        # The link preview will be shown above the text.
        await bot.send_message(chat_id, "https://python-
↪telegram-bot.org")

        # The link preview will be shown below the text.
        await bot.send_message(
            chat_id,
            "https://python-telegram-bot.org",
            link_preview_options=LinkPreviewOptions(show_above_
↪text=False)
        )

        # The link preview will be shown above the text, but␣
↪the preview will
        # show Telegram.
        await bot.send_message(
            chat_id,
            "https://python-telegram-bot.org",
            link_preview_options=LinkPreviewOptions(url=
↪"https://telegram.org")
        )
```

Added in version 20.8.

- ***do_quote*** (bool, optional) – If set to True, the reply 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.

  Added in version 20.8.

__eq__(*other*)

Defines equality condition for the *Defaults* object. Two objects of this class are considered to be equal if all their parameters are identical.

Returns
True if both objects have all parameters identical. False otherwise.

__hash__()

Builds a hash value for this object such that the hash of two objects is equal if and only if the objects are equal in terms of *__eq__()*.

Returns
int The hash value of the object.

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 do_quote

Optional. If set to `True`, the reply 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.

Added in version 20.8.

Type
bool

property explanation_parse_mode

Optional. Alias for *parse_mode*, used for the corresponding parameter of *telegram.Bot.send_poll()*.

Type
str

property link_preview_options

Optional. Link preview generation options for all outgoing messages.

Added in version 20.8.

Type
telegram.LinkPreviewOptions

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.

Added in version 20.0.

Type
bool

property question_parse_mode

Optional. Alias for *parse_mode*, used for the corresponding parameter of *telegram.Bot.send_poll()*.

Added in version 21.2.

Type
str

property quote_parse_mode

Optional. Alias for *parse_mode*, used for the corresponding parameter of *telegram.ReplyParameters()*.

Type
str

property text_parse_mode

Optional. Alias for *parse_mode*, used for the corresponding parameter of *telegram.InputPollOption* and *telegram.Bot.send_gift()*.

Added in version 21.2.

Type
str

property tzinfo

A timezone to be used for all date(time) objects appearing throughout PTB.

Type
tzinfo

6.2.8 ExtBot

class telegram.ext.**ExtBot**(*token*, *base_url='https://api.telegram.org/bot'*, *base_file_url='https://api.telegram.org/file/bot'*, *request=None*, *get_updates_request=None*, *private_key=None*, *private_key_password=None*, *defaults=None*, *arbitrary_callback_data=False*, *local_mode=False*, *rate_limiter=None*)

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*.

This class is a Generic class and accepts one type variable that specifies the generic type of the *rate_limiter* used by the bot. Use None if no rate limiter is used.

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.

Examples

Arbitrary Callback Data Bot

See also

Arbitrary callback_data

Use In

telegram.ext.ApplicationBuilder.bot()

Available In

- *telegram.ext.Application.bot*
- *telegram.ext.CallbackContext.bot*
- *telegram.ext.CallbackDataCache.bot*
- *telegram.ext.Updater.bot*

Returned In

- *telegram.ext.Application.builder()*
- *telegram.ext.ApplicationBuilder.rate_limiter()*

Added in version 13.6.

Changed in version 20.0: Removed the attribute `arbitrary_callback_data`. You can instead use *bot.callback_data_cache.maxsize* to access the size of the cache.

Changed in version 20.5: Removed deprecated methods `set_sticker_set_thumb` and `setStickerSetThumb`.

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. Defaults to `False`.

  See also

  Arbitrary callback_data

- ***rate_limiter*** (*telegram.ext.BaseRateLimiter*, optional) – A rate limiter to use for limiting the number of requests made by the bot per time interval.

  Added in version 20.0.

property callback_data_cache

Optional. The cache for objects passed as callback data for *telegram.InlineKeyboardButton*.

Examples

Arbitrary Callback Data Bot

Changed in version 20.0: * This property is now read-only. * This property is now optional and can be `None` if *arbitrary_callback_data* is set to `False`.

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.

Added 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()*.

6.2.9 Job

class telegram.ext.**Job**(*callback*, *data=None*, *name=None*, *chat_id=None*, *user_id=None*)

Bases: `typing.Generic`

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.

This class is a `Generic` class and accepts one type variable that specifies the type of the argument `context` of *callback*.

> **Important**
>
> If you want to use this class, you must install PTB with the optional requirement `job-queue`, i.e.
>
> ```
> pip install "python-telegram-bot[job-queue]"
> ```

Note

All attributes and instance methods of *job* are also directly available as attributes/methods of the corresponding *telegram.ext.Job* object.

Warning

This class should not be instantiated manually. Use the methods of *telegram.ext.JobQueue* to schedule jobs.

See also

Job Queue

Use In

telegram.ext.Application.process_error()

Changed in version 20.0:

- Removed argument and attribute `job_queue`.
- Renamed `Job.context` to *Job.data*.
- Removed argument `job`
- To use this class, PTB must be installed via `pip install "python-telegram-bot[job-queue]"`.

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__`.
- ***chat_id*** (`int`, optional) – Chat id of the chat that this job is associated with.

  Added in version 20.0.
- ***user_id*** (`int`, optional) – User id of the user that this job is associated with.

  Added 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

chat_id

Optional. Chat id of the chat that this job is associated with.

Added in version 20.0.

Type
int

user_id

Optional. User id of the user that this job is associated with.

Added in version 20.0.

Type
int

__eq__(*other*)

Defines equality condition for the *telegram.ext.Job* object. Two objects of this class are considered to be equal if their `id` are equal.

Returns
`True` if both objects have *id* parameters identical. `False` otherwise.

__getattr__(*item*)

Overrides `object.__getattr__()` to get specific attribute of the *telegram.ext.Job* object or of its attribute `apscheduler.job.Job`, if exists.

Parameters
item (`str`) – The name of the attribute.

Returns
object: The value of the attribute.

Raises
AttributeError – If the attribute does not exist in both *telegram.ext.Job* and `apscheduler.job.Job` objects.

__hash__()

Builds a hash value for this object such that the hash of two objects is equal if and only if the objects are equal in terms of *__eq__()*.

Returns
The hash value of the object.

Return type
int

__repr__()

Give a string representation of the job in the form `Job[id=..., name=..., callback=..., trigger=...]`.

As this class doesn't implement `object.__str__()`, the default implementation will be used, which is equivalent to *__repr__()*.

Returns
str

property enabled

Whether this job is enabled.

Type
bool

classmethod from_aps_job(*aps_job*)

Provides the *telegram.ext.Job* that is associated with the given APScheduler job.

> **Tip**
>
> This method can be useful when using advanced APScheduler features along with *telegram.ext.JobQueue*.

Added in version 20.4.

Parameters
aps_job (apscheduler.job.Job) – The APScheduler job

Returns
telegram.ext.Job

property job

The APS Job this job is a wrapper for.

Changed in version 20.0: This property is now read-only.

Type
apscheduler.job.Job

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.

6.2.10 JobQueue

class telegram.ext.**JobQueue**

Bases: typing.Generic

This class allows you to periodically perform tasks with the bot. It is a convenience wrapper for the APScheduler library.

This class is a Generic class and accepts one type variable that specifies the type of the argument `context` of the job callbacks (*callback*) of *run_once()* and the other scheduling methods.

Important

If you want to use this class, you must install PTB with the optional requirement `job-queue`, i.e.

```
pip install "python-telegram-bot[job-queue]"
```

Examples

Timer Bot

See also

Architecture Overview, Job Queue

Use In

telegram.ext.ApplicationBuilder.job_queue()

Available In

- *telegram.ext.Application.job_queue*
- *telegram.ext.CallbackContext.job_queue*
- *telegram.ext.JobQueue.application*

Returned In

telegram.ext.Application.builder()

Changed in version 20.0: To use this class, PTB must be installed via `pip install "python-telegram-bot[job-queue]"`.

scheduler

The scheduler.

Warning

This scheduler is configured by *set_application()*. Additional configuration settings can be made by users. However, calling configure() will delete any previous configuration settings.

Therefore, please make sure to pass the values returned by *scheduler_configuration* to the method call in addition to your custom values. Alternatively, you can also use methods like add_jobstore() to avoid using configure() altogether.

Changed in version 20.0: Uses AsyncIOScheduler instead of BackgroundScheduler

Type
apscheduler.schedulers.asyncio.AsyncIOScheduler

__repr__()

Give a string representation of the JobQueue in the form JobQueue[application=...].

As this class doesn't implement object.__str__(), the default implementation will be used, which is equivalent to *__repr__()*.

Returns
str

property application

The application this JobQueue is associated with.

get_jobs_by_name(*name*)

Returns a tuple of all *scheduled* jobs with the given name that are currently in the *JobQueue*.

Hint

This method is a convenience wrapper for *jobs()* with a pattern that matches the given name.

Returns
Tuple of all *scheduled* jobs matching the name.

Return type
tuple[*Job*]

async static job_callback(*job_queue*, *job*)

This method is used as a callback for the APScheduler jobs.

More precisely, the func argument of apscheduler.job.Job is set to this method and the arg argument (representing positional arguments to func) is set to a tuple containing the *JobQueue* itself and the *Job* instance.

Tip

This method is a static method rather than a bound method. This makes the arguments more transparent and allows for easier handling of PTBs integration of APScheduler when utilizing advanced features of APScheduler.

Hint

This method is effectively a wrapper for *telegram.ext.Job.run()*.

Added in version 20.4.

Parameters

- ***job_queue*** (*JobQueue*) – The job queue that created the job.
- ***job*** (*Job*) – The job to run.

jobs(*pattern=None*)

Returns a tuple of all *scheduled* jobs that are currently in the *JobQueue*.

Parameters

pattern (str | re.Pattern, optional) – A regular expression pattern. If passed, only jobs whose name matches the pattern will be returned. Defaults to None.

> **Hint**
>
> This uses re.search() and not re.match().

Added in version 21.10.

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.

  Added 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.

  Added 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.

  Added 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.

  Added 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.

  Added 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.

  Added 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.

  Added 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.

Added 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*

Note

Setting *first* to 0, datetime.datetime.now() or another value that indicates that the job should run immediately will not work due to how the APScheduler library works. If you want to run a job immediately, we recommend to use an approach along the lines of:

```
job = context.job_queue.run_repeating(callback, interval=5)
await job.run(context.application)
```

See also

telegram.ext.Job.run()

- ***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.

  Added 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.

  Added 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

property scheduler_configuration

Provides configuration values that are used by *JobQueue* for *scheduler*.

Tip

Since calling scheduler.configure() deletes any previous setting, please make sure to pass these values to the method call in addition to your custom values:

```
scheduler.configure(..., **job_queue.scheduler_configuration)
```

Alternatively, you can also use methods like add_jobstore() to avoid using configure() altogether.

Added in version 20.7.

Returns

The configuration values as dictionary.

Return type
dict[str, object]

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.

6.2.11 SimpleUpdateProcessor

class telegram.ext.**SimpleUpdateProcessor**(*max_concurrent_updates*)

Bases: *telegram.ext.BaseUpdateProcessor*

Instance of *telegram.ext.BaseUpdateProcessor* that immediately awaits the coroutine, i.e. does not apply any additional processing. This is used by default when *telegram.ext.ApplicationBuilder.concurrent_updates* is int.

Use In

telegram.ext.ApplicationBuilder.concurrent_updates()

Available In

telegram.ext.Application.update_processor

Added in version 20.4.

async do_process_update(*update*, *coroutine*)

Immediately awaits the coroutine, i.e. does not apply any additional processing.

Parameters

- ***update*** (object) – The update to be processed.
- ***coroutine*** (Awaitable) – The coroutine that will be awaited to process the update.

async initialize()

Does nothing.

async shutdown()

Does nothing.

6.2.12 Updater

class telegram.ext.**Updater**(*bot*, *update_queue*)

Bases: contextlib.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()
```

See also

__aenter__() and *__aexit__()*.

See also

Architecture Overview, Builder Pattern

Use In

telegram.ext.ApplicationBuilder.updater()

Available In

telegram.ext.Application.updater

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 __aenter__()

Asynchronous context manager which *initializes* the Updater.

Returns
The initialized Updater instance.

Raises
Exception – If an exception is raised during initialization, *shutdown()* is called in this case.

async __aexit__(*exc_type*, *exc_val*, *exc_tb*)

Asynchronous context manager which *shuts down* the Updater.

__repr__()

Give a string representation of the updater in the form `Updater[bot=...]`.

As this class doesn't implement `object.__str__()`, the default implementation will be used, which is equivalent to *__repr__()*.

Returns
str

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=datetime.timedelta(seconds=10)*, *bootstrap_retries=0*, *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*.

Changed in version 22.0: Removed the deprecated arguments `read_timeout`, `write_timeout`, `connect_timeout`, and `pool_timeout` in favor of setting the timeouts via the corresponding methods of *telegram.ext.ApplicationBuilder*. or by specifying the timeout via *telegram.Bot.get_updates_request*.

Parameters

- ***poll_interval*** (`float`, optional) – Time to wait between polling updates from Telegram in seconds. Default is `0.0`.
- ***timeout*** (`int | datetime.timedelta`, optional) – Passed to *telegram.Bot.get_updates.timeout*. Defaults to `timedelta(seconds=10)`.

  Changed in version v22.2: `datetime.timedelta` objects are accepted in addition to plain `int` values.

- ***bootstrap_retries*** (int, optional) – Whether the bootstrapping phase of will retry on failures on the Telegram server.
  - < 0 - retry indefinitely
  - 0 - no retries (default)
  - > 0 - retry up to X times

  Changed in version 21.11: The default value will be changed to from `-1` to `0`. Indefinite retries during bootstrapping are not recommended.
- ***allowed_updates*** (Sequence[str], optional) – Passed to *telegram.Bot.get_updates()*.

  Changed in version 21.9: Accepts any `collections.abc.Sequence` as input instead of just a list
- ***drop_pending_updates*** (bool, optional) – Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is `False`.

  Added 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*, *unix=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.

> **Important**
>
> If you want to use this method, you must install PTB with the optional requirement `webhooks`, i.e.
>
> ```
> pip install "python-telegram-bot[webhooks]"
> ```

See also

Webhooks

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`.

  Added in version 13.4.
- ***bootstrap_retries*** (`int`, optional) – Whether the bootstrapping phase of 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`.

  Added in version 13.4.
- ***allowed_updates*** (Sequence[`str`], optional) – Passed to *telegram.Bot.set_webhook()*. Defaults to `None`.

  Changed in version 21.9: Accepts any `collections.abc.Sequence` as input instead of just a list
- ***max_connections*** (`int`, optional) – Passed to *telegram.Bot.set_webhook()*. Defaults to `40`.

  Added 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.

Added in version 20.0.

- ***unix*** (pathlib.Path | str | socket.socket, optional) – Can be either:
  - the path to the unix socket file as pathlib.Path or str. This will be passed to tornado.netutil.bind_unix_socket to create the socket. If the Path does not exist, the file will be created.
  - or the socket itself. This option allows you to e.g. restrict the permissions of the socket for improved security. Note that you need to pass the correct family, type and socket options yourself.

Caution

This parameter is a replacement for the default TCP bind. Therefore, it is mutually exclusive with *listen* and *port*. When using this param, you must also run a reverse proxy to the unix socket and set the appropriate *webhook_url*.

Added in version 20.8.

Changed in version 21.1: Added support to pass a socket instance itself.

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.

6.2.13 Handlers

BaseHandler

`class telegram.ext.BaseHandler(callback, block=True)`

Bases: ABC, typing.Generic

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 three 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*.

3. The return type of the *callback* function accepted by this handler.

See also

Types of Handlers

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

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.

> **See also**
>
> Concurrency

callback

The callback function for this handler.

Type
coroutine function

block

Determines whether the callback will run in a blocking way.

Type
bool

__repr__()

Give a string representation of the handler in the form `ClassName[callback=...]`.

As this class doesn't implement `object.__str__()`, the default implementation will be used, which is equivalent to *__repr__()*.

Returns
str

abstractmethod 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.

BusinessConnectionHandler

class telegram.ext.**BusinessConnectionHandler**(*callback*, *user_id=None*, *username=None*, *block=True*)

Bases: *telegram.ext.BaseHandler*

Handler class to handle Telegram *Business Connections*.

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

Added in version 21.1.

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)
```

- ***user_id*** (int | Collection[int], optional) – Filters requests to allow only those which are from the specified user ID(s).
- ***username*** (str | Collection[str], optional) – Filters requests to allow only those which are from the specified username(s).
- ***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.

See also

Concurrency

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

BusinessMessagesDeletedHandler

class telegram.ext.**BusinessMessagesDeletedHandler**(*callback*, *chat_id=None*, *username=None*, *block=True*)

Bases: *telegram.ext.BaseHandler*

Handler class to handle *deleted Telegram Business messages*.

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

Added in version 21.1.

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)
```

- ***chat_id*** (int | Collection[int], optional) – Filters requests to allow only those which are from the specified chat ID(s).
- ***username*** (str | Collection[str], optional) – Filters requests to allow only those which are from the specified username(s).
- ***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.

> **See also**
>
> Concurrency

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

CallbackQueryHandler

class telegram.ext.**CallbackQueryHandler**(*callback*, *pattern=None*, *game_pattern=None*, *block=True*)

Bases: *telegram.ext.BaseHandler*

Handler 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*.
>
>   Added in version 13.6.
> - If neither *pattern* nor *game_pattern* is set, *any* CallbackQuery will be handled. If only *pattern* is set, queries with *game_short_name* will *not* be considered and vice versa. If both patterns are set, queries with either :attr: *~telegram.CallbackQuery.game_short_name* or *data* matching the defined pattern will be handled
>
>   Added in version 21.5.

> **Warning**
>
> When setting *block* to False, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

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.

  See also

  Arbitrary callback_data

  Changed in version 13.6: Added support for arbitrary callback data.

- ***game_pattern*** (str | re.Pattern | optional) – Pattern to test *telegram.CallbackQuery.game_short_name* against. If a string or a regex pattern is passed, re.match() is used on *telegram.CallbackQuery.game_short_name* to determine if an update should be handled by this handler.

  Added in version 21.5.

- ***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.

> **See also**
>
> Concurrency

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

game_pattern

Optional. Regex pattern to test *telegram.CallbackQuery.game_short_name*

Type
re.Pattern

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.

ChatBoostHandler

Added in version 20.8.

class telegram.ext.**ChatBoostHandler**(*callback*, *chat_boost_types=-1*, *chat_id=None*, *chat_username=None*, *block=True*)

Bases: *telegram.ext.BaseHandler*

Handler class to handle Telegram updates that contain a chat boost.

> **Warning**
>
> When setting *block* to False, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

Added in version 20.8.

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*.

- ***chat_boost_types*** (int, optional) – Pass one of *CHAT_BOOST*, *REMOVED_CHAT_BOOST* or *ANY_CHAT_BOOST* to specify if this handler should handle only updates with *telegram.Update.chat_boost*, *telegram.Update.removed_chat_boost* or both. Defaults to *CHAT_BOOST*.
- ***chat_id*** (int | Collection[int], optional) – Filters reactions to allow only those which happen in the specified chat ID(s).
- ***chat_username*** (str | Collection[str], optional) – Filters reactions to allow only those which happen in the specified username(s).
- ***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.

See also

Concurrency

callback

The callback function for this handler.

Type

coroutine function

chat_boost_types

Optional. Specifies if this handler should handle only updates with *telegram.Update.chat_boost*, *telegram.Update.removed_chat_boost* or both.

Type

int

block

Determines whether the callback will run in a blocking way.

Type
bool

ANY_CHAT_BOOST = 1

Used as a constant to handle both *telegram.Update.chat_boost* and *telegram.Update.removed_chat_boost*.

Type
int

CHAT_BOOST = -1

Used as a constant to handle only *telegram.Update.chat_boost*.

Type
int

REMOVED_CHAT_BOOST = 0

Used as a constant to handle only *telegram.Update.removed_chat_boost*.

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

ChatJoinRequestHandler

class telegram.ext.**ChatJoinRequestHandler**(*callback*, *chat_id=None*, *username=None*, *block=True*)

Bases: *telegram.ext.BaseHandler*

Handler class to handle Telegram updates that contain *telegram.Update.chat_join_request*.

Note

If neither of *username* and the *chat_id* are passed, this handler accepts *any* join request. Otherwise, this handler accepts all requests to join chats for which the chat ID is listed in *chat_id* or the username is listed in *username*, or both.

Added in version 20.0.

Warning

When setting *block* to False, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*

- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

Added 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*.

- ***chat_id*** (int | Collection[int], optional) – Filters requests to allow only those which are asking to join the specified chat ID(s).

  Added in version 20.0.

- ***username*** (str | Collection[str], optional) – Filters requests to allow only those which are asking to join the specified username(s).

  Added in version 20.0.

- ***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.

See also

Concurrency

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

ChatMemberHandler

class telegram.ext.**ChatMemberHandler**(*callback*, *chat_member_types=-1*, *block=True*, *chat_id=None*)

Bases: *telegram.ext.BaseHandler*

Handler 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.

Examples

Chat Member Bot

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

Added 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`.

> **See also**
>
> Concurrency

- ***chat_id*** (int | Collection[int], optional) – Filters chat member updates from specified chat ID(s) only. .. versionadded:: 21.3

callback

The callback function for this handler.

Type
coroutine function

chat_member_types

Optional. Specifies if this handler should handle only updates with *telegram.Update.my_chat_member*, *telegram.Update.chat_member* or both.

Type
int

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

ChosenInlineResultHandler

class telegram.ext.**ChosenInlineResultHandler**(*callback*, *block=True*, *pattern=None*)

Bases: *telegram.ext.BaseHandler*

Handler 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.

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

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`.

  See also

  Concurrency

- ***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*.

  Added 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.

Added 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*.

CommandHandler

class telegram.ext.**CommandHandler**(*command*, *callback*, *filters=None*, *block=True*, *has_args=None*)

Bases: *telegram.ext.BaseHandler*

Handler class to handle Telegram commands.

Commands are Telegram messages that start with a *telegram.MessageEntity.BOT_COMMAND* (so 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.

Note

If you want to support a different entity in the beginning, e.g. if a command message is wrapped in a *telegram.MessageEntity.CODE*, use the *telegram.ext.PrefixHandler*.

Warning

When setting *block* to False, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

Examples

- *Timer Bot*
- *Error Handler Bot*

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

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 for *telegram.BotCommand.command*.
- ***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`.

  See also

  Concurrency

- ***has_args*** (`bool` | `int`, optional) – Determines whether the command handler should process the update or not. If `True`, the handler will process any non-zero number of args.

If `False`, the handler will only process if there are no args. if `int`, the handler will only process if there are exactly that many args. Defaults to `None`, which means the handler will process any or no args.

Added in version 20.5.

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`

has_args

Optional argument, otherwise all implementations of *CommandHandler* will break. Defaults to `None`, which means the handler will process any args or no args.

Added in version 20.5.

Type
`bool` | `int` | None

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.

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.ApplicationBuilder.concurrent_updates* should be set to `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*.

Examples

- *Conversation Bot*
- *Conversation Bot 2*
- *Nested Conversation Bot*
- *Persistent Conversation Bot*

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

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. Default is `False`.
- ***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*.

Caution

- This feature relies on the *telegram.ext.Application.job_queue* being set and hence requires that the dependencies that *telegram.ext.JobQueue* relies on are installed.
- 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)

See also

Concurrency

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

__repr__()

Give a string representation of the ConversationHandler in the form `ConversationHandler[name=..., states={...}]`.

If there are more than 3 states, only the first 3 states are listed.

As this class doesn't implement `object.__str__()`, the default implementation will be used, which is equivalent to *__repr__()*.

Returns
str

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*]]

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(...)`.

Changed in version 22.0: Removed deprecated attribute *CHAT*.

`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()*.

Added in version 13.6.

`telegram.ext.filters.`**`AUDIO = filters.AUDIO`**

Messages that contain *telegram.Message.audio*.

`telegram.ext.filters.`**`BOOST_ADDED = filters.BOOST_ADDED`**

Messages that contain *telegram.Message.boost_added*.

`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.`**`CHECKLIST = filters.CHECKLIST`**

Messages that contain *telegram.Message.checklist*.

Added in version 22.3.

`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*.

`telegram.ext.filters.`**`DIRECT_MESSAGES = filters.DIRECT_MESSAGES`**

Filter chats which are the direct messages for a channel.

See also

telegram.Chat.is_direct_messages

Added in version 22.4.

`telegram.ext.filters.`**`EFFECT_ID = filters.EFFECT_ID`**

Messages that contain *telegram.Message.effect_id*.

Added in version 21.3.

telegram.ext.filters.**FORUM = filters.FORUM**

Messages that are from a forum (topics enabled) chat.

Added in version 22.4.

telegram.ext.filters.**FORWARDED = filters.FORWARDED**

Messages that contain *telegram.Message.forward_origin*.

Changed in version 20.8: Now based on *telegram.Message.forward_origin* instead of `telegram.Message.forward_date`.

telegram.ext.filters.**GAME = filters.GAME**

Messages that contain *telegram.Message.game*.

telegram.ext.filters.**GIVEAWAY = filters.GIVEAWAY**

Messages that contain *telegram.Message.giveaway*.

telegram.ext.filters.**GIVEAWAY_WINNERS = filters.GIVEAWAY_WINNERS**

Messages that contain *telegram.Message.giveaway_winners*.

telegram.ext.filters.**HAS_MEDIA_SPOILER = filters.HAS_MEDIA_SPOILER**

Messages that contain *telegram.Message.has_media_spoiler*.

Added in version 20.0.

telegram.ext.filters.**HAS_PROTECTED_CONTENT = filters.HAS_PROTECTED_CONTENT**

Messages that contain *telegram.Message.has_protected_content*.

Added 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*.

Added in version 13.9.

telegram.ext.filters.**IS_FROM_OFFLINE = filters.IS_FROM_OFFLINE**

Messages that contain *telegram.Message.is_from_offline*.

Added in version 21.1.

telegram.ext.filters.**IS_TOPIC_MESSAGE = filters.IS_TOPIC_MESSAGE**

Messages that contain *telegram.Message.is_topic_message*.

Added in version 20.0.

telegram.ext.filters.**LIVE_PHOTO = filters.LIVE_PHOTO**

Messages that contain *telegram.Message.live_photo*.

Added in version 22.8.

telegram.ext.filters.**LOCATION = filters.LOCATION**

Messages that contain *telegram.Message.location*.

telegram.ext.filters.**PAID_MEDIA = filters.PAID_MEDIA**

Messages that contain *telegram.Message.paid_media*.

Added in version 21.4.

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.**PREMIUM_USER = filters.PREMIUM_USER**

This filter filters *any* message from a *Telegram Premium user* as *telegram.Update.effective_user*.

Added in version 20.0.

telegram.ext.filters.**REPLY = filters.REPLY**

Messages that contain *telegram.Message.reply_to_message*.

telegram.ext.filters.**REPLY_TO_STORY = filters.REPLY_TO_STORY**

Messages that contain *telegram.Message.reply_to_story*.

telegram.ext.filters.**SENDER_BOOST_COUNT = filters.SENDER_BOOST_COUNT**

Messages that contain *telegram.Message.sender_boost_count*.

telegram.ext.filters.**STORY = filters.STORY**

Messages that contain *telegram.Message.story*.

Added in version 20.5.

telegram.ext.filters.**SUCCESSFUL_PAYMENT = filters.SUCCESSFUL_PAYMENT**

Messages that contain *telegram.Message.successful_payment*.

telegram.ext.filters.**SUGGESTED_POST_INFO = filters.SUGGESTED_POST_INFO**

Messages that contain *telegram.Message.suggested_post_info*.

Added in version 22.4.

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)`.

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*.

Added in version 20.0.

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.

See also

ViaBot

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.**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.

Available In

- *telegram.ext.CommandHandler.filters*
- *telegram.ext.MessageHandler.filters*
- *telegram.ext.PrefixHandler.filters*

Added 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).

__and__(*other*)

Defines *AND* bitwise operator for *BaseFilter* object. The combined filter accepts an update only if it is accepted by both filters. For example, `filters.PHOTO & filters.CAPTION` will only accept messages that contain both a photo and a caption.

Returns
BaseFilter

__or__(*other*)

Defines *OR* bitwise operator for *BaseFilter* object. The combined filter accepts an update only if it is accepted by any of the filters. For example, `filters.PHOTO | filters.CAPTION` will only accept messages that contain photo or caption or both.

Returns
BaseFilter

__xor__(*other*)

Defines *XOR* bitwise operator for *BaseFilter* object. The combined filter accepts an update only if it is accepted by any of the filters and not both of them. For example, `filters.PHOTO ^ filters.CAPTION` will only accept messages that contain photo or caption, not both of them.

Returns
BaseFilter

__invert__()

Defines *NOT* bitwise operator for *BaseFilter* object. The combined filter accepts an update only if it is accepted by any of the filters. For example, `~ filters.PHOTO` will only accept messages that do not contain photo.

Returns
BaseFilter

__repr__()

Gives name for this filter.

See also

name()

Return type
str

property data_filter

Whether this filter is a data filter.

Type
bool

property name

Name for this filter.

Type
str

check_update(*update*)

Checks if the specified update should be handled by this filter.

Changed in version 21.1: This filter now also returns `True` if the update contains *business_message* or *edited_business_message*.

Changed in version 22.8: This filter now also returns `True` if the update contains *guest_message*.

Parameters

update (*telegram.Update*) – The update to check.

Returns

`True` if the update contains one of *channel_post*, *message*, *edited_channel_post*, *edited_message*, *telegram.Update.business_message*, *telegram.Update.edited_business_message*, *telegram.Update.guest_message*, or `False` otherwise.

Return type

`bool`

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.

property name

Name for this filter.

Type

str

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.

Added 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/vnd.wave')

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*.

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_origin* and in particular

- *telegram.MessageOriginUser.sender_user*
- *telegram.MessageOriginChat.sender_chat*
- *telegram.MessageOriginChannel.chat*

Added in version 13.5.

Changed in version 20.8: Was previously based on `telegram.Message.forward_from` and `telegram.Message.forward_from_chat`.

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 *telegram.Message.forward_origin* will be of type *telegram.MessageOriginHiddenUser*. 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.

property name

Name for this filter.

Type
str

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.**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.**Mention**(*mentions*)

Bases: *telegram.ext.filters.MessageFilter*

Messages containing mentions of specified users or chats.

Examples

```
MessageHandler(filters.Mention("username"), callback)
MessageHandler(filters.Mention(["@username", 123456]), callback)
```

Added in version 20.7.

Parameters

mentions (int | str | *telegram.User* | Collection[int | str | *telegram.User*]) – Specifies the users and chats to filter for. Messages that do not mention at least one of the specified users or chats will not be handled. Leading '@' s in usernames will be discarded.

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.

See also

Advanced Filters

Available In

- *telegram.ext.CommandHandler.filters*
- *telegram.ext.MessageHandler.filters*
- *telegram.ext.PrefixHandler.filters*

check_update(*update*)

Checks if the specified update should be handled by this filter by passing *effective_message* to *filter()*.

Parameters

update (*telegram.Update*) – The update to check.

Returns

If the update should be handled by this filter, returns `True` or a dict with lists, in case the filter is a data filter. If the update should not be handled by this filter, `False` or `None`.

Return type

`bool` | dict[`str`, `list`] | `None`

abstractmethod filter(*message*)

This method must be overwritten.

Parameters

message (*telegram.Message*) – The message that is tested.

Returns

`dict` or `bool`

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.

See also

Types of Handlers

Parameters

pattern (`str` | `re.Pattern`) – The regex pattern.

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.

property name
Name for this filter.

Type
str

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.

Changed in version 22.0: Removed deprecated attribute *USER_SHARED*.

ALL = filters.StatusUpdate.ALL

Messages that contain any of the below.

CHAT_BACKGROUND_SET = filters.StatusUpdate.CHAT_BACKGROUND_SET

Messages that contain *telegram.Message.chat_background_set*.

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*.

CHAT_OWNER_CHANGED = filters.StatusUpdate.CHAT_OWNER_CHANGED

Messages that contain *telegram.Message.chat_owner_changed*.

Added in version 22.7.

CHAT_OWNER_LEFT = filters.StatusUpdate.CHAT_OWNER_LEFT

Messages that contain *telegram.Message.chat_owner_left*.

Added in version 22.7.

CHAT_SHARED = filters.StatusUpdate.CHAT_SHARED

Messages that contain *telegram.Message.chat_shared*.

Added in version 20.1.

CHECKLIST_TASKS_ADDED = filters.StatusUpdate.CHECKLIST_TASKS_ADDED

Messages that contain *telegram.Message.checklist_tasks_added*.

Added in version 22.3.

CHECKLIST_TASKS_DONE = filters.StatusUpdate.CHECKLIST_TASKS_DONE

Messages that contain *telegram.Message.checklist_tasks_done*.

Added in version 22.3.

CONNECTED_WEBSITE = filters.StatusUpdate.CONNECTED_WEBSITE

Messages that contain *telegram.Message.connected_website*.

DIRECT_MESSAGE_PRICE_CHANGED = filters.StatusUpdate.DIRECT_MESSAGE_PRICE_CHANGED

Messages that contain *telegram.Message.direct_message_price_changed*.

Added in version 22.3.

DELETE_CHAT_PHOTO = filters.StatusUpdate.DELETE_CHAT_PHOTO

Messages that contain *telegram.Message.delete_chat_photo*.

FORUM_TOPIC_CLOSED = filters.StatusUpdate.FORUM_TOPIC_CLOSED

Messages that contain *telegram.Message.forum_topic_closed*.

Added in version 20.0.

FORUM_TOPIC_CREATED = filters.StatusUpdate.FORUM_TOPIC_CREATED

Messages that contain *telegram.Message.forum_topic_created*.

Added in version 20.0.

FORUM_TOPIC_EDITED = filters.StatusUpdate.FORUM_TOPIC_EDITED

Messages that contain *telegram.Message.forum_topic_edited*.

Added in version 20.0.

FORUM_TOPIC_REOPENED = filters.StatusUpdate.FORUM_TOPIC_REOPENED

Messages that contain *telegram.Message.forum_topic_reopened*.

Added in version 20.0.

GENERAL_FORUM_TOPIC_HIDDEN = filters.StatusUpdate.GENERAL_FORUM_TOPIC_HIDDEN

Messages that contain *telegram.Message.general_forum_topic_hidden*.

Added in version 20.0.

GENERAL_FORUM_TOPIC_UNHIDDEN = filters.StatusUpdate.GENERAL_FORUM_TOPIC_UNHIDDEN

Messages that contain *telegram.Message.general_forum_topic_unhidden*.

Added in version 20.0.

GIFT = filters.StatusUpdate.GIFT

Messages that contain *telegram.Message.gift*.

Added in version 22.1.

GIFT_UPGRADE_SENT = filters.StatusUpdate.GIFT_UPGRADE_SENT

Messages that contain *telegram.Message.gift_upgrade_sent*.

Added in version 22.6.

GIVEAWAY_CREATED = filters.StatusUpdate.GIVEAWAY_CREATED

Messages that contain *telegram.Message.giveaway_created*.

Added in version 20.8.

GIVEAWAY_COMPLETED = filters.StatusUpdate.GIVEAWAY_COMPLETED

Messages that contain *telegram.Message.giveaway_completed*. .. versionadded:: 20.8

LEFT_CHAT_MEMBER = filters.StatusUpdate.LEFT_CHAT_MEMBER

Messages that contain *telegram.Message.left_chat_member*.

MANAGED_BOT_CREATED = filters.StatusUpdate.MANAGED_BOT_CREATED

Messages that contain *telegram.Message.managed_bot_created*.

MESSAGE_AUTO_DELETE_TIMER_CHANGED = filters.StatusUpdate.MESSAGE_AUTO_DELETE_TIMER_CHANGED

Messages that contain *telegram.Message.message_auto_delete_timer_changed*

Added 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*.

PAID_MESSAGE_PRICE_CHANGED = filters.StatusUpdate.PAID_MESSAGE_PRICE_CHANGED

Messages that contain *telegram.Message.paid_message_price_changed*.

Added in version 22.1.

PINNED_MESSAGE = filters.StatusUpdate.PINNED_MESSAGE

Messages that contain *telegram.Message.pinned_message*.

POLL_OPTION_ADDED = filters.StatusUpdate.POLL_OPTION_ADDED

Messages that contain *telegram.Message.poll_option_added*. .. versionadded:: 22.8

POLL_OPTION_DELETED = filters.StatusUpdate.POLL_OPTION_DELETED

Messages that contain *telegram.Message.poll_option_deleted*. .. versionadded:: 22.8

PROXIMITY_ALERT_TRIGGERED = filters.StatusUpdate.PROXIMITY_ALERT_TRIGGERED

Messages that contain *telegram.Message.proximity_alert_triggered*.

REFUNDED_PAYMENT = filters.StatusUpdate.REFUNDED_PAYMENT

Messages that contain *telegram.Message.refunded_payment*. .. versionadded:: 21.4

SUGGESTED_POST_APPROVAL_FAILED = filters.StatusUpdate.SUGGESTED_POST_APPROVAL_FAILED

Messages that contain *telegram.Message.suggested_post_approval_failed*. .. versionadded:: 22.4

SUGGESTED_POST_APPROVED = filters.StatusUpdate.SUGGESTED_POST_APPROVED

Messages that contain *telegram.Message.suggested_post_approved*. .. versionadded:: 22.4

SUGGESTED_POST_DECLINED = filters.StatusUpdate.SUGGESTED_POST_DECLINED

Messages that contain *telegram.Message.suggested_post_declined*. .. versionadded:: 22.4

SUGGESTED_POST_PAID = filters.StatusUpdate.SUGGESTED_POST_PAID

Messages that contain *telegram.Message.suggested_post_paid*. .. versionadded:: 22.4

SUGGESTED_POST_REFUNDED = filters.StatusUpdate.SUGGESTED_POST_REFUNDED

Messages that contain *telegram.Message.suggested_post_refunded*. .. versionadded:: 22.4

UNIQUE_GIFT = filters.StatusUpdate.UNIQUE_GIFT

Messages that contain *telegram.Message.unique_gift*.

Added in version 22.1.

USERS_SHARED = filters.StatusUpdate.USERS_SHARED

Messages that contain *telegram.Message.users_shared*.

Added in version 20.8.

VIDEO_CHAT_ENDED = filters.StatusUpdate.VIDEO_CHAT_ENDED

Messages that contain *telegram.Message.video_chat_ended*.

Added 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*.

Added 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*.

Added 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*.

Added 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*.

Added in version 20.0.

WRITE_ACCESS_ALLOWED = filters.StatusUpdate.WRITE_ACCESS_ALLOWED

Messages that contain *telegram.Message.write_access_allowed*.

Added in version 20.0.

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*.

Added 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*.

Added in version 20.0.

VIDEO = filters.Sticker.VIDEO

Messages that contain *telegram.Message.sticker* and is a *video sticker*.

Added in version 20.0.

PREMIUM = filters.Sticker.PREMIUM

Messages that contain *telegram.Message.sticker* and have a *premium animation*.

Added in version 20.0.

class telegram.ext.filters.**SuccessfulPayment**(*invoice_payloads=None*)

Bases: *telegram.ext.filters.MessageFilter*

Successful Payment Messages. If a list of invoice payloads is passed, it filters messages to only allow those whose *invoice_payload* is appearing in the given list.

Examples

MessageHandler(filters.SuccessfulPayment(['Custom-Payload']), callback_method)

See also

telegram.ext.filters.SUCCESSFUL_PAYMENT

Parameters

invoice_payloads (list[str] | tuple[str], optional) – Which invoice payloads to allow. Only exact matches are allowed. If not specified, will allow any invoice payload.

Added in version 20.8.

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.

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.

Available In

- *telegram.ext.CommandHandler.filters*
- *telegram.ext.MessageHandler.filters*
- *telegram.ext.PrefixHandler.filters*

check_update(*update*)

Checks if the specified update should be handled by this filter.

Parameters

update (*telegram.Update*) – The update to check.

Returns

If the update should be handled by this filter, returns `True` or a dict with lists, in case the filter is a data filter. If the update should not be handled by this filter, `False` or `None`.

Return type

`bool` | dict[`str`, `list`] | `None`

abstractmethod 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 *telegram.Update.edited_message*, *telegram.Update.edited_channel_post*, or *telegram.Update.edited_business_message*.

Added in version 20.0.

Changed in version 21.1: Added *telegram.Update.edited_business_message* to the filter.

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*.

BUSINESS_MESSAGE = filters.UpdateType.BUSINESS_MESSAGE

Updates with *telegram.Update.business_message*.

Added in version 21.1.

EDITED_BUSINESS_MESSAGE = filters.UpdateType.EDITED_BUSINESS_MESSAGE

Updates with *telegram.Update.edited_business_message*.

Added in version 21.1.

BUSINESS_MESSAGES = filters.UpdateType.BUSINESS_MESSAGES

Updates with either *telegram.Update.business_message* or *telegram.Update.edited_business_message*.

Added in version 21.1.

GUEST_MESSAGE = filters.UpdateType.GUEST_MESSAGE

Updates with *telegram.Update.guest_message*.

Added in version 22.8.

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.

property name

Name for this filter.

Type
str

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.

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)
```

See also

VIA_BOT

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.

property name

Name for this filter.

Type

str

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.

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.

Examples

Inline Bot

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

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.

  See also

  Concurrency

- ***chat_types*** (list[str], optional) – List of allowed chat types. If passed, will only handle inline queries with the appropriate *telegram.InlineQuery.chat_type*.

  Added 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.

Added 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.

ManagedBotUpdatedHandler

class telegram.ext.**ManagedBotUpdatedHandler**(*callback*, *user_id=None*, *username=None*, *block=True*)

Bases: *telegram.ext.BaseHandler*

Handler class to handle *updated Telegram Managed Bots*.

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

Added in version 22.8.

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)
```

- ***user_id*** (int | Collection[int], optional) – Filters requests to allow only those which are from the specified user ID(s). The user ID(s) can be either the creator of the managed bot or the bot itself.
- ***username*** (str | Collection[str], optional) – Filters requests to allow only those which are from the specified username(s). The username(s) can be either the creator of the managed bot or the bot itself.
- ***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.

See also

Concurrency

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

MessageHandler

class telegram.ext.**MessageHandler**(*filters*, *callback*, *block=True*)

Bases: *telegram.ext.BaseHandler*

Handler 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.

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

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). Passing `None` is a shortcut to passing *telegram.ext.filters.ALL*.

  See also

  Advanced Filters

- ***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`.

  See also

  Concurrency

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*.

MessageReactionHandler

class telegram.ext.**MessageReactionHandler**(*callback*, *chat_id=None*, *chat_username=None*, *user_id=None*, *user_username=None*, *message_reaction_types=1*, *block=True*)

Bases: *telegram.ext.BaseHandler*

Handler class to handle Telegram updates that contain a message reaction.

Note

The following rules apply to both `username` and the `chat_id` param groups, respectively:

- **If none of them are passed, the handler does not filter the update for that specific** attribute.
- **If a chat ID or a username is passed, the updates will be filtered with that** specific attribute.
- **If a chat ID and a username are passed, an update containing any of them will be** filtered.
- *telegram.MessageReactionUpdated.actor_chat* is *not* considered for *user_id* and *user_username* filtering.

Warning

When setting *block* to `False`, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

Added in version 20.8.

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*.

- ***message_reaction_types*** (int, optional) – Pass one of *MESSAGE_REACTION_UPDATED*, *MESSAGE_REACTION_COUNT_UPDATED* or *MESSAGE_REACTION* to specify if this handler should handle only updates with *telegram.Update.message_reaction*, *telegram.Update.message_reaction_count* or both. Defaults to *MESSAGE_REACTION*.

- ***chat_id*** (int | Collection[int], optional) – Filters reactions to allow only those which happen in the specified chat ID(s).

- ***chat_username*** (str | Collection[str], optional) – Filters reactions to allow only those which happen in the specified username(s).

- ***user_id*** (int | Collection[int], optional) – Filters reactions to allow only those which are set by the specified chat ID(s) (this can be the chat itself in the case of anonymous users, see the *telegram.MessageReactionUpdated.actor_chat*).

- ***user_username*** (str | Collection[str], optional) – Filters reactions to allow only those which are set by the specified username(s) (this can be the chat itself in the case of anonymous users, see the *telegram.MessageReactionUpdated.actor_chat*).

- ***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.

> **See also**
>
> Concurrency

callback

The callback function for this handler.

Type
coroutine function

message_reaction_types

Optional. Specifies if this handler should handle only updates with *telegram.Update.message_reaction*, *telegram.Update.message_reaction_count* or both.

Type
int

block

Determines whether the callback will run in a blocking way.

Type
bool

MESSAGE_REACTION = 1

Used as a constant to handle both *telegram.Update.message_reaction* and *telegram.Update.message_reaction_count*.

Type
int

MESSAGE_REACTION_COUNT_UPDATED = 0

Used as a constant to handle only *telegram.Update.message_reaction_count*.

Type
int

MESSAGE_REACTION_UPDATED = -1

Used as a constant to handle only *telegram.Update.message_reaction*.

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

PaidMediaPurchasedHandler

class telegram.ext.**PaidMediaPurchasedHandler**(*callback*, *user_id=None*, *username=None*, *block=True*)

Bases: *telegram.ext.BaseHandler*

Handler class to handle Telegram *purchased paid media*.

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

Added in version 21.6.

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)
```

- ***user_id*** (int | Collection[int], optional) – Filters requests to allow only those which are from the specified user ID(s).
- ***username*** (str | Collection[str], optional) – Filters requests to allow only those which are from the specified username(s).

- ***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.

  See also

  Concurrency

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

PollAnswerHandler

class telegram.ext.**PollAnswerHandler**(*callback*, *block=True*)

Bases: *telegram.ext.BaseHandler*

Handler 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.

Examples

Poll Bot

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

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.

See also

Concurrency

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

PollHandler

class telegram.ext.**PollHandler**(*callback*, *block=True*)

Bases: *telegram.ext.BaseHandler*

Handler 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.

Examples

Poll Bot

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

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.

See also

Concurrency

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

PreCheckoutQueryHandler

class telegram.ext.**PreCheckoutQueryHandler**(*callback*, *block=True*, *pattern=None*)

Bases: *telegram.ext.BaseHandler*

Handler 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.

Examples

Payment Bot

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

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.

> **See also**
>
> Concurrency

- ***pattern*** (`str` | `re.Pattern`, optional) – Optional. Regex pattern to test *telegram.PreCheckoutQuery.invoice_payload* against.

  Added in version 20.8.

callback

The callback function for this handler.

Type
coroutine function

block

Determines whether the callback will run in a blocking way..

Type
`bool`

pattern

Optional. Regex pattern to test *telegram.PreCheckoutQuery.invoice_payload* against.

Added in version 20.8.

Type
`str` | `re.Pattern`, optional

check_update(*update*)

Determines whether an update should be passed to this handler's *callback*.

Parameters
update (*telegram.Update* | `object`) – Incoming update.

Returns
`bool`

PrefixHandler

class telegram.ext.**PrefixHandler**(*prefix*, *command*, *callback*, *filters=None*, *block=True*)

Bases: *telegram.ext.BaseHandler*

Handler 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.

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

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.

See also

Concurrency

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.

ShippingQueryHandler

class telegram.ext.**ShippingQueryHandler**(*callback*, *block=True*)

Bases: *telegram.ext.BaseHandler*

Handler 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.

Examples

Payment Bot

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

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`.

  See also

  Concurrency

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

StringCommandHandler

class telegram.ext.**StringCommandHandler**(*command*, *callback*, *block=True*)

Bases: *telegram.ext.BaseHandler*

Handler 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.

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

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: str, 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.

> **See also**
>
> Concurrency

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.

StringRegexHandler

class telegram.ext.**StringRegexHandler**(*pattern*, *callback*, *block=True*)

Bases: *telegram.ext.BaseHandler*

Handler 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.

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

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: str, 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`.

See also

Concurrency

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.

TypeHandler

class telegram.ext.**TypeHandler**(*type*, *callback*, *strict=False*, *block=True*)

Bases: *telegram.ext.BaseHandler*

Handler 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.

Use In

- *telegram.ext.Application.add_handler()*
- *telegram.ext.Application.add_handlers()*
- *telegram.ext.Application.remove_handler()*

Available In

- *telegram.ext.ConversationHandler.entry_points*
- *telegram.ext.ConversationHandler.fallbacks*
- *telegram.ext.ConversationHandler.states*

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: object, 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.

  See also

  Concurrency

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

6.2.14 Persistence

BasePersistence

class telegram.ext.**BasePersistence**(*store_data=None*, *update_interval=60*)

Bases: ABC, typing.Generic

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()*.

See also

Architecture Overview, Making Your Bot Persistent

Use In

telegram.ext.ApplicationBuilder.persistence()

Available In

telegram.ext.Application.persistence

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.

  Added 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

abstractmethod async drop_chat_data(*chat_id*)

Will be called by the *telegram.ext.Application*, when using *drop_chat_data()*.

Added in version 20.0.

Parameters
chat_id (`int`) – The chat id to delete from the persistence.

abstractmethod async drop_user_data(*user_id*)

Will be called by the *telegram.ext.Application*, when using *drop_user_data()*.

Added in version 20.0.

Parameters
user_id (`int`) – The user id to delete from the persistence.

abstractmethod 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()`.

abstractmethod 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*]

abstractmethod 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.

Added 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.

abstractmethod 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*]

abstractmethod 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

abstractmethod 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*]

abstractmethod 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.

Tip

This method is expected to edit the object *bot_data* in-place instead of returning a new object.

Warning

When using *concurrent_updates()*, this method may be called while a handler callback is still running. This might lead to race conditions.

Added 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.

abstractmethod 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.

Tip

This method is expected to edit the object *chat_data* in-place instead of returning a new object.

Warning

When using *concurrent_updates()*, this method may be called while a handler callback is still running. This might lead to race conditions.

Added 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.

abstractmethod 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.

Tip

This method is expected to edit the object *user_data* in-place instead of returning a new object.

> **Warning**
>
> When using *concurrent_updates()*, this method may be called while a handler callback is still running. This might lead to race conditions.

Added 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*.

abstractmethod 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*.

abstractmethod async update_callback_data(*data*)

Will be called by the *telegram.ext.Application* after a handler has handled an update.

Added 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*.

abstractmethod 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].

abstractmethod 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.

Added in version 20.0.

Type
float

abstractmethod 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].

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().

See also

Making Your Bot Persistent

Use In

telegram.ext.ApplicationBuilder.persistence()

Available In

telegram.ext.Application.persistence

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 "".

  Added 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.

  Added 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.

Added 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.

Added 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*.

Added 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*.

Added in version 20.0.

Parameters
user_id (int) – The user id to delete from the persistence.

async flush()

Does nothing.

Added 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.

Added 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.

Added in version 13.6.

See also

telegram.ext.BasePersistence.refresh_bot_data()

async refresh_chat_data(*chat_id*, *chat_data*)

Does nothing.

Added in version 13.6.

See also

telegram.ext.BasePersistence.refresh_chat_data()

async refresh_user_data(*user_id*, *user_data*)

Does nothing.

Added 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).

Added 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

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*.

Available In

- *telegram.ext.BasePersistence.store_data*
- *telegram.ext.DictPersistence.store_data*

- *telegram.ext.PicklePersistence.store_data*

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

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.

Examples

Persistent Conversation Bot

See also

Making Your Bot Persistent

Use In

telegram.ext.ApplicationBuilder.persistence()

Available In

telegram.ext.Application.persistence

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.

  Added 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.

  Added 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

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`.

Type
`bool`

context_types

Container for the types used in the `context` interface.

Added 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.

Added 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.

Added 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`.

Added 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.

Added in version 13.6.

See also

telegram.ext.BasePersistence.refresh_bot_data()

async refresh_chat_data(*chat_id*, *chat_data*)

Does nothing.

Added in version 13.6.

See also

telegram.ext.BasePersistence.refresh_chat_data()

async refresh_user_data(*user_id*, *user_data*)

Does nothing.

Added 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.

Added 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].

6.2.15 Arbitrary Callback Data

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.

Important

If you want to use this class, you must install PTB with the optional requirement `callback-data`, i.e.

```
pip install "python-telegram-bot[callback-data]"
```

Examples

Arbitrary Callback Data Bot

See also

Architecture Overview, Arbitrary callback_data

Available In

telegram.ext.ExtBot.callback_data_cache

Added in version 13.6.

Changed in version 20.0: To use this class, PTB must be installed via `pip install "python-telegram-bot[callback-data]"`.

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

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)

load_persistence_data(*persistent_data*)

Loads data into the cache.

> **Warning**
> This method is not intended to be called by users directly.

Added in version 20.0.

Parameters
persistent_data (tuple[list[tuple[str, float, dict[str, object]]], dict[str, str]], optional) – Data to load, as returned by *telegram.ext.BasePersistence.get_callback_data()*.

property maxsize

The maximum size of the cache.

Changed in version 20.0: This property is now read-only.

Type
int

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.

InvalidCallbackData

class telegram.ext.**InvalidCallbackData**(*callback_data=None*)

Bases: *telegram.error.TelegramError*

Raised when the received callback data has been tampered with or deleted from cache.

Examples

Arbitrary Callback Data Bot

See also

Arbitrary callback_data

Added 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

__reduce__()
Defines how to serialize the exception for pickle. See object.__reduce__() for more info.

Returns
tuple

6.2.16 Rate Limiting

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.

See also
Architecture Overview, Avoiding Flood Limits

Use In
telegram.ext.ApplicationBuilder.rate_limiter()

Available In
telegram.ext.ExtBot.rate_limiter

Added in version 20.0.

abstractmethod async initialize()
Initialize resources used by this class. Must be implemented by a subclass.

abstractmethod 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

abstractmethod async shutdown()
Stop & clear resources used by this class. Must be implemented by a subclass.

AIORateLimiter

class telegram.ext.**AIORateLimiter**(*overall_max_rate=<FloodLimit.MESSAGES_PER_SECOND>*, *overall_time_period=1*, *group_max_rate=<FloodLimit.MESSAGES_PER_MINUTE_PER_GROUP>*, *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 or with *allow_paid_broadcast* set to True.

Tip

With Bot API 7.1 (PTB v27.1), Telegram introduced the parameter *allow_paid_broadcast*. This allows bots to send up to *1000* messages per second by paying a fee in Telegram Stars.

Changed in version 21.11: This class automatically takes the *allow_paid_broadcast* parameter into account and throttles the requests accordingly.

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.

See also

Avoiding Flood Limits

Use In

telegram.ext.ApplicationBuilder.rate_limiter()

Available In

telegram.ext.ExtBot.rate_limiter

Added 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_max_rate* 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.

6.3 Auxiliary modules

6.3.1 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.

Removed in version 22.3: Removed deprecated class `StarTransactions`. Please instead use *telegram.constants.Nanostar.VALUE*.

class telegram.constants.**AccentColor**(**values*)

Bases: Enum

This enum contains the available accent colors for *telegram.ChatFullInfo.accent_color_id*. The members of this enum are named tuples with the following attributes:

- `identifier` (int): The identifier of the accent color.
- `name` (str): Optional. The name of the accent color.
- `light_colors` (tuple[str]): Optional. The light colors of the accent color as HEX value.
- `dark_colors` (tuple[str]): Optional. The dark colors of the accent color as HEX value.

Since Telegram gives no exact specification for the accent colors, future accent colors might have a different data type.

Added in version 20.8.

COLOR_000 = (0, 'red', (), ())

Accent color 0. This color can be customized by app themes.

COLOR_001 = (1, 'orange', (), ())

Accent color 1. This color can be customized by app themes.

COLOR_002 = (2, 'purple/violet', (), ())

Accent color 2. This color can be customized by app themes.

COLOR_003 = (3, 'green', (), ())

Accent color 3. This color can be customized by app themes.

COLOR_004 = (4, 'cyan', (), ())

Accent color 4. This color can be customized by app themes.

COLOR_005 = (5, 'blue', (), ())

Accent color 5. This color can be customized by app themes.

COLOR_006 = (6, 'pink', (), ())

Accent color 6. This color can be customized by app themes.

COLOR_007 = (7, None, (14766162, 16363107), (16749440, 10039095))

Accent color 7. This contains two light colors

and two dark colors

COLOR_008 = (8, None, (14712875, 16434484), (15511630, 12801812))

Accent color 8. This contains two light colors

and two dark colors

COLOR_009 = (9, None, (10510323, 16027647), (13015039, 6173128))

Accent color 9. This contains two light colors

and two dark colors

COLOR_010 = (10, None, (2599184, 11000919), (11004782, 1474093))

Accent color 10. This contains two light colors

and two dark colors

COLOR_011 = (11, None, (2600142, 8579286), (4249808, 285823))

Accent color 11. This contains two light colors

and two dark colors

COLOR_012 = (12, None, (3379668, 8246256), (5423103, 742548))

Accent color 12. This contains two light colors

and two dark colors

COLOR_013 = (13, None, (14500721, 16760479), (16746150, 9320046))

Accent color 13. This contains two light colors

and two dark colors

COLOR_014 = (14, None, (2391021, 15747158, 16777215), (4170494, 15024719, 16777215))

Accent color 14. This contains three light colors

and three dark colors

COLOR_015 = (15, None, (14055202, 2007057, 16777215), (16748638, 3319079, 16777215))

Accent color 15. This contains three light colors

and three dark colors

COLOR_016 = (16, None, (1547842, 15223359, 16777215), (6738788, 13976655, 16777215))

Accent color 16. This contains three light colors

and three dark colors

COLOR_017 = (17, None, (2659503, 7324758, 16777215), (2276578, 4039232, 16777215))

Accent color 17. This contains three light colors

and three dark colors

COLOR_018 = (18, None, (826035, 16756117, 16770741), (2276578, 16750456, 16767595))

Accent color 18. This contains three light colors

and three dark colors

COLOR_019 = (19, None, (7821270, 16225808, 16768654), (9933311, 15889181, 16767833))

Accent color 19. This contains three light colors

and three dark colors

COLOR_020 = (20, None, (1410511, 15903517, 16777215), (4040427, 15639837, 16777215))

Accent color 20. This contains three light colors

and three dark colors

telegram.constants.**BOT_API_VERSION = '10.0'**

Telegram Bot API version supported by this version of *python-telegram-bot*. Also available as *telegram.__bot_api_version__*.

Added in version 13.4.

Type
str

telegram.constants.**BOT_API_VERSION_INFO = (10, 0)**

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__*.

Added in version 20.0.

class telegram.constants.**BackgroundFillLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.BackgroundFillGradient*. The enum members of this enumeration are instances of `int` and can be treated as such.

Added in version 21.2.

MAX_ROTATION_ANGLE = 359

Maximum value allowed for: *rotation_angle* parameter of *telegram.BackgroundFillGradient*

Type
int

class telegram.constants.**BackgroundFillType**(**values*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.BackgroundFill*. The enum members of this enumeration are instances of `str` and can be treated as such.

Added in version 21.2.

FREEFORM_GRADIENT = 'freeform_gradient'

A *telegram.BackgroundFill* with freeform_gradient fill.

Type
str

GRADIENT = 'gradient'

A *telegram.BackgroundFill* with gradient fill.

Type
str

SOLID = 'solid'

A *telegram.BackgroundFill* with solid fill.

Type
str

class telegram.constants.**BackgroundTypeLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.BackgroundTypeFill*, *telegram.BackgroundTypeWallpaper* and *telegram.BackgroundTypePattern*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 21.2.

MAX_DIMMING = 100

Maximum value allowed for:

- ***dark_theme_dimming*** **parameter of** *telegram.BackgroundTypeFill*
- ***dark_theme_dimming*** **parameter of** *telegram.BackgroundTypeWallpaper*

Type
int

MAX_INTENSITY = 100

Maximum value allowed for *intensity* parameter of *telegram.BackgroundTypePattern*

Type
int

class telegram.constants.**BackgroundTypeType**(**values*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.BackgroundType*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 21.2.

CHAT_THEME = 'chat_theme'

A *telegram.BackgroundType* with chat_theme background.

Type
str

FILL = 'fill'

A *telegram.BackgroundType* with fill background.

Type
str

PATTERN = 'pattern'

A *telegram.BackgroundType* with pattern background.

Type
str

WALLPAPER = 'wallpaper'

A *telegram.BackgroundType* with wallpaper background.

Type
str

class telegram.constants.**BaseInputMediaType**(**values*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.InputMedia*, *telegram.InputPollMedia* and *telegram.InputPollOptionMedia*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 22.8.

ANIMATION = 'animation'

Type of *telegram.InputMediaAnimation*.

Type
str

AUDIO = 'audio'

Type of *telegram.InputMediaAudio*.

Type
str

DOCUMENT = 'document'

Type of *telegram.InputMediaDocument*.

Type
str

LIVE_PHOTO = 'live_photo'

Type of *telegram.InputMediaLivePhoto*.

Type
str

LOCATION = 'location'

Type of *telegram.InputMediaLocation*.

Type
str

PHOTO = 'photo'

Type of *telegram.InputMediaPhoto*.

Type
str

STICKER = 'sticker'

Type of *telegram.InputMediaSticker*.

Type
str

VENUE = 'venue'

Type of *telegram.InputMediaVenue*.

Type
str

VIDEO = 'video'

Type of *telegram.InputMediaVideo*.

Type
str

class telegram.constants.**BotCommandLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.BotCommand* and *telegram.Bot.set_my_commands()*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

MAX_COMMAND = 32

Maximum value allowed for *command* parameter of *telegram.BotCommand*.

Type
int

MAX_COMMAND_NUMBER = 100

Maximum number of bot commands passed in a list to the *commands* parameter of *telegram.Bot.set_my_commands()*.

Type
int

MAX_DESCRIPTION = 256

Maximum value allowed for *description* parameter of *telegram.BotCommand*.

Type
int

MIN_COMMAND = 1

Minimum value allowed for *command* parameter of *telegram.BotCommand*.

Type
int

MIN_DESCRIPTION = 1

Minimum value allowed for *description* parameter of *telegram.BotCommand*.

Type
int

class telegram.constants.**BotCommandScopeType**(**values*)

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.

Added 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.**BotDescriptionLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for the methods *telegram.Bot.set_my_description()* and *telegram.Bot.set_my_short_description()*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.2.

MAX_DESCRIPTION_LENGTH = 512

Maximum length for the parameter *description* of *telegram.Bot.set_my_description()*

Type
int

MAX_SHORT_DESCRIPTION_LENGTH = 120

Maximum length for the parameter *short_description* of *telegram.Bot.set_my_short_description()*

Type
int

class telegram.constants.**BotNameLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for the methods *telegram.Bot.set_my_name()*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.3.

MAX_NAME_LENGTH = 64

Maximum length for the parameter *name* of *telegram.Bot.set_my_name()*

Type
int

class telegram.constants.**BulkRequestLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Bot.delete_messages()*, *telegram.Bot.forward_messages()* and *telegram.Bot.copy_messages()*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.8.

MAX_LIMIT = 100

Maximum number of messages required for bulk actions.

Type
int

MIN_LIMIT = 1

Minimum number of messages required for bulk actions.

Type
int

class telegram.constants.**BusinessLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations related to handling business accounts. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 22.1.

CHAT_ACTIVITY_TIMEOUT = 86400

Time in seconds in which the chat must have been active for. Relevant for *chat_id* of *read_business_message()* and *new_owner_chat_id* of *transfer_gift()*.

Type
int

MAX_BIO_LENGTH = 140

Maximum length of the bio of a business account. Relevant for *bio* of *telegram.Bot.set_business_account_bio()*.

Type
int

MAX_GIFT_RESULTS = 100

Maximum number of gifts to be returned. Relevant for

- *limit* of *telegram.Bot.get_business_account_gifts()*.
- *limit* of *telegram.Bot.get_chat_gifts()*.
- *limit* of *telegram.Bot.get_user_gifts()*.

Type
int

MAX_NAME_LENGTH = 64

Maximum length of the name of a business account. Relevant for the parameters of *telegram.Bot.set_business_account_name()*.

Type
int

MAX_STAR_COUNT = 10000

Maximum number of Telegram Stars to be transfered. Relevant for *star_count* of *telegram.Bot.transfer_business_account_stars()*.

Type
int

MAX_USERNAME_LENGTH = 32

Maximum length of the username of a business account. Relevant for *username* of *telegram.Bot.set_business_account_username()*.

Type
int

MIN_GIFT_RESULTS = 1

Minimum number of gifts to be returned. Relevant for

- *limit* of *telegram.Bot.get_business_account_gifts()*.

- *limit* of *telegram.Bot.get_chat_gifts()*.
- *limit* of *telegram.Bot.get_user_gifts()*.

Type
int

MIN_NAME_LENGTH = 1

Minimum length of the name of a business account. Relevant only for *first_name* of *telegram.Bot.set_business_account_name()*.

Type
int

MIN_STAR_COUNT = 1

Minimum number of Telegram Stars to be transfered. Relevant for *star_count* of *telegram.Bot.transfer_business_account_stars()*.

Type
int

class telegram.constants.**CallbackQueryLimit**(**values*)

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.

Added in version 20.0.

ANSWER_CALLBACK_QUERY_TEXT_LENGTH = 200

Maximum number of characters in a str passed as the *text* parameter of *telegram.Bot.answer_callback_query()*.

Type
int

class telegram.constants.**ChatAction**(**values*)

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.

Added 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.**ChatBoostSources**(**values*)

Bases: str, enum.Enum

This enum contains the available sources for a *Telegram chat boost*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 20.8.

GIFT_CODE = 'gift_code'

The source of the chat boost was a Telegram Premium gift code.

Type
str

GIVEAWAY = 'giveaway'

The source of the chat boost was a Telegram Premium giveaway.

Type
str

PREMIUM = 'premium'

The source of the chat boost was a Telegram Premium subscription/gift.

Type
str

class telegram.constants.**ChatID**(*values)

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.

Added in version 20.0.

ANONYMOUS_ADMIN = 1087968824

User ID in groups for messages sent by anonymous admins. Telegram chat: @GroupAnonymousBot.

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, or when a channel votes on a poll. Telegram chat: @Channel_Bot.

Note

- *telegram.Message.from_user* will contain this ID for backwards compatibility only. It's recommended to use *telegram.Message.sender_chat* instead.
- *telegram.PollAnswer.user* will contain this ID for backwards compatibility only. It's recommended to use *telegram.PollAnswer.voter_chat* instead.

Type
int

SERVICE_CHAT = 777000

Telegram service chat, that also acts as sender of channel posts forwarded to discussion groups. Telegram chat: Telegram.

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**(*values)

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.

Added in version 20.0.

MAX_MEMBER_LIMIT = 99999

Maximum value allowed for the *member_limit* parameter of *telegram.Bot.create_chat_invite_link()* and *member_limit* of *telegram.Bot.edit_chat_invite_link()*.

Type
`int`

MIN_MEMBER_LIMIT = 1

Minimum value allowed for the *member_limit* parameter of *telegram.Bot.create_chat_invite_link()* and *member_limit* of *telegram.Bot.edit_chat_invite_link()*.

Type
`int`

NAME_LENGTH = 32

Maximum number of characters in a `str` passed as the *name* parameter of *telegram.Bot.create_chat_invite_link()* and *name* of *telegram.Bot.edit_chat_invite_link()*.

Type
`int`

class telegram.constants.**ChatLimit**(**values*)

Bases: `enum.IntEnum`

This enum contains limitations for *telegram.Bot.set_chat_administrator_custom_title()*, *telegram.Bot.set_chat_description()*, and *telegram.Bot.set_chat_title()*. The enum members of this enumeration are instances of `int` and can be treated as such.

Added in version 20.0.

CHAT_ADMINISTRATOR_CUSTOM_TITLE_LENGTH = 16

Maximum length of a `str` passed as the *custom_title* parameter of *telegram.Bot.set_chat_administrator_custom_title()*.

Type
`int`

CHAT_DESCRIPTION_LENGTH = 255

Maximum number of characters in a `str` passed as the *description* parameter of *telegram.Bot.set_chat_description()*.

Type
`int`

MAX_CHAT_TITLE_LENGTH = 128

Maximum length of a `str` passed as the *title* parameter of *telegram.Bot.set_chat_title()*.

Type
`int`

MIN_CHAT_TITLE_LENGTH = 1

Minimum length of a `str` passed as the *title* parameter of *telegram.Bot.set_chat_title()*.

Type
`int`

class telegram.constants.**ChatMemberStatus**(**values*)

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.

Added 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.**ChatPhotoSize**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.ChatPhoto*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

BIG = 640

Width and height of a big chat photo, ID of which is passed in *big_file_id* and *big_file_unique_id* parameters of *telegram.ChatPhoto*.

Type
int

SMALL = 160

Width and height of a small chat photo, ID of which is passed in *small_file_id* and *small_file_unique_id* parameters of *telegram.ChatPhoto*.

Type
int

class telegram.constants.**ChatSubscriptionLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Bot.create_chat_subscription_invite_link.subscription_period* and *telegram.Bot.create_chat_subscription_invite_link.subscription_price*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 21.5.

MAX_PRICE = 10000

Amount of stars a user pays, maximum amount the subscription can be set to.

Changed in version 22.1: Bot API 9.0 changed the value to 10000.

Type
int

MIN_PRICE = 1

Amount of stars a user pays, minimum amount the subscription can be set to.

Type
int

SUBSCRIPTION_PERIOD = 2592000

The number of seconds the subscription will be active.

Type
int

class telegram.constants.**ChatType**(**values*)

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.

Added 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.**ContactLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.InlineQueryResultContact*, *telegram.InputContactMessageContent*, and *telegram.Bot.send_contact()*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

VCARD = 2048

Maximum value allowed for:

- *vcard* parameter of *send_contact()*
- *vcard* parameter of *InlineQueryResultContact*
- *vcard* parameter of *InputContactMessageContent*

Type
int

class telegram.constants.**CustomEmojiStickerLimit**(**values*)

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.

Added 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**(**values*)

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.

Added 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.**DiceLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Dice*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

MAX_VALUE_BASKETBALL = 5

Maximum value allowed for *value* parameter of *telegram.Dice* if *emoji* is ''.

Type
int

MAX_VALUE_BOWLING = 6

Maximum value allowed for *value* parameter of *telegram.Dice* if *emoji* is ''.

Type
int

MAX_VALUE_DARTS = 6

Maximum value allowed for *value* parameter of *telegram.Dice* if *emoji* is ''.

Type
int

MAX_VALUE_DICE = 6

Maximum value allowed for *value* parameter of *telegram.Dice* if *emoji* is ''.

Type
int

MAX_VALUE_FOOTBALL = 5

Maximum value allowed for *value* parameter of *telegram.Dice* if *emoji* is ''.

Type
int

MAX_VALUE_SLOT_MACHINE = 64

Maximum value allowed for *value* parameter of *telegram.Dice* if *emoji* is ''.

Type
int

MIN_VALUE = 1

Minimum value allowed for *value* parameter of *telegram.Dice* (any emoji).

Type
int

class telegram.constants.**FileSizeLimit**(**values*)

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.

Added in version 20.0.

FILESIZE_DOWNLOAD = 20000000

Bots can download files of up to 20MB in size.

Type
int

FILESIZE_DOWNLOAD_LOCAL_MODE = 9223372036854775807

Bots can download files without a size limit when using a local bot API server.

Type
int

FILESIZE_UPLOAD = 50000000

Bots can upload non-photo files of up to 50MB in size.

Type
int

FILESIZE_UPLOAD_LOCAL_MODE = 2000000000

Bots can upload non-photo files of up to 2000MB in size when using a local bot API server.

Type
int

PHOTOSIZE_UPLOAD = 10000000

Bots can upload photo files of up to 10MB in size.

Type
int

VOICE_NOTE_FILE_SIZE = 1000000

File size limit for the *send_voice()* method of *telegram.Bot*. Bots can send *audio/ogg* files of up to 1MB in size as a voice note. Larger voice notes (up to 20MB) will be sent as files.

Type
int

class telegram.constants.**FloodLimit**(**values*)

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.

Added 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

PAID_MESSAGES_PER_SECOND = 1000

The number of messages that can be sent per second when paying with the bot's Telegram Star balance. See e.g. parameter *allow_paid_broadcast* of *send_message()*.

Added in version 21.7.

Type
int

class telegram.constants.**ForumIconColor**(**values*)

Bases: enum.IntEnum

This enum contains the available colors for use in *telegram.Bot.create_forum_topic.icon_color*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

BLUE = 7322096

An icon with a color which corresponds to blue (0x6FB9F0).

Type
int

GREEN = 9367192

An icon with a color which corresponds to green (0x8EEE98).

Type
int

PINK = 16749490

An icon with a color which corresponds to pink (0xFF93B2).

Type
int

PURPLE = 13338331

An icon with a color which corresponds to purple (0xCB86DB).

Type
int

RED = 16478047

An icon with a color which corresponds to red (0xFB6F5F).

Type
int

YELLOW = 16766590

An icon with a color which corresponds to yellow (0xFFD67E).

Type
int

class telegram.constants.**ForumTopicLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Bot.create_forum_topic.name* and *telegram.Bot.edit_forum_topic.name*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

MAX_NAME_LENGTH = 128

Maximum length of a str passed as:

- *name* parameter of *telegram.Bot.create_forum_topic()*
- *name* parameter of *telegram.Bot.edit_forum_topic()*
- *name* parameter of *telegram.Bot.edit_general_forum_topic()*

Type
int

MIN_NAME_LENGTH = 1

Minimum length of a str passed as:

- *name* parameter of *telegram.Bot.create_forum_topic()*
- *name* parameter of *telegram.Bot.edit_forum_topic()*
- *name* parameter of *telegram.Bot.edit_general_forum_topic()*

Type
int

class telegram.constants.**GiftLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *send_gift()*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 21.8.

MAX_TEXT_LENGTH = 128

Maximum number of characters in a str passed as the *text* parameter of *send_gift()*.

Changed in version 21.11: Updated Value to 128 based on Bot API 8.3

Type
int

class telegram.constants.**GiveawayLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Giveaway* and related classes. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.8.

MAX_WINNERS = 100

Maximum number of winners allowed for *telegram.GiveawayWinners.winners*.

Type
int

class telegram.constants.**InlineKeyboardButtonLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.InlineKeyboardButton*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

MAX_CALLBACK_DATA = 64

Maximum length allowed for *callback_data* parameter of *telegram.InlineKeyboardButton*

Type
int

MAX_COPY_TEXT = 256

Maximum length allowed for *text* parameter of *telegram.CopyTextButton*

Type
int

MIN_CALLBACK_DATA = 1

Minimum length allowed for *callback_data* parameter of *telegram.InlineKeyboardButton*

Type
int

MIN_COPY_TEXT = 1

Minimum length allowed for *text* parameter of *telegram.CopyTextButton*

Type
int

class telegram.constants.**InlineKeyboardMarkupLimit**(**values*)

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.

Added 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**(**values*)

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.

Added in version 20.0.

Changed in version 22.0: Removed deprecated attributes InlineQueryLimit.MIN_SWITCH_PM_TEXT_LENGTH and InlineQueryLimit.MAX_SWITCH_PM_TEXT_LENGTH. Please instead use *InlineQueryResultsButtonLimit.MIN_START_PARAMETER_LENGTH* and *InlineQueryResultsButtonLimit.MAX_START_PARAMETER_LENGTH*.

MAX_OFFSET_LENGTH = 64

Maximum number of bytes in a str passed as the *next_offset* parameter of *telegram.Bot.answer_inline_query()*.

Type
int

MAX_QUERY_LENGTH = 256

Maximum number of characters in a str passed as the *query* parameter of *telegram.InlineQuery*.

Type
int

RESULTS = 50

Maximum number of results that can be passed to *telegram.Bot.answer_inline_query()*.

Type
int

class telegram.constants.**InlineQueryResultLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.InlineQueryResult* and its subclasses. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

MAX_ID_LENGTH = 64

Maximum number of bytes in a str passed as the *id* parameter of *telegram.InlineQueryResult* and its subclasses

Type
int

MIN_ID_LENGTH = 1

Minimum number of bytes in a str passed as the *id* parameter of *telegram.InlineQueryResult* and its subclasses

Type
int

class telegram.constants.**InlineQueryResultType**(**values*)

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.

Added 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.**InlineQueryResultsButtonLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.InlineQueryResultsButton*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.3.

MAX_START_PARAMETER_LENGTH = 64

Maximum number of characters in a str passed as the *start_parameter* parameter of *telegram.InlineQueryResultsButton()*.

Type
int

MIN_START_PARAMETER_LENGTH = 1

Minimum number of characters in a str passed as the *start_parameter* parameter of *telegram.InlineQueryResultsButton()*.

Type
int

class telegram.constants.**InputChecklistLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.InputChecklist*/ *telegram.InputChecklistTask*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 22.3.

MAX_TASK_NUMBER = 30

Maximum number of tasks passed as *tasks* parameter of *telegram.InputChecklistTask*

Type
int

MAX_TEXT_LENGTH = 100

Maximum number of characters in a str passed as *text* parameter of *telegram.InputChecklistTask*

Type
int

MAX_TITLE_LENGTH = 255

Maximum number of characters in a str passed as *title* parameter of *telegram.InputChecklist*

Type
int

MIN_TASK_NUMBER = 1

Minimum number of tasks passed as *tasks* parameter of *telegram.InputChecklist*

Type
int

MIN_TEXT_LENGTH = 1

Minimum number of characters in a str passed as *text* parameter of *telegram.InputChecklistTask*

Type
int

MIN_TITLE_LENGTH = 1

Minimum number of characters in a str passed as *title* parameter of *telegram.InputChecklist*

Type
int

class telegram.constants.**InputMediaType**(**values*)

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.

Deprecated since version 22.8: Use *telegram.constants.BaseInputMediaType* instead.

Added 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

LIVE_PHOTO = 'live_photo'

Type of *telegram.InputMediaLivePhoto*.

Added in version 22.8.

Type
str

PHOTO = 'photo'

Type of *telegram.InputMediaPhoto*.

Type
str

VIDEO = 'video'

Type of *telegram.InputMediaVideo*.

Type
str

class telegram.constants.**InputPaidMediaType**(**values*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.InputPaidMedia*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 21.4.

LIVE_PHOTO = 'live_photo'

Type of *telegram.InputPaidMediaLivePhoto*.

Added in version 22.8.

Type
str

PHOTO = 'photo'

Type of *telegram.InputPaidMediaPhoto*.

Type
str

VIDEO = 'video'

Type of *telegram.InputPaidMediaVideo*.

Type
str

class telegram.constants.**InputProfilePhotoType**(**values*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.InputProfilePhoto*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 22.1.

ANIMATED = 'animated'

Type of *telegram.InputProfilePhotoAnimated*.

Type
str

STATIC = 'static'

Type of *telegram.InputProfilePhotoStatic*.

Type
str

class telegram.constants.**InputStoryContentLimit**(**values*)

Bases: str, enum.Enum

This enum contains limitations for *telegram.InputStoryContentPhoto*/ *telegram.InputStoryContentVideo*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 22.1.

MAX_VIDEO_DURATION = '60'

Maximum duration of the video to be passed to *duration* parameter of *telegram.InputStoryContentVideo*.

Type
int

PHOTOSIZE_UPLOAD = '10000000'

Maximum file size of the photo to be passed to *photo* parameter of *telegram.InputStoryContentPhoto* in Bytes.

Type
int

PHOTO_HEIGHT = '1920'

Vertical resolution of the video to be passed to *photo* parameter of *telegram.InputStoryContentPhoto*.

Type
int

PHOTO_WIDTH = '1080'

Horizontal resolution of the photo to be passed to *photo* parameter of *telegram.InputStoryContentPhoto*.

Type
int

VIDEOSIZE_UPLOAD = '30000000'

Maximum file size of the video to be passed to *video* parameter of *telegram.InputStoryContentVideo* in Bytes.

Type
int

VIDEO_HEIGHT = '1080'

Vertical resolution of the video to be passed to *video* parameter of *telegram.InputStoryContentVideo*.

Type
int

VIDEO_WIDTH = '720'

Horizontal resolution of the video to be passed to *video* parameter of *telegram.InputStoryContentVideo*.

Type
int

class telegram.constants.**InputStoryContentType**(**values*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.InputStoryContent*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 22.1.

PHOTO = 'photo'

Type of *telegram.InputStoryContentPhoto*.

Type
str

VIDEO = 'video'

Type of *telegram.InputStoryContentVideo*.

Type
str

class telegram.constants.**InvoiceLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.InputInvoiceMessageContent*, *telegram.Bot.send_invoice()*, and *telegram.Bot.create_invoice_link()*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

MAX_DESCRIPTION_LENGTH = 255

Maximum number of characters in a str passed as:

- *description* parameter of *telegram.InputInvoiceMessageContent*
- *description* parameter of *telegram.Bot.send_invoice()*.
- *description* parameter of *telegram.Bot.create_invoice_link()*.

Type
int

MAX_PAYLOAD_LENGTH = 128

Maximum amount of bytes in a str passed as:

- *payload* parameter of *telegram.InputInvoiceMessageContent*
- *payload* parameter of *telegram.Bot.send_invoice()*.
- *payload* parameter of *telegram.Bot.create_invoice_link()*.
- *payload* parameter of *telegram.Bot.send_paid_media()*.

Type
int

MAX_STAR_COUNT = 25000

Maximum amount of starts that must be paid to buy access to a paid media passed as *star_count* parameter of *telegram.Bot.send_paid_media()*.

Added in version 21.6.

Changed in version 22.1: Bot API 9.0 changed the value to 10000.

Changed in version 22.6: Bot API 9.3 changed the value to 25000.

Type
int

MAX_TIP_AMOUNTS = 4

Maximum length of a Sequence passed as:

- *suggested_tip_amounts* parameter of *telegram.Bot.send_invoice()*.
- *suggested_tip_amounts* parameter of *telegram.Bot.create_invoice_link()*.

Type
int

MAX_TITLE_LENGTH = 32

Maximum number of characters in a str passed as:

- *title* parameter of *telegram.InputInvoiceMessageContent*
- *title* parameter of *telegram.Bot.send_invoice()*.
- *title* parameter of *telegram.Bot.create_invoice_link()*.

Type
int

MIN_DESCRIPTION_LENGTH = 1

Minimum number of characters in a str passed as:

- *description* parameter of *telegram.InputInvoiceMessageContent*
- *description* parameter of *telegram.Bot.send_invoice()*.
- *description* parameter of *telegram.Bot.create_invoice_link()*.

Type
int

MIN_PAYLOAD_LENGTH = 1

Minimum amount of bytes in a str passed as:

- *payload* parameter of *telegram.InputInvoiceMessageContent*
- *payload* parameter of *telegram.Bot.send_invoice()*.
- *payload* parameter of *telegram.Bot.create_invoice_link()*.

Type
int

MIN_STAR_COUNT = 1

Minimum amount of starts that must be paid to buy access to a paid media passed as *star_count* parameter of *telegram.Bot.send_paid_media()*.

Added in version 21.6.

Type
int

MIN_TITLE_LENGTH = 1

Minimum number of characters in a str passed as:

- *title* parameter of *telegram.InputInvoiceMessageContent*
- *title* parameter of *telegram.Bot.send_invoice()*.
- *title* parameter of *telegram.Bot.create_invoice_link()*.

Type
int

SUBSCRIPTION_MAX_PRICE = 10000

The maximum price of a subscription created wtih *telegram.Bot.create_invoice_link()*.

Added in version 21.9.

Changed in version 22.1: Bot API 9.0 changed the value to 10000.

Type
int

SUBSCRIPTION_PERIOD = 2592000

The period of time for which the subscription is active before the next payment, passed as *subscription_period* parameter of *telegram.Bot.create_invoice_link()*.

Added in version 21.8.

Type
int

class telegram.constants.**KeyboardButtonRequestUsersLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.KeyboardButtonRequestUsers*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.8.

MAX_QUANTITY = 10

Maximum value allowed for *max_quantity* parameter of *telegram.KeyboardButtonRequestUsers*.

Type
int

MIN_QUANTITY = 1

Minimum value allowed for *max_quantity* parameter of *telegram.KeyboardButtonRequestUsers*.

Type
int

class telegram.constants.**KeyboardButtonStyle**(**values*)

Bases: str, enum.Enum

This enum contains the available button styles for *telegram.InlineKeyboardButton* and *telegram.KeyboardButton*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 22.7.

BLUE = 'primary'

Alias for *PRIMARY*. Blue button style for the *style* and *style* parameters.

Type
str

DANGER = 'danger'

Danger/destructive button style (usually red) for the *style* and *style* parameters.

Type
str

GREEN = 'success'

Alias for *SUCCESS*. Green button style for the *style* and *style* parameters.

Type
str

PRIMARY = 'primary'

Primary button style (usually blue) for the *style* and *style* parameters.

Type
str

RED = 'danger'

Alias for *DANGER*. Red button style for the *style* and *style* parameters.

Type
str

SUCCESS = 'success'

Success button style (usually green) for the *style* and *style* parameters.

Type
str

class telegram.constants.**LocationLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Location*/*telegram.ChatLocation*/ *telegram.Bot.edit_message_live_location()*/*telegram.Bot.send_location()*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

HORIZONTAL_ACCURACY = 1500

Maximum value allowed for:

- *horizontal_accuracy* parameter of *telegram.Location*
- *horizontal_accuracy* parameter of *telegram.InlineQueryResultLocation*
- *horizontal_accuracy* parameter of *telegram.InputLocationMessageContent*
- *horizontal_accuracy* parameter of *telegram.Bot.edit_message_live_location()*
- *horizontal_accuracy* parameter of *telegram.Bot.send_location()*
- *horizontal_accuracy* parameter of *telegram.InputMediaLocation*

Type
int

LIVE_PERIOD_FOREVER = 2147483647

Value for live locations that can be edited indefinitely. Passed in:

- *live_period* parameter of *telegram.InlineQueryResultLocation*
- *live_period* parameter of *telegram.InputLocationMessageContent*
- *live_period* parameter of *telegram.Bot.edit_message_live_location()*
- *live_period* parameter of *telegram.Bot.send_location()*

Added in version 21.2.

Type
int

MAX_CHAT_LOCATION_ADDRESS = 64

Minimum value allowed for *address* parameter of *telegram.ChatLocation*

Type
int

MAX_HEADING = 360

Maximum value allowed for:

- *heading* parameter of *telegram.Location*
- *heading* parameter of *telegram.InlineQueryResultLocation*
- *heading* parameter of *telegram.InputLocationMessageContent*
- *heading* parameter of *telegram.Bot.edit_message_live_location()*
- *heading* parameter of *telegram.Bot.send_location()*

Type
int

MAX_LIVE_PERIOD = 86400

Maximum value allowed for:

- *live_period* parameter of *telegram.InlineQueryResultLocation*
- *live_period* parameter of *telegram.InputLocationMessageContent*
- *live_period* parameter of *telegram.Bot.edit_message_live_location()*
- *live_period* parameter of *telegram.Bot.send_location()*

Type
int

MAX_PROXIMITY_ALERT_RADIUS = 100000

Maximum value allowed for:

- *proximity_alert_radius* parameter of *telegram.InlineQueryResultLocation*
- *proximity_alert_radius* parameter of *telegram.InputLocationMessageContent*
- *proximity_alert_radius* parameter of *telegram.Bot.edit_message_live_location()*
- *proximity_alert_radius* parameter of *telegram.Bot.send_location()*

Type
int

MIN_CHAT_LOCATION_ADDRESS = 1

Minimum value allowed for *address* parameter of *telegram.ChatLocation*

Type
int

MIN_HEADING = 1

Minimum value allowed for:

- *heading* parameter of *telegram.Location*
- *heading* parameter of *telegram.InlineQueryResultLocation*
- *heading* parameter of *telegram.InputLocationMessageContent*
- *heading* parameter of *telegram.Bot.edit_message_live_location()*
- *heading* parameter of *telegram.Bot.send_location()*

Type
int

MIN_LIVE_PERIOD = 60

Minimum value allowed for:

- *live_period* parameter of *telegram.InlineQueryResultLocation*
- *live_period* parameter of *telegram.InputLocationMessageContent*
- *live_period* parameter of *telegram.Bot.edit_message_live_location()*
- *live_period* parameter of *telegram.Bot.send_location()*

Type
int

MIN_PROXIMITY_ALERT_RADIUS = 1

Minimum value allowed for:

- *proximity_alert_radius* parameter of *telegram.InlineQueryResultLocation*
- *proximity_alert_radius* parameter of *telegram.InputLocationMessageContent*
- *proximity_alert_radius* parameter of *telegram.Bot.edit_message_live_location()*
- *proximity_alert_radius* parameter of *telegram.Bot.send_location()*

Type
int

class telegram.constants.**ManagedBotAccessLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *set_managed_bot_access_settings()*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 22.8.

MAX_ALLOWED_USERS = 10

Maximum number of users that can be allowed to access a managed bot in the *added_user_ids* parameter of *set_managed_bot_access_settings()*.

Type
int

class telegram.constants.**MaskPosition**(**values*)

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.

Added 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.**MediaGroupLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Bot.send_media_group()*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

MAX_MEDIA_LENGTH = 10

Maximum length of a `list` passed as the *media* parameter of *telegram.Bot.send_media_group()*.

Type
int

MIN_MEDIA_LENGTH = 2

Minimum length of a `list` passed as the *media* parameter of *telegram.Bot.send_media_group()*.

Type
int

class telegram.constants.**MenuButtonType**(**values*)

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.

Added 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**(**values*)

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.

Added 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

LIVE_PHOTO = 'live_photo'

Messages with *telegram.Message.live_photo*.

Added in version 22.8.

Type
str

LOCATION = 'location'

Messages with *telegram.Message.location*.

Type
str

PAID_MEDIA = 'paid_media'

Messages with *telegram.Message.paid_media*.

Added in version 21.4.

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

STORY = 'story'

Messages with *telegram.Message.story*.

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.**MessageEntityDateTimeFormats**(**values*)

Bases: str, enum.Enum

This enum contains all possible formats for *telegram.MessageEntity.date_time_format*. Please read date-time entity formatting for more details. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 22.7.

LOCALIZED_WEEKDAY = 'w'

Displays the day of the week in the user's localized language.

Type
str

LOCALIZED_WEEKDAY_LONG_DATE = 'wD'

Displays the day of the week in the user's localized language and the date in long form.

Type
str

LOCALIZED_WEEKDAY_LONG_DATE_LONG_TIME = 'wDT'

Displays the day of the week in the user's localized language, the date in long form and the time in long form.

Type
str

LOCALIZED_WEEKDAY_LONG_DATE_SHORT_TIME = 'wDt'

Displays the day of the week in the user's localized language, the date in long form and the time in short form.

Type
str

LOCALIZED_WEEKDAY_LONG_TIME = 'wT'

Displays the day of the week in the user's localized language and the time in long form.

Type
str

LOCALIZED_WEEKDAY_SHORT_DATE = 'wd'

Displays the day of the week in the user's localized language and the date in short form.

Type
str

LOCALIZED_WEEKDAY_SHORT_DATE_LONG_TIME = 'wdT'

Displays the day of the week in the user's localized language, the date in short form and the time in long form.

Type
str

LOCALIZED_WEEKDAY_SHORT_DATE_SHORT_TIME = 'wdt'

Displays the day of the week in the user's localized language, the date in short form and the time in short form.

Type
str

LOCALIZED_WEEKDAY_SHORT_TIME = 'wt'

Displays the day of the week in the user's localized language and the time in short form.

Type
str

LONG_DATE = 'D'

Displays the date in long form (e.g., `March 17, 2022`).

Type
str

LONG_DATE_LONG_TIME = 'DT'

Displays the date in long form and the time in long form.

Type
str

LONG_DATE_SHORT_TIME = 'Dt'

Displays the date in long form and the time in short form.

Type
str

LONG_TIME = 'T'

Displays the time in long form (e.g., `22:45:00`).

Type
str

RELATIVE = 'r'

Displays the time relative to the current time.

Type
str

SHORT_DATE = 'd'

Displays the date in short form (e.g., `17.03.22`).

Type
str

SHORT_DATE_LONG_TIME = 'dT'

Displays the date in short form and the time in long form.

Type
str

SHORT_DATE_SHORT_TIME = 'dt'

Displays the date in short form and the time in short form.

Type
str

SHORT_TIME = 't'

Displays the time in short form (e.g., `22:45`).

Type
str

class telegram.constants.**MessageEntityType**(**values*)

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.

Added in version 20.0.

BLOCKQUOTE = 'blockquote'

Message entities representing a block quotation.

Added in version 20.8.

Type
str

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.

Added in version 20.0.

Type
str

DATE_TIME = 'date_time'

Message entities representing formatted date and time.

Added in version 22.7.

Type
str

EMAIL = 'email'

Message entities representing a email.

Type
str

EXPANDABLE_BLOCKQUOTE = 'expandable_blockquote'

Message entities representing collapsed-by-default block quotation.

Added in version 21.3.

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**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Message*/ *telegram.InputTextMessageContent*/ *telegram.Bot.send_message()* & friends. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

CAPTION_LENGTH = 1024

Maximum number of characters in a str passed as:

- *caption* parameter of *telegram.Message*
- *caption* parameter of *telegram.InputMedia* and its subclasses
- caption parameter of subclasses of *telegram.InlineQueryResult*
- caption parameter of *telegram.Bot.send_photo()*, *telegram.Bot.send_audio()*, *telegram.Bot.send_document()*, *telegram.Bot.send_video()*, *telegram.Bot.send_animation()*, *telegram.Bot.send_voice()*, *telegram.Bot.edit_message_caption()*, *telegram.Bot.copy_message()*

Type
int

DEEP_LINK_LENGTH = 64

Maximum number of characters for a deep link.

Type
int

MAX_TEXT_LENGTH = 4096

Maximum number of characters in a str passed as:

- *text* parameter of *telegram.Game*
- *text* parameter of *telegram.Message*
- *message_text* parameter of *telegram.InputTextMessageContent*
- *text* parameter of *telegram.Bot.send_message()*
- *text* parameter of *telegram.Bot.edit_message_text()*
- *text* parameter of *telegram.Bot.send_message_draft()*

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

MIN_TEXT_LENGTH = 1

Minimum number of characters in a str passed as:

- *message_text* parameter of *telegram.InputTextMessageContent*.
- *text* parameter of *telegram.Bot.edit_message_text()*.
- *text* parameter of *telegram.Bot.send_message_draft()*.

Type
int

class telegram.constants.**MessageOriginType**(**values*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.MessageOrigin*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 20.8.

CHANNEL = 'channel'

A *telegram.MessageOrigin* who is sent by a channel.

Type
str

CHAT = 'chat'

A *telegram.MessageOrigin* who is sent by a chat.

Type
str

HIDDEN_USER = 'hidden_user'

A *telegram.MessageOrigin* who is sent by a hidden user.

Type
str

USER = 'user'

A *telegram.MessageOrigin* who is sent by an user.

Type
str

class telegram.constants.**MessageType**(**values*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.Message*. Here, a "type" means a kind of message that is visually distinct from other kinds of messages in the Telegram app. In particular, auxiliary attributes that can be present for multiple types of messages are not considered in this enumeration.

The enum members of this enumeration are instances of str and can be treated as such.

Added in version 20.0.

ANIMATION = 'animation'

Messages with *telegram.Message.animation*.

Type
str

AUDIO = 'audio'

Messages with *telegram.Message.audio*.

Type
str

BOOST_ADDED = 'boost_added'

Messages with *telegram.Message.boost_added*.

Added in version 21.0.

Type
str

BUSINESS_CONNECTION_ID = 'business_connection_id'

Messages with *telegram.Message.business_connection_id*.

Added in version 21.1.

Type
str

CHANNEL_CHAT_CREATED = 'channel_chat_created'

Messages with *telegram.Message.channel_chat_created*.

Type
str

CHAT_BACKGROUND_SET = 'chat_background_set'

Messages with *telegram.Message.chat_background_set*.

Added in version 21.2.

Type
str

CHAT_OWNER_CHANGED = 'chat_owner_changed'

Messages with *telegram.Message.chat_owner_changed*.

Added in version 22.7.

Type
str

CHAT_OWNER_LEFT = 'chat_owner_left'

Messages with *telegram.Message.chat_owner_left*.

Added in version 22.7.

Type
str

CHAT_SHARED = 'chat_shared'

Messages with *telegram.Message.chat_shared*.

Added in version 20.8.

Type
str

CHECKLIST = 'checklist'

Messages with *telegram.Message.checklist*.

Added in version 22.3.

Type
str

CHECKLIST_TASKS_ADDED = 'checklist_tasks_added'

Messages with *telegram.Message.checklist_tasks_added*.

Added in version 22.3.

Type
str

CHECKLIST_TASKS_DONE = 'checklist_tasks_done'

Messages with *telegram.Message.checklist_tasks_done*.

Added in version 22.3.

Type
str

CONNECTED_WEBSITE = 'connected_website'

Messages with *telegram.Message.connected_website*.

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

DIRECT_MESSAGE_PRICE_CHANGED = 'direct_message_price_changed'

Messages with *telegram.Message.direct_message_price_changed*.

Added in version 22.3.

Type
str

DOCUMENT = 'document'

Messages with *telegram.Message.document*.

Type
str

EFFECT_ID = 'effect_id'

Messages with *telegram.Message.effect_id*.

Added in version 21.3.

Type
str

FORUM_TOPIC_CLOSED = 'forum_topic_closed'

Messages with *telegram.Message.forum_topic_closed*.

Added in version 20.8.

Type
str

FORUM_TOPIC_CREATED = 'forum_topic_created'

Messages with *telegram.Message.forum_topic_created*.

Added in version 20.8.

Type
str

FORUM_TOPIC_EDITED = 'forum_topic_edited'

Messages with *telegram.Message.forum_topic_edited*.

Added in version 20.8.

Type
str

FORUM_TOPIC_REOPENED = 'forum_topic_reopened'

Messages with *telegram.Message.forum_topic_reopened*.

Added in version 20.8.

Type
str

GAME = 'game'

Messages with *telegram.Message.game*.

Type
str

GENERAL_FORUM_TOPIC_HIDDEN = 'general_forum_topic_hidden'

Messages with *telegram.Message.general_forum_topic_hidden*.

Added in version 20.8.

Type
str

GENERAL_FORUM_TOPIC_UNHIDDEN = 'general_forum_topic_unhidden'

Messages with *telegram.Message.general_forum_topic_unhidden*.

Added in version 20.8.

Type
str

GIFT = 'gift'

Messages with *telegram.Message.gift*.

Added in version 22.1.

Type
str

GIFT_UPGRADE_SENT = 'gift_upgrade_sent'

Messages with *telegram.Message.gift_upgrade_sent*.

Added in version 22.6.

Type
str

GIVEAWAY = 'giveaway'

Messages with *telegram.Message.giveaway*.

Added in version 20.8.

Type
str

GIVEAWAY_COMPLETED = 'giveaway_completed'

Messages with *telegram.Message.giveaway_completed*.

Added in version 20.8.

Type
str

GIVEAWAY_CREATED = 'giveaway_created'

Messages with *telegram.Message.giveaway_created*.

Added in version 20.8.

Type
str

GIVEAWAY_WINNERS = 'giveaway_winners'

Messages with *telegram.Message.giveaway_winners*.

Added in version 20.8.

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

LIVE_PHOTO = 'live_photo'

Messages with *telegram.Message.live_photo*.

Added in version 22.8.

Type
str

LOCATION = 'location'

Messages with *telegram.Message.location*.

Type
str

MANAGED_BOT_CREATED = 'managed_bot_created'

Messages with *telegram.Message.managed_bot_created*.

Type
str

MESSAGE_AUTO_DELETE_TIMER_CHANGED = 'message_auto_delete_timer_changed'

Messages with *telegram.Message.message_auto_delete_timer_changed*.

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

PAID_MEDIA = 'paid_media'

Messages with *telegram.Message.paid_media*.

Added in version 21.4.

Type
str

PAID_MESSAGE_PRICE_CHANGED = 'paid_message_price_changed'

Messages with *telegram.Message.paid_message_price_changed*.

Added in version v22.2.

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

POLL_OPTION_ADDED = 'poll_option_added'

Messages with *telegram.Message.poll_option_added*.

Added in version 22.8.

Type
str

POLL_OPTION_DELETED = 'poll_option_deleted'

Messages with *telegram.Message.poll_option_deleted*.

Added in version 22.8.

Type
str

PROXIMITY_ALERT_TRIGGERED = 'proximity_alert_triggered'

Messages with *telegram.Message.proximity_alert_triggered*.

Type
str

REFUNDED_PAYMENT = 'refunded_payment'

Messages with *telegram.Message.refunded_payment*.

Added in version 21.4.

Type
str

REPLY_TO_STORY = 'reply_to_story'

Messages with *telegram.Message.reply_to_story*.

Added in version 21.0.

Type
str

SENDER_BOOST_COUNT = 'sender_boost_count'

Messages with *telegram.Message.sender_boost_count*.

Added in version 21.0.

Type
str

SENDER_BUSINESS_BOT = 'sender_business_bot'

Messages with *telegram.Message.sender_business_bot*.

Added in version 21.1.

Type
str

STICKER = 'sticker'

Messages with *telegram.Message.sticker*.

Type
str

STORY = 'story'

Messages with *telegram.Message.story*.

Type
str

SUCCESSFUL_PAYMENT = 'successful_payment'

Messages with *telegram.Message.successful_payment*.

Type
str

SUGGESTED_POST_APPROVAL_FAILED = 'suggested_post_approval_failed'

Messages with *telegram.Message.suggested_post_approval_failed*.

Added in version 22.4.

Type
str

SUGGESTED_POST_APPROVED = 'suggested_post_approved'

Messages with *telegram.Message.suggested_post_approved*.

Added in version 22.4.

Type
str

SUGGESTED_POST_DECLINED = 'suggested_post_declined'

Messages with *telegram.Message.suggested_post_declined*.

Added in version 22.4.

Type
str

SUGGESTED_POST_INFO = 'suggested_post_info'

Messages with *telegram.Message.suggested_post_info*.

Added in version 22.4.

Type
str

SUGGESTED_POST_PAID = 'suggested_post_paid'

Messages with *telegram.Message.suggested_post_paid*.

Added in version 22.4.

Type
str

SUGGESTED_POST_REFUNDED = 'suggested_post_refunded'

Messages with *telegram.Message.suggested_post_refunded*.

Added in version 22.4.

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

UNIQUE_GIFT = 'unique_gift'

Messages with *telegram.Message.unique_gift*.

Added in version 22.1.

Type
str

USERS_SHARED = 'users_shared'

Messages with *telegram.Message.users_shared*.

Added in version 20.8.

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

WEB_APP_DATA = 'web_app_data'

Messages with *telegram.Message.web_app_data*.

Added in version 20.8.

Type
str

WRITE_ACCESS_ALLOWED = 'write_access_allowed'

Messages with *telegram.Message.write_access_allowed*.

Added in version 20.8.

Type
str

class telegram.constants.**Nanostar**(**values*)

Bases: float, enum.Enum

This enum contains constants for nanostar_amount parameter of *telegram.StarAmount*, *telegram.StarTransaction* and *telegram.AffiliateInfo*. The enum members of this enumeration are instances of float and can be treated as such.

Added in version 22.1.

VALUE = 1e-09

The value of one nanostar as used in *telegram.StarTransaction.nanostar_amount* parameter of *telegram.StarTransaction*, *telegram.StarAmount.nanostar_amount* parameter of *telegram.StarAmount* and *telegram.AffiliateInfo.nanostar_amount* parameter of *telegram.AffiliateInfo*

Type
float

class telegram.constants.**NanostarLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for nanostar_amount parameter of *telegram.AffiliateInfo*, *telegram.StarTransaction* and *telegram.StarAmount*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 22.1.

MAX_AMOUNT = 999999999

Maximum value allowed for *nanostar_amount* parameter of *telegram.StarTransaction*, *nanostar_amount* parameter of *telegram.AffiliateInfo* and *nanostar_amount* parameter of *telegram.StarAmount*.

Type
int

MIN_AMOUNT = -999999999

Minimum value allowed for *nanostar_amount* parameter of *telegram.AffiliateInfo* and *nanostar_amount* parameter of *telegram.StarAmount*.

Type
int

class telegram.constants.**OwnedGiftType**(**values*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.OwnedGift*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 22.1.

REGULAR = 'regular'

a regular owned gift.

Type
str

UNIQUE = 'unique'

a unique owned gift.

Type
str

class telegram.constants.**PaidMediaType**(**values*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.PaidMedia*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 21.4.

LIVE_PHOTO = 'live_photo'

The type of *telegram.PaidMediaLivePhoto*

Added in version 22.8.

Type
str

PHOTO = 'photo'

The type of *telegram.PaidMediaPhoto*.

Type
str

PREVIEW = 'preview'

The type of *telegram.PaidMediaPreview*.

Type
str

VIDEO = 'video'

The type of *telegram.PaidMediaVideo*.

Type
str

class telegram.constants.**ParseMode**(**values*)

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.

Added 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.**PersonalChatMessagesLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Bot.get_user_personal_chat_messages.limit*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 22.8.

MAX_LIMIT = 20

Maximum value allowed for the *limit* parameter of *telegram.Bot.get_user_personal_chat_messages()*.

Type
int

MIN_LIMIT = 1

Minimum value allowed for the *limit* parameter of *telegram.Bot.get_user_personal_chat_messages()*.

Type
int

class telegram.constants.**PollLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Poll*/*telegram.PollOption*/ *telegram.Bot.send_poll()*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

MAX_COUNTRY_CODES = 12

Maximum number of two-letter ISO 3166-1 alpha-2 country codes passed in a list to the *country_codes* parameter of *telegram.Bot.send_poll()*.

Added in version 22.8.

Type
int

MAX_DESCRIPTION_CHARACTERS = 1024

Maximum value allowed for the *description* parameter of *telegram.Bot.send_poll()*.

Added in version 22.8.

Type
int

MAX_EXPLANATION_LENGTH = 200

Maximum number of characters in a str passed as the *explanation* parameter of *telegram.Poll* and the *explanation* parameter of *telegram.Bot.send_poll()*.

Type
int

MAX_EXPLANATION_LINE_FEEDS = 2

Maximum number of line feeds in a str passed as the *explanation* parameter of *telegram.Bot.send_poll()* after entities parsing.

Type
int

MAX_OPEN_PERIOD = 2628000

Maximum value allowed for the *open_period* parameter of *telegram.Bot.send_poll()*. Also used in the *close_date* parameter of *telegram.Bot.send_poll()*.

Changed in version 22.8: Changed from 600 to 2628000 since Bot API 9.6.

Type
int

MAX_OPTION_LENGTH = 100

Maximum length of each str passed in a list to the *options* parameter of *telegram.Bot.send_poll()*.

Type
int

MAX_OPTION_NUMBER = 12

Maximum number of strings passed in a list to the *options* parameter of *telegram.Bot.send_poll()*.

Changed in version 22.3: This value was changed from 10 to 12 in accordance to Bot API 9.1.

Type
int

MAX_QUESTION_LENGTH = 300

Maximum value allowed for the *question* parameter of *telegram.Poll* and the *question* parameter of *telegram.Bot.send_poll()*.

Type
int

MIN_MEMBERSHIP_HOURS = 24

Minimum number of hours a user must have been a member of the chat before they can vote in a members-only poll.

Added in version 22.8.

Type
int

MIN_OPEN_PERIOD = 5

Minimum value allowed for the *open_period* parameter of *telegram.Bot.send_poll()*. Also used in the *close_date* parameter of *telegram.Bot.send_poll()*.

Type
int

MIN_OPTION_LENGTH = 1

Minimum length of each str passed in a list to the *options* parameter of *telegram.Bot.send_poll()*.

Type
int

MIN_OPTION_NUMBER = 1

Minimum number of strings passed in a list to the *options* parameter of *telegram.Bot.send_poll()*.

Changed in version 22.8: Bot API 10.0 decreased this value from 2 to 1.

Type
int

MIN_QUESTION_LENGTH = 1

Minimum value allowed for the *question* parameter of *telegram.Poll* and the *question* parameter of *telegram.Bot.send_poll()*.

Type
int

class telegram.constants.**PollType**(**values*)

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.

Added in version 20.0.

QUIZ = 'quiz'

quiz polls.

Type
str

REGULAR = 'regular'

regular polls.

Type
str

class telegram.constants.**PollingLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Bot.get_updates.limit*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

MAX_LIMIT = 100

Maximum value allowed for the *limit* parameter of *telegram.Bot.get_updates()*.

Type
int

MIN_LIMIT = 1

Minimum value allowed for the *limit* parameter of *telegram.Bot.get_updates()*.

Type
int

class telegram.constants.**PremiumSubscription**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *gift_premium_subscription()*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 22.1.

MAX_TEXT_LENGTH = 128

Maximum number of characters in a str passed as the *text* parameter of *gift_premium_subscription()*.

Type
int

MONTH_COUNT_SIX = 6

Possible value for *month_count* parameter of *gift_premium_subscription()*; number of months the Premium subscription will be active for.

Type
int

MONTH_COUNT_THREE = 3

Possible value for *month_count* parameter of *gift_premium_subscription()*; number of months the Premium subscription will be active for.

Type
int

MONTH_COUNT_TWELVE = 12

Possible value for *month_count* parameter of *gift_premium_subscription()*; number of months the Premium subscription will be active for.

Type
int

STARS_SIX_MONTHS = 1500

Number of Telegram Stars to pay for a Premium subscription of *6* months period. Relevant for *star_count* parameter of *gift_premium_subscription()*.

Type
int

STARS_THREE_MONTHS = 1000

Number of Telegram Stars to pay for a Premium subscription of *3* months period. Relevant for *star_count* parameter of *gift_premium_subscription()*.

Type
int

STARS_TWELVE_MONTHS = 2500

Number of Telegram Stars to pay for a Premium subscription of *12* months period. Relevant for *star_count* parameter of *gift_premium_subscription()*.

Type
int

class telegram.constants.**ProfileAccentColor**(**values*)

Bases: Enum

This enum contains the available accent colors for *telegram.ChatFullInfo.profile_accent_color_id*. The members of this enum are named tuples with the following attributes:

- identifier (int): The identifier of the accent color.
- name (str): Optional. The name of the accent color.
- light_colors (tuple[str]): Optional. The light colors of the accent color as HEX value.
- dark_colors (tuple[str]): Optional. The dark colors of the accent color as HEX value.

Since Telegram gives no exact specification for the accent colors, future accent colors might have a different data type.

Added in version 20.8.

COLOR_000 = (0, None, (12211792,), (10241344,))

Accent color 0. This contains one light color

and one dark color

COLOR_001 = (1, None, (12745790,), (9723436,))

Accent color 1. This contains one light color

and one dark color

COLOR_002 = (2, None, (9792200,), (7426201,))

Accent color 2. This contains one light color

and one dark color

COLOR_003 = (3, None, (4825941,), (3371323,))

Accent color 3. This contains one light color

and one dark color

COLOR_004 = (4, None, (4102061,), (3702407,))

Accent color 4. This contains one light color

and one dark color

COLOR_005 = (5, None, (5935035,), (4682132,))

Accent color 5. This contains one light color

and one dark color

COLOR_006 = (6, None, (12079992,), (9717603,))

Accent color 6. This contains one light color

and one dark color

COLOR_007 = (7, None, (8358805,), (4412001,))

Accent color 7. This contains one light color

and one dark color

COLOR_008 = (8, None, (13194845, 14253143), (10044227, 11294782))

Accent color 8. This contains two light colors

and two dark colors

COLOR_009 = (9, None, (13595204, 13407283), (9393455, 10580530))

Accent color 9. This contains two light colors

and two dark colors

COLOR_010 = (10, None, (9855700, 12150454), (6506129, 9588898))

Accent color 10. This contains two light colors

and two dark colors

COLOR_011 = (11, None, (4036437, 9021008), (2714179, 6262596))

Accent color 11. This contains two light colors

and two dark colors

COLOR_012 = (12, None, (4036026, 5287320), (3173500, 4102270))

Accent color 12. This contains two light colors

and two dark colors

COLOR_013 = (13, None, (5475266, 5089469), (3694988, 4557729))

Accent color 13. This contains two light colors

and two dark colors

COLOR_014 = (14, None, (11554676, 13723245), (8929632, 10900057))

Accent color 14. This contains two light colors

and two dark colors

COLOR_015 = (15, None, (6517890, 8096407), (5464174, 3688020))

Accent color 15. This contains two light colors

and two dark colors

class telegram.constants.**ReactionEmoji**(**values*)

Bases: str, enum.Enum

This enum contains the available emojis of *telegram.ReactionTypeEmoji*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 20.8.

ALIEN_MONSTER = ''

Alien monster

Type
str

BANANA = ''

Banana

Type
str

BOTTLE_WITH_POPPING_CORK = ''

Bottle with popping cork

Type
str

BROKEN_HEART = ''

Broken heart

Type
str

CHRISTMAS_TREE = ''

Christmas tree

Type
str

CLAPPING_HANDS = ''

Clapping Hands

Type
str

CLOWN_FACE = ''

Clown face

Type
str

CRYING_FACE = ''

Crying face

Type
str

DOVE_OF_PEACE = ''

Dove of peace

Type
str

EYES = ''

Eyes

Type
str

FACE_SCREAMING_IN_FEAR = ''

Face screaming in fear

Type
str

FACE_THROWING_A_KISS = ''

Face throwing a kiss

Type
str

FACE_WITH_ONE_EYEBROW_RAISED = ''

Face with one eyebrow raised

Type
str

FACE_WITH_OPEN_MOUTH_VOMITING = ''

Face with open mouth vomiting

Type
str

FACE_WITH_UNEVEN_EYES_AND_WAVY_MOUTH = ''

Face with uneven eyes and wavy mouth

Type
str

FATHER_CHRISTMAS = ''

Father christmas

Type
str

FEARFUL_FACE = ''

Fearful face

Type
str

FIRE = ''

Fire

Type
str

GHOST = ''

Ghost

Type
str

GRINNING_FACE_WITH_ONE_LARGE_AND_ONE_SMALL_EYE = ''

Grinning face with one large and one small eye

Type
str

GRINNING_FACE_WITH_SMILING_EYES = ''

Grinning face with smiling eyes

Type
str

GRINNING_FACE_WITH_STAR_EYES = ''

Grinning face with star eyes

Type
str

HANDSHAKE = ''

Handshake

Type
str

HEART_ON_FIRE = '\u200d'

Heart on fire

Type
str

HEART_WITH_ARROW = ''

Heart with arrow

Type
str

HEAR_NO_EVIL_MONKEY = ''

Hear-no-evil monkey

Type
str

HIGH_VOLTAGE_SIGN = ''

High voltage sign

Type
str

HOT_DOG = ''

Hot dog

Type
str

HUGGING_FACE = ''

Hugging face

Type
str

HUNDRED_POINTS_SYMBOL = ''

Hundred points symbol

Type
str

JACK_O_LANTERN = ''

Jack-o-lantern

Type
str

KISS_MARK = ''

Kiss mark

Type
str

LOUDLY_CRYING_FACE = ''

Loudly crying face

Type
str

MAN_SHRUGGING = '\u200d'

Man Shrugging

Type
str

MAN_TECHNOLOGIST = '\u200d'

Man Technologist

Type
str

MOYAI = ''

Moyai

Type
str

NAIL_POLISH = ''

Nail polish

Type
str

NERD_FACE = ''

Nerd face

Type
str

NEUTRAL_FACE = ''

Neutral face

Type
str

NEW_MOON_WITH_FACE = ''

New moon with face

Type
str

OK_HAND_SIGN = ''

Ok hand sign

Type
str

PARTY_POPPER = ''

Party popper

Type
str

PERSON_WITH_FOLDED_HANDS = ''

Person with folded hands

Type
str

PILE_OF_POO = ''

Pile of poo

Type
str

PILL = ''

Pill

Type
str

POUTING_FACE = ''

Pouting face

Type
str

RED_HEART = ''

Red Heart

Type
str

REVERSED_HAND_WITH_MIDDLE_FINGER_EXTENDED = ''

Reversed hand with middle finger extended

Type
str

ROLLING_ON_THE_FLOOR_LAUGHING = ''

Rolling on the floor laughing

Type
str

SALUTING_FACE = ''

Saluting face

Type
str

SEE_NO_EVIL_MONKEY = ''

See-no-evil monkey

Type
str

SERIOUS_FACE_WITH_SYMBOLS_COVERING_MOUTH = ''

Serious face with symbols covering mouth

Type
str

SHOCKED_FACE_WITH_EXPLODING_HEAD = ''

Shocked face with exploding head

Type
str

SHRUG = ''

Shrug

Type
str

SLEEPING_FACE = ''

Sleeping face

Type
str

SMILING_FACE_WITH_HALO = ''

Smiling face with halo

Type
str

SMILING_FACE_WITH_HEARTS = ''

Smiling Face with Hearts

Type
str

SMILING_FACE_WITH_HEART_SHAPED_EYES = ''

Smiling face with heart-shaped eyes

Type
str

SMILING_FACE_WITH_HORNS = ''

Smiling face with horns

Type
str

SMILING_FACE_WITH_SUNGLASSES = ''

Smiling face with sunglasses

Type
str

SNOWMAN = ''

Snowman

Type
str

SPEAK_NO_EVIL_MONKEY = ''

Speak-no-evil monkey

Type
str

SPOUTING_WHALE = ''

Spouting whale

Type
str

SQUARED_COOL = ''

Squared cool

Type
str

STRAWBERRY = ''

Strawberry

Type
str

THINKING_FACE = ''

Thinking face

Type
str

THUMBS_DOWN = ''

Thumbs Down

Type
str

THUMBS_UP = ''

Thumbs Up

Type
str

TROPHY = ''

Trophy

Type
str

UNICORN_FACE = ''

Unicorn face

Type
str

WOMAN_SHRUGGING = '\u200d'

Woman Shrugging

Type
str

WRITING_HAND = ''

Writing hand

Type
str

YAWNING_FACE = ''

Yawning face

Type
str

class telegram.constants.**ReactionType**(**values*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.ReactionType*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 20.8.

CUSTOM_EMOJI = 'custom_emoji'

A *telegram.ReactionType* with a custom emoji.

Type
str

EMOJI = 'emoji'

A *telegram.ReactionType* with a normal emoji.

Type
str

PAID = 'paid'

A *telegram.ReactionType* with a paid reaction.

Added in version 21.5.

Type
str

class telegram.constants.**ReplyLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.ForceReply* and *telegram.ReplyKeyboardMarkup*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

MAX_INPUT_FIELD_PLACEHOLDER = 64

Maximum value allowed for *input_field_placeholder* parameter of *telegram.ForceReply* and *input_field_placeholder* parameter of *telegram.ReplyKeyboardMarkup*

Type
int

MIN_INPUT_FIELD_PLACEHOLDER = 1

Minimum value allowed for *input_field_placeholder* parameter of *telegram.ForceReply* and *input_field_placeholder* parameter of *telegram.ReplyKeyboardMarkup*

Type
int

class telegram.constants.**RevenueWithdrawalStateType**(**values*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.RevenueWithdrawalState*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 21.4.

FAILED = 'failed'

A withdrawal failed and the transaction was refunded.

Type
str

PENDING = 'pending'

A withdrawal in progress.

Type
str

SUCCEEDED = 'succeeded'

A withdrawal succeeded.

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.**StarTransactionsLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Bot.get_star_transactions* and *telegram.StarTransaction*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 21.4.

Removed in version 22.3: Removed deprecated attributes StarTransactionsLimit.NANOSTAR_MIN_AMOUNT and StarTransactionsLimit.NANOSTAR_MAX_AMOUNT. Please instead use *telegram.constants.NanostarLimit.MIN_AMOUNT* and *telegram.constants.NanostarLimit.MAX_AMOUNT*.

MAX_LIMIT = 100

Maximum value allowed for the *limit* parameter of *telegram.Bot.get_star_transactions()*.

Type
int

MIN_LIMIT = 1

Minimum value allowed for the *limit* parameter of *telegram.Bot.get_star_transactions()*.

Type
int

class telegram.constants.StickerFormat(*values)

Bases: str, enum.Enum

This enum contains the available formats of *telegram.Sticker* in the set. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 20.2.

ANIMATED = 'animated'

Animated sticker.

Type
str

STATIC = 'static'

Static sticker.

Type
str

VIDEO = 'video'

Video sticker.

Type
str

class telegram.constants.StickerLimit(*values)

Bases: enum.IntEnum

This enum contains limitations for various sticker methods, such as *telegram.Bot.create_new_sticker_set()*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

MAX_KEYWORD_LENGTH = 64

Maximum number of characters in a search keyword for a sticker, for each item in *keywords* sequence of *telegram.Bot.set_sticker_keywords()*.

Added in version 20.2.

Type
int

MAX_NAME_AND_TITLE = 64

Maximum number of characters in a str passed as the *name* parameter or the *title* parameter of *telegram.Bot.create_new_sticker_set()*.

Type
int

MAX_SEARCH_KEYWORDS = 20

Maximum number of search keywords for a sticker, passed as the *keywords* parameter of *telegram.Bot.set_sticker_keywords()*.

Added in version 20.2.

Type
int

MAX_STICKER_EMOJI = 20

Maximum number of emojis associated with a sticker, passed as the *emoji_list* parameter of *telegram.Bot.set_sticker_emoji_list()*.

Added in version 20.2.

Type
int

MIN_NAME_AND_TITLE = 1

Minimum number of characters in a str passed as the *name* parameter or the *title* parameter of *telegram.Bot.create_new_sticker_set()*.

Type
int

MIN_STICKER_EMOJI = 1

Minimum number of emojis associated with a sticker, passed as the *emoji_list* parameter of *telegram.Bot.set_sticker_emoji_list()*.

Added in version 20.2.

Type
int

class telegram.constants.**StickerSetLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for various sticker set methods, such as *telegram.Bot.create_new_sticker_set()* and *telegram.Bot.add_sticker_to_set()*.

The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.2.

MAX_ANIMATED_STICKERS = 50

Maximum number of stickers allowed in an animated or video sticker set, as given in *telegram.Bot.add_sticker_to_set()*.

Deprecated since version 21.1: The animated sticker limit is now 120, the same as *MAX_STATIC_STICKERS*.

Type
int

MAX_ANIMATED_THUMBNAIL_SIZE = 32

Maximum size of the thumbnail if it is a .TGS or .WEBM in kilobytes, as given in *telegram.Bot.set_sticker_set_thumbnail()*.

Type
int

MAX_EMOJI_STICKERS = 200

Maximum number of stickers allowed in an emoji sticker set, as given in *telegram.Bot.add_sticker_to_set()*.

Type
int

MAX_INITIAL_STICKERS = 50

Maximum number of stickers allowed while creating a sticker set, passed as the *stickers* parameter of *telegram.Bot.create_new_sticker_set()*.

Type
int

MAX_STATIC_STICKERS = 120

Maximum number of stickers allowed in a static sticker set, as given in *telegram.Bot.add_sticker_to_set()*.

Type
int

MAX_STATIC_THUMBNAIL_SIZE = 128

Maximum size of the thumbnail if it is a `.WEBP` or `.PNG` in kilobytes, as given in *telegram.Bot.set_sticker_set_thumbnail()*.

Type
int

MIN_INITIAL_STICKERS = 1

Minimum number of stickers needed to create a sticker set, passed as the *stickers* parameter of *telegram.Bot.create_new_sticker_set()*.

Type
int

STATIC_THUMB_DIMENSIONS = 100

Exact height and width of the thumbnail if it is a `.WEBP` or `.PNG` in pixels, as given in *telegram.Bot.set_sticker_set_thumbnail()*.

Type
int

class telegram.constants.**StickerType**(**values*)

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.

Added 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.**StoryAreaPositionLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.StoryAreaPosition*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 22.1.

MAX_ROTATION_ANGLE = 360

Maximum value allowed for: *rotation_angle* parameter of *telegram.StoryAreaPosition*

Type
int

class telegram.constants.**StoryAreaTypeLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for subclasses of *telegram.StoryAreaType*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 22.1.

MAX_LINK_AREAS = 3

Maximum number of link areas that a story can have.

Type
int

MAX_LOCATION_AREAS = 10

Maximum number of location areas that a story can have.

Type
int

MAX_SUGGESTED_REACTION_AREAS = 5

Maximum number of suggested reaction areas that a story can have.

Type
int

MAX_UNIQUE_GIFT_AREAS = 1

Maximum number of unique gift areas that a story can have.

Type
int

MAX_WEATHER_AREAS = 3

Maximum number of weather areas that a story can have.

Type
int

class telegram.constants.**StoryAreaTypeType**(**values*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.StoryAreaType*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 22.1.

LINK = 'link'

Type of *telegram.StoryAreaTypeLink*.

Type
str

LOCATION = 'location'

Type of *telegram.StoryAreaTypeLocation*.

Type
str

SUGGESTED_REACTION = 'suggested_reaction'

Type of *telegram.StoryAreaTypeSuggestedReaction*.

Type
str

UNIQUE_GIFT = 'unique_gift'

Type of *telegram.StoryAreaTypeUniqueGift*.

Type
str

WEATHER = 'weather'

Type of *telegram.StoryAreaTypeWeather*.

Type
str

class telegram.constants.**StoryLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *post_story()* and *edit_story()*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 22.1.

ACTIVITY_ONE_DAY = 86400

Possible value for *active_period*` parameter of *telegram.Bot.post_story()*.

Type
int

ACTIVITY_SIX_HOURS = 21600

Possible value for *active_period*` parameter of *telegram.Bot.post_story()*.

Type
int

ACTIVITY_TWELVE_HOURS = 43200

Possible value for *active_period*` parameter of *telegram.Bot.post_story()*.

Type
int

ACTIVITY_TWO_DAYS = 172800

Possible value for *active_period*` parameter of *telegram.Bot.post_story()*.

Type
int

CAPTION_LENGTH = 2048

Maximum number of characters in *telegram.Bot.post_story.caption* parameter of *telegram.Bot.post_story()* and *telegram.Bot.edit_story.caption* of *telegram.Bot.edit_story()*.

Type
int

class telegram.constants.**SuggestedPost**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.SuggestedPostPrice*/*telegram.SuggestedPostParameters*/*telegram.Bot.decline_suggested_post()*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 22.4.

MAX_COMMENT_LENGTH = 128

Maximum number of characters in the *telegram.Bot.decline_suggested_post.comment* parameter.

Type
int

MAX_PRICE_NANOTONCOINS = 10000000000000

Maximum number of nanotoncoins in *amount* parameter of *telegram.SuggestedPostPrice*.

Type
int

MAX_PRICE_STARS = 100000

Maximum number of Telegram Stars in *amount* parameter of *telegram.SuggestedPostPrice*.

Type
int

MAX_SEND_DATE = 2678400

Maximum number of seconds in the future for the *send_date* parameter of *telegram.SuggestedPostParameters*.

Type
int

MIN_PRICE_NANOTONCOINS = 10000000

Minimum number of nanotoncoins in *amount* parameter of *telegram.SuggestedPostPrice*.

Type
int

MIN_PRICE_STARS = 5

Minimum number of Telegram Stars in *amount* parameter of *telegram.SuggestedPostPrice*.

Type
int

MIN_SEND_DATE = 300

Minimum number of seconds in the future for the *send_date* parameter of *telegram.SuggestedPostParameters*.

Type
int

class telegram.constants.**SuggestedPostInfoState**(**values*)

Bases: str, enum.Enum

This enum contains the available states of *telegram.SuggestedPostInfo.state*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 22.4.

APPROVED = 'approved'

Suggested post was approved.

Type
str

DECLINED = 'declined'

Suggested post was declined.

Type
str

PENDING = 'pending'

Suggested post is pending.

Type
str

class telegram.constants.**SuggestedPostRefunded**(**values*)

Bases: str, enum.Enum

This enum contains available refund reasons for *telegram.SuggestedPostRefunded*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 22.4.

PAYMENT_REFUNDED = 'payment_refunded'

The payer refunded their payment.

Type
str

POST_DELETED = 'post_deleted'

The post was deleted within 24 hours of being posted or removed from scheduled messages without being posted.

Type
str

class telegram.constants.**TagLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *set_chat_member_tag()*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 22.7.

MAX_TAG_LENGTH = 16

Maximum number of characters in a str passed as the *tag* parameter.

Type
int

class telegram.constants.**TransactionPartnerType**(**values*)

Bases: str, enum.Enum

This enum contains the available types of *telegram.TransactionPartner*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 21.4.

AFFILIATE_PROGRAM = 'affiliate_program'

Transaction with Affiliate Program.

Added in version 21.9.

Type
str

CHAT = 'chat'

Transaction with a chat.

Added in version 21.11.

Type
str

FRAGMENT = 'fragment'

Withdrawal transaction with Fragment.

Type
str

OTHER = 'other'

Transaction with unknown source or recipient.

Type
str

TELEGRAM_ADS = 'telegram_ads'

Transaction with Telegram Ads.

Type
str

TELEGRAM_API = 'telegram_api'

Transaction with with payment for paid broadcasting.

Added in version 21.7.

Type
str

USER = 'user'

Transaction with a user.

Type
str

class telegram.constants.**TransactionPartnerUser**(**values*)

Bases: str, enum.Enum

This enum contains constants for *telegram.TransactionPartnerUser*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 22.1.

BUSINESS_ACCOUNT_TRANSFER = 'business_account_transfer'

Possible value for *telegram.TransactionPartnerUser.transaction_type*.

Type
str

GIFT_PURCHASE = 'gift_purchase'

Possible value for *telegram.TransactionPartnerUser.transaction_type*.

Type
str

INVOICE_PAYMENT = 'invoice_payment'

Possible value for *telegram.TransactionPartnerUser.transaction_type*.

Type
str

PAID_MEDIA_PAYMENT = 'paid_media_payment'

Possible value for *telegram.TransactionPartnerUser.transaction_type*.

Type
str

PREMIUM_PURCHASE = 'premium_purchase'

Possible value for *telegram.TransactionPartnerUser.transaction_type*.

Type
str

class telegram.constants.**UniqueGiftInfoOrigin**(**values*)

Bases: str, enum.Enum

This enum contains the available origins for *telegram.UniqueGiftInfo*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 22.1.

GIFTED_UPGRADE = 'gifted_upgrade'

str upgrades purchased after the gift was sent

Added in version 22.6.

OFFER = 'OFFER'

str gift bought or sold through gift purchase offers

Added in version 22.6.

RESALE = 'resale'

str gift bought from other users

Added in version 22.3.

TRANSFER = 'transfer'

str gift transfered

UPGRADE = 'upgrade'

str gift upgraded

class telegram.constants.**UniqueGiftModelRarity**(**values*)

Bases: str, enum.Enum

This enum contains the available rarities for *telegram.UniqueGiftModel*. The enum members of this enumeration are instances of str and can be treated as such.

Added in version 22.7.

EPIC = 'epic'

str epic rarity

LEGENDARY = 'legendary'

str legendary rarity

RARE = 'rare'

str rare rarity

UNCOMMON = 'uncommon'

str uncommon rarity

class telegram.constants.**UpdateType**(**values*)

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.

Added in version 20.0.

BUSINESS_CONNECTION = 'business_connection'

Updates with *telegram.Update.business_connection*.

Added in version 21.1.

Type
str

BUSINESS_MESSAGE = 'business_message'

Updates with *telegram.Update.business_message*.

Added in version 21.1.

Type
str

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_BOOST = 'chat_boost'

Updates with *telegram.Update.chat_boost*.

Added in version 20.8.

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

DELETED_BUSINESS_MESSAGES = 'deleted_business_messages'

Updates with *telegram.Update.deleted_business_messages*.

Added in version 21.1.

Type
str

EDITED_BUSINESS_MESSAGE = 'edited_business_message'

Updates with *telegram.Update.edited_business_message*.

Added in version 21.1.

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

GUEST_MESSAGE = 'guest_message'

Updates with *telegram.Update.guest_message*.

Added in version 22.8.

Type
str

INLINE_QUERY = 'inline_query'

Updates with *telegram.Update.inline_query*.

Type
str

MANAGED_BOT = 'managed_bot'

Updates with *telegram.Update.managed_bot*.

Added in version 22.8.

Type
str

MESSAGE = 'message'

Updates with *telegram.Update.message*.

Type
str

MESSAGE_REACTION = 'message_reaction'

Updates with *telegram.Update.message_reaction*.

Added in version 20.8.

Type
str

MESSAGE_REACTION_COUNT = 'message_reaction_count'

Updates with *telegram.Update.message_reaction_count*.

Added in version 20.8.

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

PURCHASED_PAID_MEDIA = 'purchased_paid_media'

Updates with *telegram.Update.purchased_paid_media*.

Added in version 21.6.

Type
str

REMOVED_CHAT_BOOST = 'removed_chat_boost'

Updates with *telegram.Update.removed_chat_boost*.

Added in version 20.8.

Type
str

SHIPPING_QUERY = 'shipping_query'

Updates with *telegram.Update.shipping_query*.

Type
str

class telegram.constants.**UserProfileAudiosLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Bot.get_user_profile_audios.limit*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 22.7.

MAX_LIMIT = 100

Maximum value allowed for *limit* parameter of *telegram.Bot.get_user_profile_audios()*.

Type
int

MIN_LIMIT = 1

Minimum value allowed for *limit* parameter of *telegram.Bot.get_user_profile_audios()*.

Type
int

class telegram.constants.**UserProfilePhotosLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Bot.get_user_profile_photos.limit*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

MAX_LIMIT = 100

Maximum value allowed for *limit* parameter of *telegram.Bot.get_user_profile_photos()*.

Type
int

MIN_LIMIT = 1

Minimum value allowed for *limit* parameter of *telegram.Bot.get_user_profile_photos()*.

Type
int

class telegram.constants.**VerifyLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *verify_chat()* and *verify_user()*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 21.10.

MAX_TEXT_LENGTH = 70

Maximum number of characters in a str passed as the *custom_description* or *custom_description* parameter.

Type
int

class telegram.constants.**WebhookLimit**(**values*)

Bases: enum.IntEnum

This enum contains limitations for *telegram.Bot.set_webhook.max_connections* and *telegram.Bot.set_webhook.secret_token*. The enum members of this enumeration are instances of int and can be treated as such.

Added in version 20.0.

MAX_CONNECTIONS_LIMIT = 100

Maximum value allowed for the *max_connections* parameter of *telegram.Bot.set_webhook()*.

Type
int

MAX_SECRET_TOKEN_LENGTH = 256

Maximum length of the secret token for the *secret_token* parameter of *telegram.Bot.set_webhook()*.

Type
int

MIN_CONNECTIONS_LIMIT = 1

Minimum value allowed for the *max_connections* parameter of *telegram.Bot.set_webhook()*.

Type
int

MIN_SECRET_TOKEN_LENGTH = 1

Minimum length of the secret token for the *secret_token* parameter of *telegram.Bot.set_webhook()*.

Type
int

telegram.constants.**ZERO_DATE = datetime.datetime(1970, 1, 1, 0, 0, tzinfo=datetime.timezone.utc)**

Added in version 20.8.

6.3.2 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.

See also

Storing Bot, User and Chat Related Data

Parameters
new_chat_id (int) – The new chat id of the group.

new_chat_id

The new chat id of the group.

Type
int

__reduce__()

Defines how to serialize the exception for pickle.

See also

object.__reduce__(), pickle.

Returns
tuple

exception telegram.error.**Conflict**(*message*)

Bases: *telegram.error.TelegramError*

Raised when a long poll or webhook conflicts with another one.

__reduce__()

Defines how to serialize the exception for pickle.

See also

object.__reduce__(), pickle.

Returns
tuple

exception telegram.error.**EndPointNotFound**(*message*)

Bases: *telegram.error.TelegramError*

Raised when the requested endpoint is not found. Only relevant for *telegram.Bot.do_api_request()*.

Added in version 20.8.

exception telegram.error.**Forbidden**(*message*)

Bases: *telegram.error.TelegramError*

Raised when the bot has not enough rights to perform the requested action.

Examples

Raw API Bot

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.

Added in version 20.0.

exception telegram.error.**NetworkError**(*message*)

Bases: *telegram.error.TelegramError*

Base class for exceptions due to networking errors.

Tip

This exception (and its subclasses) usually originates from the networking backend used by *HTTPXRequest*, or a custom implementation of *BaseRequest*. In this case, the original exception can be accessed via the `__cause__` attribute.

Examples

Raw API Bot

See also

Handling network 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`.

__reduce__()

Defines how to serialize the exception for pickle.

See also

`object.__reduce__()`, `pickle`.

Returns

tuple

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 | datetime.timedelta) – Time in seconds, after which the bot can retry the request.

Changed in version v22.2: `datetime.timedelta` objects are accepted in addition to plain `int` values.

retry_after

Time in seconds, after which the bot can retry the request.

Deprecated since version v22.2: In a future major version this attribute will be of type `datetime.timedelta`. You can opt-in early by setting *PTB_TIMEDELTA=true* or `PTB_TIMEDELTA=1` as an environment variable.

Type
`int | datetime.timedelta`

__reduce__()

Defines how to serialize the exception for pickle.

See also

`object.__reduce__()`, `pickle`.

Returns
`tuple`

exception telegram.error.**TelegramError**(*message*)

Bases: `Exception`

Base class for Telegram errors.

Tip

Objects of this type can be serialized via Python's `pickle` module and pickled objects from one version of PTB are usually loadable in future versions. However, we can not guarantee that this compatibility will always be provided. At least a manual one-time conversion of the data may be needed on major updates of the library.

See also

Exceptions, Warnings and Logging

__reduce__()

Defines how to serialize the exception for pickle.

See also

`object.__reduce__()`, `pickle`.

Returns
`tuple`

__repr__()

Gives an unambiguous string representation of the exception.

Returns
`str`

__str__()

Gives the string representation of exceptions message.

Returns
str

exception telegram.error.**TimedOut**(*message=None*)

Bases: *telegram.error.NetworkError*

Raised when a request took too long to finish.

See also

Handling network errors

Parameters
message (str, optional) – Any additional information about the exception.

Added in version 20.0.

6.3.3 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/features#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")
- *Deep Linking*

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

Raises
ValueError – If the length of the *payload* exceeds *64* characters, contains invalid characters, or if the *bot_username* is less than 4 characters.

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.

Changed in version 20.3: Custom emoji entity escaping is now supported.

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'* and *'custom_emoji'*, 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*)

Helper function to create a user mention as HTML tag.

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*)

Helper function to create a user mention in Markdown syntax.

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`

6.3.4 telegram.request Module

Added in version 20.0.

BaseRequest

class telegram.request.**BaseRequest**

Bases: contextlib.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()
```

See also

__aenter__() and *__aexit__()*.

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*.

See also

Architecture Overview, Builder Pattern

Use In

- *telegram.ext.ApplicationBuilder.get_updates_request()*
- *telegram.ext.ApplicationBuilder.request()*

Available In

telegram.Bot.request

Added 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 v22.8 (https://python-telegram-bot.org)'

A description that can be used as user agent for requests made to the Bot API.

Type
str

async __aenter__()

Asynchronous context manager which *initializes* the Request.

Returns
The initialized Request instance.

Raises
Exception – If an exception is raised during initialization, *shutdown()* is called in this case.

async __aexit__(*exc_type*, *exc_val*, *exc_tb*)

Asynchronous context manager which *shuts down* the Request.

abstractmethod 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]

abstractmethod 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

final 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.

abstract property read_timeout

This property must return the default read timeout in seconds used by this class. More precisely, the returned value should be the one used when *post.read_timeout* of :meth:post` is not passed/equal to *DEFAULT_NONE*.

Added in version 20.7.

Changed in version 22.0: This property is now required to be implemented by subclasses.

Returns
The read timeout in seconds.

Return type
float | None

final 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

abstractmethod async shutdown()

Stop & clear resources used by this class. Must be implemented by a subclass.

RequestData

final 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.

Added in version 20.0.

> **Warning**
>
> How exactly instances of this 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.

Returns
dict[str, str]

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.

Returns
bytes

property multipart_data

Gives the files contained in this object as mapping of part name to encoded content.

Changed in version 21.5: Content may now be a file handle.

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.

Returns
dict[str, str | int | list[any] | dict[any, any]]

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().

Returns

str

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().

Returns

str

HTTPXRequest

class telegram.request.**HTTPXRequest**(*connection_pool_size=256*, *read_timeout=5.0*, *write_timeout=5.0*, *connect_timeout=5.0*, *pool_timeout=1.0*, *http_version='1.1'*, *socket_options=None*, *proxy=None*, *media_write_timeout=20.0*, *httpx_kwargs=None*)

Bases: *telegram.request.BaseRequest*

Implementation of *BaseRequest* using the library httpx.

Use In

- *telegram.ext.ApplicationBuilder.get_updates_request()*
- *telegram.ext.ApplicationBuilder.request()*

Available In

telegram.Bot.request

Added in version 20.0.

Changed in version 22.0: Removed the deprecated parameter proxy_url. Use *proxy* instead.

Parameters

- ***connection_pool_size*** (int, optional) – Number of connections to keep in the connection pool. Defaults to 256.

  Changed in version 22.4: Set the default to 256. Stopped applying to httpx.Limits.max_keepalive_connections. Now only applies to httpx.Limits.max_connections. See Resource Limits

- ***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.

Hint

This timeout is used for all requests except for those that upload media/files. For the latter, *media_write_timeout* is used.

- ***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!

- ***http_version*** (str, optional) – If "2" or "2.0", HTTP/2 will be used instead of HTTP/1.1. Defaults to "1.1".

  Added in version 20.1.

  Changed in version 20.2: Reset the default version to 1.1.

  Changed in version 20.5: Accept "2" as a valid value.
- ***socket_options*** (Collection[tuple], optional) – Socket options to be passed to the underlying library.

Note

The values accepted by this parameter depend on the operating system. This is a low-level parameter and should only be used if you are familiar with these concepts.

Added in version 20.7.

- ***proxy*** (str | httpx.Proxy | httpx.URL, optional) – The URL to a proxy server, a httpx.Proxy object or a httpx.URL object. 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.
- HTTPS proxies can be configured by passing a httpx.Proxy object with a corresponding ssl_context.
- 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.

Added in version 20.7.

- ***media_write_timeout*** (float | None, optional) – Like *write_timeout*, but used only for requests that upload media/files. This value is used unless a different value is passed to *do_request.write_timeout* of *do_request()*. Defaults to 20 seconds.

  Added in version 21.0.

- ***httpx_kwargs*** (dict[str, Any], optional) – Additional keyword arguments to be passed to the httpx.AsyncClient constructor.

  Warning

  This parameter is intended for advanced users that want to fine-tune the behavior of the underlying `httpx` client. The values passed here will override all the defaults set by `python-telegram-bot` and all other parameters passed to *HTTPXRequest*. The only exception is the *media_write_timeout* parameter, which is not passed to the client constructor. No runtime warnings will be issued about parameters that are overridden in this way.

  Added in version 21.6.

async do_request(*url*, *method*, *request_data=None*, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*, *pool_timeout=None*)

See *BaseRequest.do_request()*.

property http_version

Used HTTP version, see *http_version*.

Added in version 20.2.

Type
str

async initialize()

See *BaseRequest.initialize()*.

property read_timeout

See *BaseRequest.read_timeout*.

Returns

The default read timeout in seconds as passed to *HTTPXRequest.read_timeout*.

Return type
float | None

async shutdown()

See *BaseRequest.shutdown()*.

6.3.5 telegram.warnings Module

This module contains classes used for warnings issued by this library.

Added in version 20.0.

exception telegram.warnings.**PTBDeprecationWarning**(*version*, *message*)

Bases: *telegram.warnings.PTBUserWarning*, DeprecationWarning

Custom warning class for deprecations in this library.

Changed in version 20.0: Renamed TelegramDeprecationWarning to PTBDeprecationWarning.

Parameters

- **version** (str) – The version in which the feature was deprecated.

  Added in version 21.2.

- **message** (str) – The message to display.

  Added in version 21.2.

version

The version in which the feature was deprecated.

Added in version 21.2.

Type
str

message

The message to display.

Added in version 21.2.

Type
str

__str__()

Returns a string representation of the warning, using *message* and *version*.

Added in version 21.2.

exception telegram.warnings.**PTBRuntimeWarning**

Bases: *telegram.warnings.PTBUserWarning*, RuntimeWarning

Custom runtime warning class used for warnings in this library.

Added in version 20.0.

exception telegram.warnings.**PTBUserWarning**

Bases: UserWarning

Custom user warning class used for warnings in this library.

See also

Exceptions, Warnings and Logging

Added in version 20.0.

6.4 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.

6.4.1 *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.

6.4.2 *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. Note: To use `JobQueue`, you must install PTB via `pip install "python-telegram-bot[job-queue]"`

6.4.3 *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*.

6.4.4 *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!

6.4.5 *nestedconversationbot.py*

An even more complex example of a bot that uses the nested `ConversationHandlers`. While it's certainly not that complex that you couldn't built it without nested `ConversationHanldlers`, it gives a good impression on how to work with them. Of course, there is a *fancy state diagram* for this example, too!

6.4.6 *persistentconversationbot.py*

A basic example of a bot store conversation state and user_data over multiple restarts.

6.4.7 *inlinekeyboard.py*

This example sheds some light on inline keyboards, callback queries and message editing. A wiki site explaining this examples lives here.

6.4.8 *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.

6.4.9 *deeplinking.py*

A basic example on how to use deeplinking with inline keyboards.

6.4.10 *inlinebot.py*

A basic example of an inline bot. Don't forget to enable inline mode with @BotFather.

6.4.11 *pollbot.py*

This example sheds some light on polls, poll answers and the corresponding handlers.

6.4.12 *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. Note: To use Telegram Passport, you must install PTB via `pip install "python-telegram-bot[passport]"`

6.4.13 *paymentbot.py*

A basic example of a bot that can accept payments. Don't forget to enable and configure payments with @BotFather.

6.4.14 *errorhandlerbot.py*

A basic example on how to set up a custom error handler.

6.4.15 *chatmemberbot.py*

A basic example on how `(my_)chat_member` updates can be used.

6.4.16 *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.

6.4.17 *contexttypesbot.py*

This example showcases how `telegram.ext.ContextTypes` can be used to customize the `context` argument of handler and job callbacks.

6.4.18 *customwebhookbot.py*

This example showcases how a custom webhook setup can be used in combination with `telegram.ext.Application`.

6.4.19 *arbitrarycallbackdatabot.py*

This example showcases how PTBs "arbitrary callback data" feature can be used. Note: To use arbitrary callback data, you must install PTB via `pip install "python-telegram-bot[callback-data]"`

6.4.20 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
# 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

Note:
To use arbitrary callback data, you must install PTB via
`pip install "python-telegram-bot[callback-data]"`
"""
```

(continues on next page)

(continued from previous page)

```

import logging
from typing import cast

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
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

logger = logging.getLogger(__name__)


async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """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:
```

(continues on next page)

(continued from previous page)

```
    """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␣
↪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(allowed_updates=Update.ALL_TYPES)


if __name__ == "__main__":
```

(continues on next page)

(continued from previous page)

```
    main()
```

chatmemberbot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument
# 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 telegram import Chat, ChatMember, ChatMemberUpdated, Update
from telegram.constants import ParseMode
from telegram.ext import (
    Application,
    ChatMemberHandler,
    CommandHandler,
    ContextTypes,
    MessageHandler,
    filters,
)

# Enable logging

logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)

# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

logger = logging.getLogger(__name__)


def extract_status_change(chat_member_update: ChatMemberUpdated) -> tuple[bool, bool]␣
↪| None:
    """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
```

(continues on next page)

(continued from previous page)

```
    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)
    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:
            # This may not be really needed in practice because most clients will␣
↪automatically
            # send a /start command after the user unblocks the bot, and start_
↪private_chat()
            # will add the user to "user_ids".
            # We're including this here for the sake of the example.
            logger.info("%s unblocked 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)
    elif 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)


```

(continues on next page)

(continued from previous page)

```
102
103 async def show_chats(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
104     """Shows which chats the bot is in"""
105     user_ids = ", ".join(str(uid) for uid in context.bot_data.setdefault("user_ids",␣
    ↪set()))
106     group_ids = ", ".join(str(gid) for gid in context.bot_data.setdefault("group_ids",
    ↪ set()))
107     channel_ids = ", ".join(str(cid) for cid in context.bot_data.setdefault("channel_
    ↪ids", set()))
108     text = (
109         f"@{context.bot.username} is currently in a conversation with the user IDs
    ↪{user_ids}."
110         f" Moreover it is a member of the groups with IDs {group_ids} "
111         f"and administrator in the channels with IDs {channel_ids}."
112     )
113     await update.effective_message.reply_text(text)
114
115
116 async def greet_chat_members(update: Update, context: ContextTypes.DEFAULT_TYPE) ->␣
    ↪None:
117     """Greets new users in chats and announces when someone leaves"""
118     result = extract_status_change(update.chat_member)
119     if result is None:
120         return
121
122     was_member, is_member = result
123     cause_name = update.chat_member.from_user.mention_html()
124     member_name = update.chat_member.new_chat_member.user.mention_html()
125
126     if not was_member and is_member:
127         await update.effective_chat.send_message(
128             f"{member_name} was added by {cause_name}. Welcome!",
129             parse_mode=ParseMode.HTML,
130         )
131     elif was_member and not is_member:
132         await update.effective_chat.send_message(
133             f"{member_name} is no longer with us. Thanks a lot, {cause_name} ...",
134             parse_mode=ParseMode.HTML,
135         )
136
137
138 async def start_private_chat(update: Update, context: ContextTypes.DEFAULT_TYPE) ->␣
    ↪None:
139     """Greets the user and records that they started a chat with the bot if it's a␣
    ↪private chat.
140     Since no `my_chat_member` update is issued when a user starts a private chat with␣
    ↪the bot
141     for the first time, we have to track it explicitly here.
142     """
143     user_name = update.effective_user.full_name
144     chat = update.effective_chat
145     if chat.type != Chat.PRIVATE or chat.id in context.bot_data.get("user_ids",␣
    ↪set()):
146         return
147
148     logger.info("%s started a private chat with the bot", user_name)
```

(continues on next page)

(continued from previous page)

```
    context.bot_data.setdefault("user_ids", set()).add(chat.id)

    await update.effective_message.reply_text(
        f"Welcome {user_name}. Use /show_chats to see what chats I'm in."
    )


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))

    # Interpret any other command or text message as a start of a private chat.
    # This will record the user as being in a private chat with bot.
    application.add_handler(MessageHandler(filters.ALL, start_private_chat))

    # 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
# 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.
"""

import logging
from collections import defaultdict

from telegram import InlineKeyboardButton, InlineKeyboardMarkup, Update
from telegram.constants import ParseMode
from telegram.ext import (
    Application,
    CallbackContext,
```

(continues on next page)

(continued from previous page)

```
    CallbackQueryHandler,
    CommandHandler,
    ContextTypes,
    ExtBot,
    TypeHandler,
)

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

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 = None,
        user_id: int | None = None,
    ):
        super().__init__(application=application, chat_id=chat_id, user_id=user_id)
        self._message_id: int | None = None

    @property
    def bot_user_ids(self) -> set[int]:
        """Custom shortcut to access a value stored in the bot_data dict"""
        return self.bot_data.setdefault("user_ids", set())

    @property
    def message_clicks(self) -> int | None:
        """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.")
```

(continues on next page)

(continued from previous page)

```
        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,
    )


async def print_users(update: Update, context: CustomContext) -> None:
    """Show which users have been using this bot."""
    await update.message.reply_text(
        f"The following user IDs have used this bot: {', '.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:
```

(continues on next page)

(continued from previous page)

```
    """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(allowed_updates=Update.ALL_TYPES)


if __name__ == "__main__":
    main()
```

conversationbot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument
# 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 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
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

logger = logging.getLogger(__name__)

```

(continues on next page)

(continued from previous page)

```
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(),
    )

    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_to_drive("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
```

(continues on next page)

(continued from previous page)

```


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


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)],
```

(continues on next page)

(continued from previous page)

```
        states={
            # Use case-insensitive regex to accept gender input regardless of letter casing,
            # e.g., "boy", "BOY", "Girl", etc., will all be matched
            GENDER: [MessageHandler(filters.Regex("(?i)^(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(allowed_updates=Update.ALL_TYPES)


if __name__ == "__main__":
    main()
```

State Diagram

conversationbot2.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument
# 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 ReplyKeyboardMarkup, ReplyKeyboardRemove, Update
from telegram.ext import (
    Application,
    CommandHandler,
    ContextTypes,
    ConversationHandler,
    MessageHandler,
    filters,
)
```

(continues on next page)

(continued from previous page)

```
28
29  # Enable logging
30  logging.basicConfig(
31      format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
32  )
33  # set higher logging level for httpx to avoid all GET and POST requests being logged
34  logging.getLogger("httpx").setLevel(logging.WARNING)
35
36  logger = logging.getLogger(__name__)
37
38  CHOOSING, TYPING_REPLY, TYPING_CHOICE = range(3)
39
40  reply_keyboard = [
41      ["Age", "Favourite colour"],
42      ["Number of siblings", "Something else..."],
43      ["Done"],
44  ]
45  markup = ReplyKeyboardMarkup(reply_keyboard, one_time_keyboard=True)
46
47
48  def facts_to_str(user_data: dict[str, str]) -> str:
49      """Helper function for formatting the gathered user info."""
50      facts = [f"{key} - {value}" for key, value in user_data.items()]
51      return "\n".join(facts).join(["\n", "\n"])
52
53
54  async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
55      """Start the conversation and ask user for input."""
56      await update.message.reply_text(
57          "Hi! My name is Doctor Botter. I will hold a more complex conversation with␣
    ↪you. "
58          "Why don't you tell me something about yourself?",
59          reply_markup=markup,
60      )
61
62      return CHOOSING
63
64
65  async def regular_choice(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
66      """Ask the user for info about the selected predefined choice."""
67      text = update.message.text
68      context.user_data["choice"] = text
69      await update.message.reply_text(f"Your {text.lower()}? Yes, I would love to hear␣
    ↪about that!")
70
71      return TYPING_REPLY
72
73
74  async def custom_choice(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
75      """Ask the user for a description of a custom category."""
76      await update.message.reply_text(
77          'Alright, please send me the category first, for example "Most impressive␣
    ↪skill"'
78      )
79
80      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."""
    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"
        " 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$")),␣
```

(continues on next page)

(continued from previous page)

```
↪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)

    # Run the bot until the user presses Ctrl-C
    application.run_polling(allowed_updates=Update.ALL_TYPES)


if __name__ == "__main__":
    main()
```

State Diagram

customwebhookbot.py

This example is available for different web frameworks. You can select your preferred framework by opening one of the tabs above the code example.

Hint

The following examples show how different Python web frameworks can be used alongside PTB. This can be useful for two use cases:

1. For extending the functionality of your existing bot to handling updates of external services
2. For extending the functionality of your exisiting web application to also include chat bot functionality

How the PTB and web framework components of the examples below are viewed surely depends on which use case one has in mind. We are fully aware that a combination of PTB with web frameworks will always mean finding a tradeoff between usability and best practices for both PTB and the web framework and these examples are certainly far from optimal solutions. Please understand them as starting points and use your expertise of the web framework of your choosing to build up on them. You are of course also very welcome to help improve these examples!

```
#!/usr/bin/env python
# This program is dedicated to the public domain under the CC0 license.
# pylint: disable=import-error,unused-argument
"""
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~=0.23.2`.
Note that any other `asyncio` based web server framework can be used for a custom
↪webhook setup
just as well.
```

(continues on next page)

(continued from previous page)

```
10
11  Usage:
12  Set bot Token, URL, admin CHAT_ID and PORT after the imports.
13  You may also need to change the `listen` value in the uvicorn configuration to match␣
    ↪your setup.
14  Press Ctrl-C on the command line or send a signal to the process to stop the bot.
15  """
16
17  import asyncio
18  import html
19  import logging
20  from dataclasses import dataclass
21  from http import HTTPStatus
22
23  import uvicorn
24  from starlette.applications import Starlette
25  from starlette.requests import Request
26  from starlette.responses import PlainTextResponse, Response
27  from starlette.routing import Route
28
29  from telegram import Update
30  from telegram.constants import ParseMode
31  from telegram.ext import (
32      Application,
33      CallbackContext,
34      CommandHandler,
35      ContextTypes,
36      ExtBot,
37      TypeHandler,
38  )
39
40  # Enable logging
41  logging.basicConfig(
42      format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
43  )
44  # set higher logging level for httpx to avoid all GET and POST requests being logged
45  logging.getLogger("httpx").setLevel(logging.WARNING)
46
47  logger = logging.getLogger(__name__)
48
49  # Define configuration constants
50  URL = "https://domain.tld"
51  ADMIN_CHAT_ID = 123456
52  PORT = 8000
53  TOKEN = "123:ABC"  # nosec B105
54
55
56  @dataclass
57  class WebhookUpdate:
58      """Simple dataclass to wrap a custom update type"""
59
60      user_id: int
61      payload: str
62
63
64  class CustomContext(CallbackContext[ExtBot, dict, dict, dict]):
```

(continues on next page)

(continued from previous page)

```
    """
    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."""
    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>."
    )
    await update.message.reply_html(text=text)


async def webhook_update(update: WebhookUpdate, context: CustomContext) -> None:
    """Handle 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=ADMIN_CHAT_ID, text=text, parse_
↪mode=ParseMode.HTML)


async def main() -> None:
    """Set up PTB application and a web application for handling the incoming␣
↪requests."""
    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()
    )

```

(continues on next page)

(continued from previous page)

```
    # 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", allowed_updates=Update.
↪ALL_TYPES)

    # 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:
        """
        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,
```

(continues on next page)

(continued from previous page)

```
            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())
```

```
#!/usr/bin/env python
# This program is dedicated to the public domain under the CC0 license.
# pylint: disable=import-error,unused-argument
"""
Simple example of a bot that uses a custom webhook setup and handles custom updates.
For the custom webhook setup, the libraries `flask`, `asgiref` and `uvicorn` are used. Please
install them as `pip install flask[async]~=2.3.2 uvicorn~=0.23.2 asgiref~=3.7.2`.
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 after the imports.
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 asgiref.wsgi import WsgiToAsgi
from flask import Flask, Response, abort, make_response, request

from telegram import Update
from telegram.constants import ParseMode
from telegram.ext import (
    Application,
    CallbackContext,
    CommandHandler,
    ContextTypes,
    ExtBot,
    TypeHandler,
)

```

(continues on next page)

(continued from previous page)

```
# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

logger = logging.getLogger(__name__)

# Define configuration constants
URL = "https://domain.tld"
ADMIN_CHAT_ID = 123456
PORT = 8000
TOKEN = "123:ABC"  # nosec B105


@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."""
    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>."
    )
    await update.message.reply_html(text=text)


async def webhook_update(update: WebhookUpdate, context: CustomContext) -> None:
    """Handle custom updates."""
    chat_member = await context.bot.get_chat_member(chat_id=update.user_id, user_
```

(continues on next page)

(continued from previous page)

```
↪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=ADMIN_CHAT_ID, text=text, parse_
↪mode=ParseMode.HTML)


async def main() -> None:
    """Set up PTB application and a web application for handling the incoming␣
↪requests."""
    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()
    )

    # 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", allowed_updates=Update.
↪ALL_TYPES)

    # Set up webserver
    flask_app = Flask(__name__)

    @flask_app.post("/telegram")  # type: ignore[untyped-decorator]
    async def telegram() -> Response:
        """Handle incoming Telegram updates by putting them into the `update_queue`"""
        await application.update_queue.put(Update.de_json(data=request.json,␣
↪bot=application.bot))
        return Response(status=HTTPStatus.OK)

    @flask_app.route("/submitpayload", methods=["GET", "POST"])  # type:␣
↪ignore[untyped-decorator]
    async def custom_updates() -> Response:
        """
        Handle incoming webhook updates by also putting them into the `update_queue`␣
↪if
        the required parameters were passed correctly.
        """
        try:
            user_id = int(request.args["user_id"])
            payload = request.args["payload"]
        except KeyError:
            abort(
```

(continues on next page)

(continued from previous page)

```
                HTTPStatus.BAD_REQUEST,
                "Please pass both `user_id` and `payload` as query parameters.",
            )
        except ValueError:
            abort(HTTPStatus.BAD_REQUEST, "The `user_id` must be a string!")

        await application.update_queue.put(WebhookUpdate(user_id=user_id,␣
↪payload=payload))
        return Response(status=HTTPStatus.OK)

    @flask_app.get("/healthcheck")  # type: ignore[untyped-decorator]
    async def health() -> Response:
        """For the health endpoint, reply with a simple plain text message."""
        response = make_response("The bot is still running fine :)", HTTPStatus.OK)
        response.mimetype = "text/plain"
        return response

    webserver = uvicorn.Server(
        config=uvicorn.Config(
            app=WsgiToAsgi(flask_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())
```

```
#!/usr/bin/env python
# This program is dedicated to the public domain under the CC0 license.
# pylint: disable=import-error,unused-argument
"""
Simple example of a bot that uses a custom webhook setup and handles custom updates.
For the custom webhook setup, the libraries `quart` and `uvicorn` are used. Please
install them as `pip install quart~=0.18.4 uvicorn~=0.23.2`.
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 after the imports.
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
```

(continues on next page)

(continued from previous page)

```
import logging
from dataclasses import dataclass
from http import HTTPStatus

import uvicorn
from quart import Quart, Response, abort, make_response, request

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
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

logger = logging.getLogger(__name__)

# Define configuration constants
URL = "https://domain.tld"
ADMIN_CHAT_ID = 123456
PORT = 8000
TOKEN = "123:ABC"  # nosec B105


@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)
```

(continues on next page)

(continued from previous page)

```
        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."""
    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>."
    )
    await update.message.reply_html(text=text)


async def webhook_update(update: WebhookUpdate, context: CustomContext) -> None:
    """Handle 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=ADMIN_CHAT_ID, text=text, parse_
↪mode=ParseMode.HTML)


async def main() -> None:
    """Set up PTB application and a web application for handling the incoming␣
↪requests."""
    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()
    )

    # 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", allowed_updates=Update.
↪ALL_TYPES)

    # Set up webserver
    quart_app = Quart(__name__)

    @quart_app.post("/telegram")  # type: ignore[untyped-decorator]
    async def telegram() -> Response:
```

(continues on next page)

(continued from previous page)

```
        """Handle incoming Telegram updates by putting them into the `update_queue`"""
        await application.update_queue.put(
            Update.de_json(data=await request.get_json(), bot=application.bot)
        )
        return Response(status=HTTPStatus.OK)

    @quart_app.route("/submitpayload", methods=["GET", "POST"])  # type: ignore[untyped-decorator]
    async def custom_updates() -> Response:
        """
        Handle incoming webhook updates by also putting them into the `update_queue` if
        the required parameters were passed correctly.
        """
        try:
            user_id = int(request.args["user_id"])
            payload = request.args["payload"]
        except KeyError:
            abort(
                HTTPStatus.BAD_REQUEST,
                "Please pass both `user_id` and `payload` as query parameters.",
            )
        except ValueError:
            abort(HTTPStatus.BAD_REQUEST, "The `user_id` must be a string!")

        await application.update_queue.put(WebhookUpdate(user_id=user_id, payload=payload))
        return Response(status=HTTPStatus.OK)

    @quart_app.get("/healthcheck")  # type: ignore[untyped-decorator]
    async def health() -> Response:
        """For the health endpoint, reply with a simple plain text message."""
        response = await make_response("The bot is still running fine :)", HTTPStatus.OK)
        response.mimetype = "text/plain"
        return response

    webserver = uvicorn.Server(
        config=uvicorn.Config(
            app=quart_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())
```

```python
#!/usr/bin/env python
# This program is dedicated to the public domain under the CC0 license.
# pylint: disable=import-error,unused-argument
"""
Simple example of a bot that uses a custom webhook setup and handles custom updates.
For the custom webhook setup, the libraries `Django` and `uvicorn` are used. Please
install them as `pip install Django~=4.2.4 uvicorn~=0.23.2`.
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 after the imports.
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 json
import logging
from dataclasses import dataclass
from uuid import uuid4

import uvicorn
from django.conf import settings
from django.core.asgi import get_asgi_application
from django.http import HttpRequest, HttpResponse, HttpResponseBadRequest
from django.urls import path

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
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

logger = logging.getLogger(__name__)

# Define configuration constants
URL = "https://domain.tld"
ADMIN_CHAT_ID = 123456
PORT = 8000
TOKEN = "123:ABC"  # nosec B105

```

(continues on next page)

(continued from previous page)

```

@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."""
    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>."
    )
    await update.message.reply_html(text=text)


async def webhook_update(update: WebhookUpdate, context: CustomContext) -> None:
    """Handle 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=ADMIN_CHAT_ID, text=text, parse_
↪mode=ParseMode.HTML)


async def telegram(request: HttpRequest) -> HttpResponse:
    """Handle incoming Telegram updates by putting them into the `update_queue`"""
```

(continues on next page)

(continued from previous page)

```
    await ptb_application.update_queue.put(
        Update.de_json(data=json.loads(request.body), bot=ptb_application.bot)
    )
    return HttpResponse()


async def custom_updates(request: HttpRequest) -> HttpResponse:
    """
    Handle incoming webhook updates by also putting them into the `update_queue` if
    the required parameters were passed correctly.
    """
    try:
        user_id = int(request.GET["user_id"])
        payload = request.GET["payload"]
    except KeyError:
        return HttpResponseBadRequest(
            "Please pass both `user_id` and `payload` as query parameters.",
        )
    except ValueError:
        return HttpResponseBadRequest("The `user_id` must be a string!")

    await ptb_application.update_queue.put(WebhookUpdate(user_id=user_id,␣
↪payload=payload))
    return HttpResponse()


async def health(_: HttpRequest) -> HttpResponse:
    """For the health endpoint, reply with a simple plain text message."""
    return HttpResponse("The bot is still running fine :)")


# Set up PTB application and a web application for handling the incoming requests.

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
ptb_application = (
    Application.builder().token(TOKEN).updater(None).context_types(context_types).
↪build()
)

# register handlers
ptb_application.add_handler(CommandHandler("start", start))
ptb_application.add_handler(TypeHandler(type=WebhookUpdate, callback=webhook_update))

urlpatterns = [
    path("telegram", telegram, name="Telegram updates"),
    path("submitpayload", custom_updates, name="custom updates"),
    path("healthcheck", health, name="health check"),
]
settings.configure(ROOT_URLCONF=__name__, SECRET_KEY=uuid4().hex)


async def main() -> None:
    """Finalize configuration and run the applications."""
```

(continues on next page)

(continued from previous page)

```
    webserver = uvicorn.Server(
        config=uvicorn.Config(
            app=get_asgi_application(),
            port=PORT,
            use_colors=False,
            host="127.0.0.1",
        )
    )

    # Pass webhook settings to telegram
    await ptb_application.bot.set_webhook(url=f"{URL}/telegram", allowed_
↪updates=Update.ALL_TYPES)

    # Run application and webserver together
    async with ptb_application:
        await ptb_application.start()
        await webserver.serve()
        await ptb_application.stop()


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

deeplinking.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument
# 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 (
    InlineKeyboardButton,
    InlineKeyboardMarkup,
    LinkPreviewOptions,
    Update,
    helpers,
)
from telegram.constants import ParseMode
from telegram.ext import Application, CallbackQueryHandler, CommandHandler,␣
```

(continues on next page)

(continued from previous page)

```
↪ContextTypes, filters

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)

# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

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
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, link_preview_options=LinkPreviewOptions(is_
↪disabled=True)
    )
```

(continues on next page)

(continued from previous page)

```

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)


async def deep_linked_level_4(update: Update, context: ContextTypes.DEFAULT_TYPE) ->␣
↪None:
    """Reached through the USING_KEYBOARD payload"""
    payload = context.args
    await update.message.reply_text(
        f"Congratulations! This is as deep as it gets \n\nThe payload was: {payload}"
    )


def main() -> None:
    """Start the bot."""
    # Create the Application and pass it your bot's token.
    application = Application.builder().token("TOKEN").build()

    # More info on what deep linking actually is (read this first if it's unclear to␣
↪you):
    # https://core.telegram.org/bots/features#deep-linking

    # Register a deep-linking handler
    application.add_handler(
        CommandHandler("start", deep_linked_level_1, filters.Regex(CHECK_THIS_OUT))
    )

    # This one works with a textual link instead of an URL
    application.add_handler(CommandHandler("start", deep_linked_level_2, filters.
↪Regex(SO_COOL)))

    # We can also pass on the deep-linking payload
    application.add_handler(
        CommandHandler("start", deep_linked_level_3, filters.Regex(USING_ENTITIES))
    )

    # Possible with inline keyboard buttons as well
    application.add_handler(
```

(continues on next page)

(continued from previous page)

```
        CommandHandler("start", deep_linked_level_4, filters.Regex(USING_KEYBOARD))
    )

    # register callback handler for inline keyboard button
    application.add_handler(
        CallbackQueryHandler(deep_link_level_3_callback, pattern=KEYBOARD_
↪CALLBACKDATA)
    )

    # Make sure the deep-linking handlers occur *before* the normal /start handler.
    application.add_handler(CommandHandler("start", start))

    # Run the bot until the user presses Ctrl-C
    application.run_polling(allowed_updates=Update.ALL_TYPES)


if __name__ == "__main__":
    main()
```

echobot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument
# 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 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
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

logger = logging.getLogger(__name__)


# Define a few command handlers. These usually take the two arguments update and
# context.
```

(continues on next page)

(continued from previous page)

```python
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),
    )


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(allowed_updates=Update.ALL_TYPES)


if __name__ == "__main__":
    main()
```

errorhandlerbot.py

```python
#!/usr/bin/env python
# pylint: disable=unused-argument
# 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 Update
from telegram.constants import ParseMode
from telegram.ext import Application, CommandHandler, ContextTypes

# Enable logging
```

(continues on next page)

(continued from previous page)

```
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

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("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 = (
        "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."""
    await update.effective_message.reply_html(
        "Use /bad_command to cause an error.\n"
```

(continues on next page)

(continued from previous page)

```
        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(allowed_updates=Update.ALL_TYPES)


if __name__ == "__main__":
    main()
```

inlinebot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument
# 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 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
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)
```

(continues on next page)

(continued from previous page)

```

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 not query:  # empty query should not be handled
        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
            ),
        ),
    ]

    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))
```

(continues on next page)

(continued from previous page)

```
    # on inline queries - show corresponding inline results
    application.add_handler(InlineQueryHandler(inline_query))

    # Run the bot until the user presses Ctrl-C
    application.run_polling(allowed_updates=Update.ALL_TYPES)


if __name__ == "__main__":
    main()
```

inlinekeyboard.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument
# 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 InlineKeyboardButton, InlineKeyboardMarkup, Update
from telegram.ext import Application, CallbackQueryHandler, CommandHandler, ContextTypes

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

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:
```

(continues on next page)

(continued from previous page)

```
    """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(allowed_updates=Update.ALL_TYPES)


if __name__ == "__main__":
    main()
```

inlinekeyboard2.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument
# 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

```

(continues on next page)

(continued from previous page)

```
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
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

logger = logging.getLogger(__name__)

# Stages
START_ROUTES, END_ROUTES = range(2)
# Callback data
ONE, TWO, THREE, FOUR = range(4)


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()
```

(continues on next page)

(continued from previous page)

```
    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)),
        ]
    ]
    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)),
```

(continues on next page)

(continued from previous page)

```
            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(
        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) + "$"),
                CallbackQueryHandler(two, pattern="^" + str(TWO) + "$"),
```

(continues on next page)

(continued from previous page)

```
                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(allowed_updates=Update.ALL_TYPES)


if __name__ == "__main__":
    main()
```

nestedconversationbot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument
# 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

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
```

(continues on next page)

(continued from previous page)

```
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

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
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)),
        ],
```

(continues on next page)

(continued from previous page)

```
        [
            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_
↪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,
```

(continues on next page)

(continued from previous page)

```
↪ '-')}"
                )
        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)

    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)

```

(continues on next page)

(continued from previous page)

```
    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)

    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()
```

(continues on next page)

(continued from previous page)

```
        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


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


```

(continues on next page)

(continued from previous page)

```

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) + "$|^" + str(FEMALE) + "$"
            )
        ],
        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,
        },
```

(continues on next page)

(continued from previous page)

```
    )

    # 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) + "$"),
        CallbackQueryHandler(adding_self, pattern="^" + str(ADDING_SELF) + "$"),
        CallbackQueryHandler(end, pattern="^" + str(END) + "$"),
    ]
    conv_handler = ConversationHandler(
        entry_points=[CommandHandler("start", start)],
        states={
            SHOWING: [CallbackQueryHandler(start, pattern="^" + str(END) + "$")],
            SELECTING_ACTION: selection_handlers,  # type: ignore[dict-item]
            SELECTING_LEVEL: selection_handlers,  # type: ignore[dict-item]
            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(allowed_updates=Update.ALL_TYPES)


if __name__ == "__main__":
    main()
```

State Diagram

passportbot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument
# 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.

Note:
To use Telegram Passport, you must install PTB via
`pip install "python-telegram-bot[passport]"`
"""

import logging
```

(continues on next page)

(continued from previous page)

```
from pathlib import Path

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
)

# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

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":
            logger.info("Phone: %s", data.phone_number)
        elif data.type == "email":
            logger.info("Email: %s", data.email)
        if data.type in (
            "personal_details",
            "passport",
            "driver_license",
            "identity_card",
            "internal_passport",
            "address",
        ):
            logger.info(data.type, data.data)
        if data.type in (
            "utility_bill",
            "bank_statement",
            "rental_agreement",
            "passport_registration",
            "temporary_registration",
        ):
            logger.info(data.type, len(data.files), "files")
            for file in data.files:
                actual_file = await file.get_file()
                logger.info(actual_file)
                await actual_file.download_to_drive()
```

(continues on next page)

(continued from previous page)

```
        if (
            data.type in ("passport", "driver_license", "identity_card", "internal_passport")
            and data.front_side
        ):
            front_file = await data.front_side.get_file()
            logger.info(data.type, front_file)
            await front_file.download_to_drive()
        if data.type in ("driver_license" and "identity_card") and data.reverse_side:
            reverse_file = await data.reverse_side.get_file()
            logger.info(data.type, reverse_file)
            await reverse_file.download_to_drive()
        if (
            data.type in ("passport", "driver_license", "identity_card", "internal_passport")
            and data.selfie
        ):
            selfie_file = await data.selfie.get_file()
            logger.info(data.type, selfie_file)
            await selfie_file.download_to_drive()
        if data.translation and data.type in (
            "passport",
            "driver_license",
            "identity_card",
            "internal_passport",
            "utility_bill",
            "bank_statement",
            "rental_agreement",
            "passport_registration",
            "temporary_registration",
        ):
            logger.info(data.type, len(data.translation), "translation")
            for file in data.translation:
                actual_file = await file.get_file()
                logger.info(actual_file)
                await actual_file.download_to_drive()


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(allowed_updates=Update.ALL_TYPES)


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
# This program is dedicated to the public domain under the CC0 license.

"""Basic example for a bot that can receive payments from users."""

import logging

from telegram import LabeledPrice, ShippingOption, Update
from telegram.ext import (
    Application,
    CommandHandler,
    ContextTypes,
    MessageHandler,
    PreCheckoutQueryHandler,
    ShippingQueryHandler,
```

(continues on next page)

(continued from previous page)

```
    filters,
)

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

logger = logging.getLogger(__name__)

# Insert the token from your payment provider.
# In order to get a provider_token see https://core.telegram.org/bots/payments
↪#getting-a-token
PAYMENT_PROVIDER_TOKEN = "PAYMENT_PROVIDER_TOKEN"


async def start_callback(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Provides instructions on how to use the bot."""
    msg = (
        "Use /shipping to receive an invoice with shipping included, 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 which triggers a shipping query."""
    chat_id = update.message.chat_id
    title = "Payment Example"
    description = "Example of a payment process using the python-telegram-bot library.
↪"
    # Unique payload to identify this payment request as being from your bot
    payload = "Custom-Payload"
    # Set up the currency.
    # List of supported currencies: https://core.telegram.org/bots/payments#supported-
↪currencies
    currency = "USD"
    # Price in dollars
    price = 1
    # Convert price to cents from dollars.
    prices = [LabeledPrice("Test", price * 100)]
    # Optional parameters like need_shipping_address and is_flexible trigger extra␣
↪user prompts
    # https://docs.python-telegram-bot.org/en/stable/telegram.bot.html#telegram.Bot.
↪send_invoice
    await context.bot.send_invoice(
        chat_id,
        title,
        description,
        payload,
        currency,
        prices,
```

(continues on next page)

(continued from previous page)

```
        provider_token=PAYMENT_PROVIDER_TOKEN,
        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 requiring shipping details."""
    chat_id = update.message.chat_id
    title = "Payment Example"
    description = "Example of a payment process using the python-telegram-bot library."
    # Unique payload to identify this payment request as being from your bot
    payload = "Custom-Payload"
    currency = "USD"
    # Price in dollars
    price = 1
    # Convert price to cents from dollars.
    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,
        currency,
        prices,
        provider_token=PAYMENT_PROVIDER_TOKEN,
    )


async def shipping_callback(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Handles the ShippingQuery with available shipping options."""
    query = update.shipping_query
    # Verify if the payload matches, ensure it's from your bot
    if query.invoice_payload != "Custom-Payload":
        # If not, respond with an error
        await query.answer(ok=False, error_message="Something went wrong...")
        return

    # Define available shipping options
    # First option with a single price entry
    options = [ShippingOption("1", "Shipping Option A", [LabeledPrice("A", 100)])]
    # Second option with multiple price entries
    price_list = [LabeledPrice("B1", 150), LabeledPrice("B2", 200)]
    options.append(ShippingOption("2", "Shipping Option B", price_list))
    await query.answer(ok=True, shipping_options=options)

```

(continues on next page)

(continued from previous page)

```python

# After (optional) shipping, process the pre-checkout step
async def precheckout_callback(update: Update, context: ContextTypes.DEFAULT_TYPE) ->␣
↪None:
    """Responds to the PreCheckoutQuery as the final confirmation for checkout."""
    query = update.pre_checkout_query
    # Verify if the payload matches, ensure it's from your bot
    if query.invoice_payload != "Custom-Payload":
        # If not, respond with an error
        await query.answer(ok=False, error_message="Something went wrong...")
    else:
        await query.answer(ok=True)


# Final callback after successful payment
async def successful_payment_callback(update: Update, context: ContextTypes.DEFAULT_
↪TYPE) -> None:
    """Acknowledges successful payment and thanks the user."""
    await update.message.reply_text("Thank you for your payment.")


def main() -> None:
    """Starts the bot and sets up handlers."""
    # Create the Application and pass it your bot's token.
    application = Application.builder().token("TOKEN").build()

    # Start command to display usage instructions
    application.add_handler(CommandHandler("start", start_callback))

    # Command handlers for starting the payment process
    application.add_handler(CommandHandler("shipping", start_with_shipping_callback))
    application.add_handler(CommandHandler("noshipping", start_without_shipping_
↪callback))

    # Handler for shipping query (if product requires shipping)
    application.add_handler(ShippingQueryHandler(shipping_callback))

    # Pre-checkout handler for verifying payment details.
    application.add_handler(PreCheckoutQueryHandler(precheckout_callback))

    # Handler for successful payment. Notify the user that the payment was successful.
    application.add_handler(
        MessageHandler(filters.SUCCESSFUL_PAYMENT, successful_payment_callback)
    )

    # Start polling for updates until interrupted (CTRL+C)
    application.run_polling(allowed_updates=Update.ALL_TYPES)


if __name__ == "__main__":
    main()
```

persistentconversationbot.py

```python
#!/usr/bin/env python
# pylint: disable=unused-argument
# 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 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
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

logger = logging.getLogger(__name__)

CHOOSING, TYPING_REPLY, TYPING_CHOICE = range(3)

reply_keyboard = [
    ["Age", "Favourite colour"],
    ["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:
```

(continues on next page)

(continued from previous page)

```
    """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 "
            "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


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:"
```

(continues on next page)

(continued from previous page)

```
        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)$"),␣
↪regular_choice
                ),
                MessageHandler(filters.Regex("^Something else...$"), custom_choice),
            ],
            TYPING_CHOICE: [
                MessageHandler(
                    filters.TEXT & ~(filters.COMMAND | filters.Regex("^Done$")),␣
↪regular_choice
                )
            ],
            TYPING_REPLY: [
                MessageHandler(
                    filters.TEXT & ~(filters.COMMAND | filters.Regex("^Done$")),
```

(continues on next page)

(continued from previous page)

```
                    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(allowed_updates=Update.ALL_TYPES)


if __name__ == "__main__":
    main()
```

pollbot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument
# 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 (
    KeyboardButton,
    KeyboardButtonPollType,
    Poll,
    ReplyKeyboardMarkup,
    ReplyKeyboardRemove,
    Update,
)
from telegram.constants import ParseMode
from telegram.ext import (
    Application,
    CommandHandler,
    ContextTypes,
    MessageHandler,
    PollAnswerHandler,
    PollHandler,
    filters,
)

```

(continues on next page)

(continued from previous page)

```
# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

logger = logging.getLogger(__name__)


TOTAL_VOTER_COUNT = 3


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,
            "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 = ""
```

(continues on next page)

(continued from previous page)

```
    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"] == TOTAL_VOTER_COUNT:
        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
    if update.poll.is_closed:
        return
    if update.poll.total_voter_count == TOTAL_VOTER_COUNT:
        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)
```

(continues on next page)

(continued from previous page)

```
    )


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))

    # Run the bot until the user presses Ctrl-C
    application.run_polling(allowed_updates=Update.ALL_TYPES)


if __name__ == "__main__":
    main()
```

rawapibot.py

This example uses only the pure, “bare-metal” API wrapper.

```
#!/usr/bin/env python
"""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
```

(continues on next page)

(continued from previous page)

```
import contextlib
import datetime as dtm
import logging
from typing import NoReturn

from telegram import Bot, Update
from telegram.error import Forbidden, NetworkError

logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

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:
            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=dtm.timedelta(seconds=10), allowed_updates=Update.ALL_TYPES
    )
    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)
```

(continues on next page)

(continued from previous page)

```
        return next_update_id
    return update_id


if __name__ == "__main__":
    with contextlib.suppress(KeyboardInterrupt):  # Ignore exception when Ctrl-C is␣
↪pressed
        asyncio.run(main())
```

timerbot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument
# 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:
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.

Note:
To use the JobQueue, you must install PTB via
`pip install "python-telegram-bot[job-queue]"`
"""

import logging

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")

```

(continues on next page)

(continued from previous page)

```

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


async def set_timer(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Add a job to the queue."""
    chat_id = update.effective_message.chat_id
    try:
        # args[0] should contain the time for the timer in seconds
        due = float(context.args[0])
        if due < 0:
            await update.effective_message.reply_text("Sorry we can not go back to future!")
            return

        job_removed = remove_job_if_exists(str(chat_id), context)
        context.job_queue.run_once(alarm, due, chat_id=chat_id, name=str(chat_id), data=due)

        text = "Timer successfully set!"
        if job_removed:
            text += " Old one was removed."
        await update.effective_message.reply_text(text)

    except (IndexError, ValueError):
        await update.effective_message.reply_text("Usage: /set <seconds>")


async def unset(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Remove the job if the user changed their mind."""
    chat_id = update.message.chat_id
    job_removed = remove_job_if_exists(str(chat_id), context)
    text = "Timer successfully cancelled!" if job_removed else "You have no active timer."
    await update.message.reply_text(text)


def main() -> None:
    """Run bot."""
    # Create the Application and pass it your bot's token.
    application = Application.builder().token("TOKEN").build()

```

(continues on next page)

(continued from previous page)

```
    # on different commands - answer in Telegram
    application.add_handler(CommandHandler(["start", "help"], start))
    application.add_handler(CommandHandler("set", set_timer))
    application.add_handler(CommandHandler("unset", unset))

    # Run the bot until the user presses Ctrl-C
    application.run_polling(allowed_updates=Update.ALL_TYPES)


if __name__ == "__main__":
    main()
```

webappbot.py

```
#!/usr/bin/env python
# pylint: disable=unused-argument
# 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 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
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

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)

```
    )


# 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(allowed_updates=Update.ALL_TYPES)


if __name__ == "__main__":
    main()
```

HTML Page

```
<!--
    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),
    });
```

(continues on next page)

(continued from previous page)

```
    colorPicker.on('color:change', function (color) {
        document.body.style.background = color.hexString;
    });

    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>
```

6.5 Stability Policy

Important

This stability policy is in place since version 20.3. While earlier versions of `python-telegram-bot` also had stable interfaces, they had no explicit stability policy and hence did not follow the rules outlined below in all detail. Please also refer to the *changelog*.

Caution

Large parts of the *telegram* package are the Python representations of the Telegram Bot API, whose stability policy PTB can not influence. This policy hence includes some special cases for those parts.

6.5.1 What does this policy cover?

This policy includes any API or behavior that is covered in this documentation. This covers both the *telegram* package and the *telegram.ext* package.

6.5.2 What doesn't this policy cover?

Introduction of new features or changes of flavors of comparable behavior (e.g. the default for the HTTP protocol version being used) are not covered by this policy.

The internal structure of classes in PTB, i.e. things like the result of `dir(obj)` or the contents of `obj.__dict__`, is not covered by this policy.

Objects are in general not guaranteed to be pickleable (unless stated otherwise) and pickled objects from one version

of PTB may not be loadable in future versions. We may provide a way to convert pickled objects from one version to another, but this is not guaranteed.

Functionality that is part of PTBs API but is explicitly documented as not being intended to be used directly by users (e.g. *telegram.request.BaseRequest.do_request()*) may change. This also applies to functions or attributes marked as final in the sense of PEP 591.

PTB has dependencies to third-party packages. The versions that PTB uses of these third-party packages may change if that does not affect PTBs public API.

PTB does not give guarantees about which Python versions are supported. In general, we will try to support all Python versions that have not yet reached their end of life, but we reserve ourselves the option to drop support for Python versions earlier if that benefits the advancement of the library.

PTB provides static type hints for all public attributes, parameters, return values and generic classes. These type hints are not covered by this policy and may change at any time under the condition that these changes have no impact on the runtime behavior of PTB.

Bot API Functionality

Comparison of equality of instances of the classes in the *telegram* package is subject to change and the PTB team will update the behavior to best reflect updates in the Bot API. Changes in this regard will be documented in the affected classes. Note that equality comparison with objects that where serialized by an older version of PTB may hence give unexpected results.

When the order of arguments of the Bot API methods changes or they become optional/mandatory due to changes in the Bot API, PTB will always try to reflect these changes. While we try to make such changes backward compatible, this is not always possible or only with significant effort. In such cases we will find a trade-off between backward compatibility and fully complying with the Bot API, which may result in breaking changes. We highly recommend using keyword arguments, which can help make such changes non-breaking on your end.

When the Bot API changes attributes of classes, the method *telegram.TelegramObject.to_dict()* will change as necessary to reflect these changes. In particular, attributes deprecated by Telegram will be removed from the returned dictionary. Deprecated attributes that are still passed by Telegram will be available in the *api_kwargs* dictionary as long as PTB can support that with feasible effort. Since attributes of the classes in the *telegram* package are not writable, we may change them to properties where appropriate.

Development Versions

Pre-releases marked as alpha, beta or release candidate are not covered by this policy. Before a feature is in a stable release, i.e. the feature was merged into the `master` branch but not released yet (or only in a pre-release), it is not covered by this policy either and may change.

Security

We make exceptions from our stability policy for security. We will violate this policy as necessary in order to resolve a security issue or harden PTB against a possible attack.

6.5.3 Versioning

PTB uses a versioning scheme that roughly follows https://semver.org/, although it may not be quite as strict.

Given a version of PTB X.Y.Z,

- X indicates the major version number. This is incremented when backward incompatible changes are introduced.
- Y indicates the minor version number. This is incremented when new functionality or backward compatible changes are introduced by PTB. *This is also incremented when PTB adds support for a new Bot API version, which may include backward incompatible changes in some cases as outlined below.*
- Z is the patch version. This is incremented if backward compatible bug fixes or smaller changes are introduced. If this number is 0, it can be omitted, i.e. we just write X.Y instead of X.Y.0.

Deprecation

From time to time we will want to change the behavior of an API or remove it entirely, or we do so to comply with changes in the Telegram Bot API. In those cases, we follow a deprecation schedule as detailed below.

Functionality is marked as deprecated by a corresponding note in the release notes and the documentation. Where possible, a *PTBDeprecationWarning* is issued when deprecated functionality is used, but this is not mandatory.

From time to time, we may decide to deprecate an API that is particularly widely used. In these cases, we may decide to provide an extended deprecation period, at our discretion.

With version 20.0.0, PTB introduced major structural breaking changes without the above deprecation period. Should a similarly big change ever be deemed necessary again by the development team and should a deprecation period prove too much additional effort, this violation of the stability policy will be announced well ahead of the release in our channel, as was done for v20.

Non-Bot API Functionality

Starting with version 20.3, deprecated functionality will stay available for the current and the next major version. For example:

- In PTB v20.1.1 the feature exists
- In PTB v20.1.2 or v20.2.0 the feature is marked as deprecated
- In PTB v21.*.* the feature is marked as deprecated
- In PTB v22.0 the feature is removed or changed

Bot API Functionality

As PTB has no control over deprecations introduced by Telegram and the schedule of these deprecations rarely coincides with PTBs deprecation schedule, we have a special policy for Bot API functionality.

Starting with 20.3, deprecated Bot API functionality will stay available for the current and the next major version of PTB *or* until the next version of the Bot API. More precisely, two cases are possible, for which we show examples below.

Case 1

- In PTB v20.1 the feature exists
- Bot API version 6.6 is released and deprecates the feature
- PTB v20.2 adds support for Bot API 6.6 and the feature is marked as deprecated
- In PTB v21.0 the feature is removed or changed

Case 2

- In PTB v20.1 the feature exists
- Bot API version 6.6 is released and deprecates the feature
- PTB v20.2 adds support for Bot API version 6.6 and the feature is marked as deprecated
- In PTB v20.2.* and v20.3.* the feature is marked as deprecated
- Bot API version 6.7 is released
- PTB v20.4 adds support for Bot API version 6.7 and the feature is removed or changed

6.6 Changelog

6.6.1 Unreleased

unknown

Bug Fixes

- Fixed `parse_lpo_and_dwpp` silently overwriting `link_preview_options` when `disable_web_page_preview=False`. (#5268 by @JSap0914)

Dependencies

- Update dependency tornado to v6.5.6 [SECURITY] (#5264 by @renovate[bot])
- Update dependency tornado to v6.5.7 [SECURITY] (#5273 by @renovate[bot])
- Update dependency cryptography to v48.0.1 [SECURITY] (#5274 by @renovate[bot])

Internal Changes

- Update Ruff to v0.15.16 (#5265 by @renovate[bot])
- Update Ruff to v0.15.17 (#5276 by @renovate[bot])
- Update Pylint to v4.0.6 (#5277 by @renovate[bot])
- Fix Failing 'Lock Closed Threads' Github Action (#5278 by @harshil21, @Copilot)
- Update Ruff to v0.15.18 (#5283 by @renovate[bot])
- Update actions/checkout action to v6.0.3 (#5286 by @renovate[bot])
- Update dependency astral-sh/uv to v0.11.24 (#5287 by @renovate[bot])
- Update github/codeql-action action to v4.36.2 (#5288 by @renovate[bot])
- Update dependency pytest to v9.1.1 (#5289 by @renovate[bot])
- Update sigstore/gh-action-sigstore-python action to v3.4.0 (#5290 by @renovate[bot])
- Update Ruff to v0.15.20 (#5291 by @renovate[bot])

6.6.2 22.8

2026-06-12

Security Changes

- Disallow packages newer than last 7 days (#5247 by @harshil21 closes #5195)

Deprecations

- – Deprecated passing the `filename` parameter positionally to the classes:
    * `InputMediaAnimation`
    * `InputMediaAudio`
    * `InputMediaPhoto`
    * `InputMediaDocument`
    * `InputMediaVideo`

  Please pass `filename` as a keyword argument instead, as this parameter will become keyword-only in the future.

- Deprecated `InputPollOption.de_json`. The class `InputPollOption` is input only and its `de_json` method will be removed in future versions. The Bot API 10.0 `media` field of `InputPollOption` will not be included for deserialization.

(#5229 by @aelkheir closes #5228; #5230 by @harshil21; #5235 by @harshil21; #5238 by @harshil21; #5232 by @aelkheir; #5232 by @Poolitzer, @Phil9l, @harshil21, @aelkheir)

New Features

- Full Support for Bot API 9.6

> **Warning**
>
> - Bot API 9.6 replaces the field `correct_option_id` of `Poll` with the new field `correct_option_ids`. The field `correct_option_id` is still present in PTB for backward compatibility, but it will be removed in future releases.
> - Bot API 9.6 replaces the argument `correct_option_id` of `Bot.send_poll` with the new argument `correct_option_ids`. The argument `correct_option_id` is still present in PTB for backward compatibility, but it will be removed in future releases.
> - Bot API 9.6 introduces a now required argument `persistent_id` to `PollOption`. For backward compatibility, the argument is currently still marked as optional in the signature and its presence is enforced through a runtime check. In future versions, this argument will be made required in the signature as well.
> - Bot API 9.6 introduces a now required argument `option_persistent_ids` to `PollAnswer`. For backward compatibility, the argument is currently still marked as optional in the signature and its presence is enforced through a runtime check. In future versions, this argument will be made required in the signature as well.
> - Bot API 9.6 introduces a now required argument `allows_revoting` to `Poll`. For backward compatibility, the argument is currently still marked as optional in the signature and its presence is enforced through a runtime check. In future versions, this argument will be made required in the signature as well.
>
> Please make sure to update your code accordingly to avoid potential issues in the future. We recommend using keyword arguments to ensure compatibility with future updates.

(#5196 by @harshil21; #5202 by @ouyooung; #5197 by @harshil21)

- Full Support for Bot API 10.0

> **Warning**
>
> – Bot API 10.0 introduces a now required argument `members_only` to `Poll`. For backward compatibility, the argument is currently still marked as optional in the signature and its presence is enforced through a runtime check. In future versions, this argument will be made required in the signature as well.
>
> Please make sure to update your code accordingly to avoid potential issues in the future. We recommend using keyword arguments to ensure compatibility with future updates.

(#5229 by @aelkheir closes #5228; #5230 by @harshil21; #5235 by @harshil21; #5238 by @harshil21; #5232 by @aelkheir; #5232 by @Poolitzer, @Phil9l, @harshil21, @aelkheir)

Bug Fixes

- Fixed incorrect isinstance check in class telegram.ext.filters._MergedFilter for or_filter. (#5125 by @gistrec)
- *Update.effective_user* now checks for *channel_post* and *edited_channel_post* (#5237 by @karin0 closes #5236)

Dependencies

- Update dependency cryptography to v46.0.6 [SECURITY] (#5185 by @renovate[bot])
- Update dependency cryptography to v46.0.7 [SECURITY] (#5203 by @renovate[bot])

Other Changes

- Update codecov/codecov-action action to v5.5.4 (#5190 by @renovate[bot])
- Make instantiation of `TelegramObject`'s 15% faster (#5250 by @harshil21)
- Support Python 3.15 Beta
  - Python 3.15 free threading is not fully supported yet, as the optional dependency `cryptography` is not yet compatible with it.

  (#5259 by @harshil21 closes #5231)
- Bump Version to 22.8 (#5262 by @Poolitzer)

Documentation

- Documentation Improvements (#5240 by @harshil21, @Poolitzer; #5241 by @harshil21)

Internal Changes

- Update Ruff to v0.15.7 (#5177 by @renovate[bot])
- Migrate from deprecated test-results-action to codecov-action (#5183 by @Krishnachaitanyakc closes #5158)
- Update Ruff to v0.15.8 (#5184 by @renovate[bot])
- Update astral-sh/setup-uv action to v7.6.0 (#5191 by @renovate[bot])
- Update dependency astral-sh/uv to v0.11.2 (#5192 by @renovate[bot])
- Update dependency mypy to v1.20.0 (#5193 by @renovate[bot])
- Update github/codeql-action action to v4.35.1 (#5194 by @renovate[bot])
- Update Ruff to v0.15.9 (#5198 by @renovate[bot])
- Update Mypy to v1.20.1 (#5199 by @renovate[bot])
- Update Ruff to v0.15.10 (#5205 by @renovate[bot])

- Update dependency pytest to v9.0.3 [SECURITY] (#5206 by @renovate[bot])
- Update Ruff to v0.15.11 (#5209 by @renovate[bot])
- Update Ruff to v0.15.12 (#5217 by @renovate[bot])
- Update actions/upload-artifact action to v7.0.1 (#5220 by @renovate[bot])
- Update dependency astral-sh/uv to v0.11.8 (#5221 by @renovate[bot])
- Update github/codeql-action action to v4.35.2 (#5222 by @renovate[bot])
- Update Mypy to v1.20.2 (#5223 by @renovate[bot])
- Update jannekem/run-python-script-action action to v1.8 (#5224 by @renovate[bot])
- Update Ruff to v0.15.13 (#5233 by @renovate[bot])
- Update Ruff to v0.15.14 (#5245 by @renovate[bot])
- Update Ruff to v0.15.15 (#5252 by @renovate[bot])
- Update dependency astral-sh/uv to v0.11.16 (#5253 by @renovate[bot])
- Update dependency sphinxcontrib-mermaid to v2.0.2 (#5254 by @renovate[bot])
- Update actions/stale action to v10.3.0 (#5255 by @renovate[bot])
- Update github/codeql-action action to v4.36.0 (#5256 by @renovate[bot])
- Update pypa/gh-action-pypi-publish action to v1.14.0 (#5257 by @renovate[bot])

6.6.3 22.7

2026-03-16

Highlights

- Hand over Maintainer Role from Bibo-Joshi to Poolitzer and Harshil (#5119 by @Bibo-Joshi)

Breaking Changes

- Remove Functionality Deprecated in Bot API 9.3
  - Remove deprecated argument and attribute `UniqueGiftInfo.last_resale_star_count`.
  - Remove deprecated argument and attribute `Bot.get_business_account_gifts.exclude_limited`.
  - *`telegram.UniqueGift.gift_id`* is now a positional argument.

  (#5143 by @harshil21 closes #5093)

New Features

- Full Support for Bot API 9.4 (#5137 by @harshil21; #5133 by @ouyooung; #5141 by @hethon; #5129 by @Poolitzer; #5148 by @harshil21)
- Full support for Bot API 9.5 (#5155 by @Poolitzer)

Bug Fixes

- Preserve *InlineKeyboardButton* Arguments During *callback_data* Replacement (#5159 by @harshil21)

Dependencies

- Update dependency cryptography to v46.0.5 [SECURITY] (#5125 by @renovate[bot])
- Update dependency tornado to v6.5.5 [SECURITY] (#5164 by @renovate[bot])

Other Changes

- Fixing failing sphinx builds (#5124 by @Poolitzer)
- Fix: Moved inline test file to _inline folder (#5140 by @Poolitzer)
- Update astral-sh/setup-uv action to v7.5.0 (#5170 by @renovate[bot])
- Bump Version to v22.7 (#5176 by @harshil21)

Internal Changes

- Update Ruff to v0.14.14 (#5110 by @renovate[bot])
- Update codecov/codecov-action action to v5.5.2 (#5112 by @renovate[bot])
- Update dependency astral-sh/uv to v0.9.28 (#5113 by @renovate[bot])
- Update dependency pytest to v9.0.2 (#5114 by @renovate[bot])
- Update actions/setup-python action to v6.2.0 (#5115 by @renovate[bot])
- Update astral-sh/setup-uv action to v7.2.1 (#5116 by @renovate[bot])
- Update pre-commit hook cachetools to v7 (#5117 by @renovate[bot])
- Update Ruff to v0.15.0 (#5122 by @renovate[bot])
- Update Ruff to v0.15.0 (#5122 by @renovate[bot])
- Update Ruff to v0.15.1 (#5135 by @renovate[bot])
- Replace *pre-commit* with *prek* (#5142 by @harshil21 closes #5138)
- Update Pylint to v4.0.5 (#5145 by @renovate[bot])
- Update Ruff to v0.15.2 (#5146 by @renovate[bot])
- Update Ruff to v0.15.4 (#5150 by @renovate[bot])
- Update actions/stale action to v10.2.0 (#5151 by @renovate[bot])
- Update astral-sh/setup-uv action to v7.3.1 (#5152 by @renovate[bot])
- Update codecov/test-results-action action to v1.2.1 (#5153 by @renovate[bot])
- Update dependency astral-sh/uv to v0.10.7 (#5154 by @renovate[bot])
- Update Ruff to v0.15.5 (#5156 by @renovate[bot])
- Make CI run after chango commits (#5157 by @harshil21 closes #5144)
- Refactor *TestStickerSetWithRequest* tests (#5161 by @harshil21 closes #4514)
- Update Ruff to v0.15.6 (#5166 by @renovate[bot])
- Update dependency sphinx to v9 (#5167 by @renovate[bot])
- Bump Sphinx Related Dependencies, Fix *uv* and Docs Build (#5168 by @harshil21)
- Update dependency astral-sh/uv to v0.10.10 (#5169 by @renovate[bot])
- Update GitHub Artifact Actions (major) (#5171 by @renovate[bot])
- Update sigstore/gh-action-sigstore-python action to v3.2.0 (#5172 by @renovate[bot])
- Update stefanzweifel/git-auto-commit-action action to v7.1.0 (#5173 by @renovate[bot])
- Update actions/checkout action to v6 (#5174 by @renovate[bot])
- Lock file maintenance (#5175 by @renovate[bot])

6.6.4 22.6

2026-01-24

New Features

- Full Support for Bot API 9.3

  > **Warning**
  >
  > - Bot API 9.3 deprecates the field `last_resale_star_count` of `UniqueGiftInfo` in favor of the new fields `last_resale_currency` and `last_resale_amount`. The field `last_resale_star_count` is still present in PTB for backward compatibility, but it will be removed in future releases. Please update your code accordingly.
  > - Bot API 9.3 deprecates the argument `exclude_limited` of `Bot.get_business_account_gifts` in favor of the new arguments `exclude_limited_upgradable` and `exclude_limited_non_upgradable`. The argument `exclude_limited` is still present in PTB for backward compatibility, but it will be removed in future releases. Please update your code accordingly.
  > - Bot API 9.3 introduces a now required argument `gift_id` to `UniqueGift`. For backward compatibility, the argument is currently still marked as optional in the signature and it's presence is enforced through a runtime check. In future versions, this argument will be made required in the signature as well.
  >
  >   Please make sure to update your code accordingly to avoid potential issues in the future. We recommend using keyword arguments to ensure compatibility with future updates.

  (#5086 by @Bibo-Joshi; #5078 by @aelkheir closes #5077; #5079 by @aelkheir; #5084 by @Bibo-Joshi; #5085 by @Bibo-Joshi; #5087 by @Bibo-Joshi; #5091 by @Bibo-Joshi; #5090 by @Bibo-Joshi; #5089 by @Bibo-Joshi; #5092 by @Bibo-Joshi; #5095 by @Bibo-Joshi; #5094 by @Bibo-Joshi; #5106 by @aelkheir)

- Add Fallback Name Support for Job Callbacks without `__name__` (#5088 by @SmartDever02 closes #4992)

Bug Fixes

- Fix a Bug in Initialization Logic of `Bot` (#5030 by @codomposer closes #5021)

Other Changes

- Remove Support for Python 3.9 (#4827 by @harshil21)
- Bump Version to v22.6 (#5108 by @Bibo-Joshi)

Documentation

- Documentation Improvements. Among others, fix dead links. (#4988 by @Bibo-Joshi)
- Update Copyright to 2026 (#5075 by @Bibo-Joshi)
- Fix Broken Links in Documentation (#5080 by @calm329 closes #5074)

Internal Changes

- Lock file maintenance (#4980 by @renovatebot)
- Update github/codeql-action action to v3.30.6 (#4984 by @renovatebot)
- Update Ruff to v0.13.3 (#4985 by @renovatebot)
- Update actions/stale action to v10.1.0 (#4986 by @renovatebot)
- Update astral-sh/setup-uv action to v6.8.0 (#4987 by @renovatebot)

- Update dependency astral-sh/uv to v0.8.23 (#4989 by @renovatebot)
- Update configuration of actions/stale to use issue types instead of labels (#4990 by @Bibo-Joshi)
- Update github/codeql-action action to v3.30.8 (#4994 by @renovatebot)
- Update Pylint to v3.3.9 (#4995 by @renovatebot)
- Update dependency astral-sh/uv to v0.9.2 (#4996 by @renovatebot)
- Update Ruff to v0.14.0 (#4997 by @renovatebot)
- Update astral-sh/setup-uv action to v7 (#4998 by @renovatebot)
- Update github/codeql-action action to v4 (#4999 by @renovatebot)
- Update Pylint to v4 (major) (#5000 by @renovatebot)
- Update stefanzweifel/git-auto-commit-action action to v7 (#5001 by @renovatebot)
- Update astral-sh/setup-uv action to v7.1.0 (#5002 by @renovatebot)
- Use Python 3.14 Final in the Test Suite (#5004 by @harshil21)
- Add Freethreaded Python 3.14 to the Test Suite (#5006 by @harshil21)
- Make `chango` Commit Re-Trigger Workflows (#5007 by @Bibo-Joshi)
- Temporarily disable `actions/stale` due to a bug with `only-issue-types`. (#5008 by @Bibo-Joshi)
- Re-Enable `actions/stale` Workflow (#5009 by @Bibo-Joshi)
- Update dependency astral-sh/uv to v0.9.3 (#5011 by @renovatebot)
- Update github/codeql-action action to v4.30.9 (#5012 by @renovatebot)
- Update Ruff to v0.14.1 (#5013 by @renovatebot)
- Update dependency chango to ~=0.6.0 (#5014 by @renovatebot)
- Update dependency astral-sh/uv to v0.9.4 (#5015 by @renovatebot)
- Update astral-sh/setup-uv action to v7.1.1 (#5016 by @renovatebot)
- Update dependency astral-sh/uv to v0.9.5 (#5022 by @renovatebot)
- Update Pylint to v4.0.2 (#5023 by @renovatebot)
- Update Ruff to v0.14.2 (#5024 by @renovatebot)
- Update github/codeql-action action to v4.31.0 (#5025 by @renovatebot)
- Update sigstore/gh-action-sigstore-python action to v3.1.0 (#5026 by @renovatebot)
- Update GitHub Artifact Actions (major) (#5027 by @renovatebot)
- Update astral-sh/setup-uv action to v7.1.2 (#5029 by @renovatebot)
- Update dependency astral-sh/uv to v0.9.7 (#5032 by @renovatebot)
- Update github/codeql-action action to v4.31.2 (#5033 by @renovatebot)
- Update Ruff to v0.14.3 (#5034 by @renovatebot)
- Lock file maintenance (#5035 by @renovatebot)
- Update dependency astral-sh/uv to v0.9.8 (#5040 by @renovatebot)
- Update Ruff to v0.14.4 (#5041 by @renovatebot)
- Update dependency pytest to v9 (#5042 by @renovatebot)
- Reduce Frequence of Renovate Updates for Development Dependencies (#5044 by @Bibo-Joshi)
- Update Pylint to v4.0.3 (#5048 by @renovatebot)
- Update Ruff to v0.14.5 (#5049 by @renovatebot)

- Stabilize some unit tests (#5050 by @Bibo-Joshi)
- Update Ruff to v0.14.6 (#5055 by @renovatebot)
- Update Ruff to v0.14.7 (#5056 by @renovatebot)
- Update Pylint to v4.0.4 (#5057 by @renovatebot)
- Update actions/checkout action to v5.0.1 (#5058 by @renovatebot)
- Update astral-sh/setup-uv action to v7.1.4 (#5059 by @renovatebot)
- Update dependency astral-sh/uv to v0.9.13 (#5060 by @renovatebot)
- Update dependency pytest to v9.0.1 (#5061 by @renovatebot)
- Update github/codeql-action action to v4.31.5 (#5062 by @renovatebot)
- Update Ruff to v0.14.8 (#5067 by @renovatebot)
- Update Ruff to v0.14.9 (#5068 by @renovatebot)
- Update Ruff to v0.14.10 (#5070 by @renovatebot)
- Bump `pre-commit` Hooks to Latest VErsion (#5083 by @pre-commit-ci, @Bibo-Joshi)
- Update Ruff to v0.14.11 (#5100 by @renovatebot)
- Bump `chango` to 0.6.1 (#5101 by @Bibo-Joshi)
- Update Ruff to v0.14.13 (#5107 by @renovatebot)

6.6.5 22.5

2025-09-27

Breaking Changes

- Move param `ReplyParameters.checklist_task_id` to last position.

  Hint

  This change addresses a breaking change accidentally introduced in version 22.4 where the introduction of the new parameter was not done in a backward compatible way.

  Existing code using keyword arguments is unaffected. Only code using positional arguments (and based on version 22.4) may need updates.

  (#4972 by @aelkheir)

New Features

- Add convenience methods for `BusinessOpeningHours`:
  - `get_opening_hours_for_day`: returns the opening hours applicable for a specific date
  - `is_open`: indicates whether the business is open at the specified date and time.

  (#4861 by @Aweryc closes #4194)

Bug Fixes

- Fix Handling of Infinite Bootstrap Retries in `Application.run_*` and `Updater.start_*` (#4973 by @Bibo-Joshi closes #4966)

Other Changes

- Bump Version to v22.5 (#4979 by @Bibo-Joshi)

Documentation

- Documentation Improvemennts (#4974 by @Bibo-Joshi)

Internal Changes

- Lock file maintenance (#4938 by @renovatebot)
- Update astral-sh/setup-uv digest to b75a909 (#4943 by @renovatebot)
- Update codecov/codecov-action action to v5.5.1 (#4944 by @renovatebot)
- Update dependency astral-sh/uv to v0.8.17 (#4945 by @renovatebot)
- Update github/codeql-action action to v3.30.3 (#4946 by @renovatebot)
- Update Pylint to v3.3.8 (#4947 by @renovatebot)
- Update Chango to v0.5.0 (#4948 by @renovatebot)
- Update Mypy to v1.18.1 (#4949 by @renovatebot)
- Align pre-commit hook APScheduler to with `pyproject.toml` (#4950 by @renovatebot, @Bibo-Joshi)
- Align pre-commit hook cachetools to with `pyproject.toml` (#4951 by @renovatebot, @Bibo-Joshi)
- Update pypa/gh-action-pypi-publish action to v1.13.0 (#4952 by @renovatebot)
- Renovate: No README updates, label behaviour change, automerge lockfiles (#4953 by @harshil21)
- Lock file maintenance (#4954 by @renovatebot)
- Lock file maintenance (#4955 by @renovatebot)
- Update astral-sh/setup-uv digest to 208b0c0 (#4958 by @renovatebot)
- Update dependency astral-sh/uv to v0.8.19 (#4959 by @renovatebot)
- Update Mypy to v1.18.2 (#4960 by @renovatebot)
- Update astral-sh/setup-uv action to v6.7.0 (#4961 by @renovatebot)
- Update Ruff to v0.13.1 (#4962 by @renovatebot)
- Update actions/stale action to v10 (#4964 by @renovatebot)
- Properly Pin Dependency to `astral/setup-uv` in Copilot Setup Steps (#4965 by @Bibo-Joshi)
- Lock file maintenance (#4967 by @renovatebot)
- Tune Renovate Configuration (#4968 by @harshil21)
- Update dependency astral-sh/uv to v0.8.22 (#4975 by @renovatebot)
- Update github/codeql-action action to v3.30.5 (#4976 by @renovatebot)
- Update Ruff to v0.13.2 (#4977 by @renovatebot)
- Update dependency furo to v2025.9.25 (#4978 by @renovatebot)

6.6.6 22.4

2025-09-13

New Features

- Extend *telegram.Message.delete()* shortcut to support business message deletion (#4869 by @jainamoswal closes #4867)
- Add convenience properties for `firstname`, `lastname`, and `username` to `SharedUser` and `ChatShared` (#4881 by @aelkheir; #4713 by @david-shiko)
- Add `filters.FORUM` to filter messages from forum topic chats (#4906 by @harshil21)
- Full Support for Bot API 9.2 (#4911 by @aelkheir closes #4910; #4918 by @Poolitzer; #4917 by @Poolitzer; #4914 by @harshil21; #4916 by @harshil21; #4912 by @aelkheir; #4921 by @aelkheir; #4936 by @Bibo-Joshi; #4935 by @Bibo-Joshi; #4931 by @aelkheir)

Bug Fixes

- Adapt logic on getting the event loop in `Application.run_polling/webhook` to Python 3.14 (#4875 by @harshil21 closes #4874)
- Fix `ResourceWarning` when passing `pathlib.Path` objects to methods which accept file input (#4908 by @harshil21 closes #4907)

Dependencies

- Update cachetools requirement from <6.2.0,>=5.3.3 to >=5.3.3,<6.3.0 (#4923 by @dependabot)

Other Changes

- Set the default connection pool size for `HTTPXRequest` to 256 to allow more concurrent requests by default. Drop the `httpx` parameter `max_keepalive_connections`. This way, the `httpx` default of 20 is used, leading to a smaller number of idle connections in large connection pools.

  Hint

  If you manually build the `HTTPXRequest` objects, please be aware that these changes also applies to you. Kindly double check your settings. To specify custom limits, you can set them via the parameter `httpx_kwargs` of `HTTPXRequest`. See also the httpx documentation for more details on these settings."

  (#4882 by @Poolitzer)
- Bump Version to v22.4 (#4939 by @Bibo-Joshi)

Documentation

- Documentation Improvements (#4878 by @Bibo-Joshi; #4872 by @Ca5parAD)
- Bump `furo` from 2024.8.6 to 2025.7.19 (#4894 by @dependabot)

Internal Changes

- Address Failing Unit Test for `send_paid_media` (#4879 by @Bibo-Joshi)
- Improve Internal Logic for Network Retries (#4880 by @Bibo-Joshi closes #4871)
- Introduce utility module `_utils.usernames` refactoring convenience properties around Telegram Objects' `firstname`, `lastname`, and `username` (#4881 by @aelkheir; #4713 by @david-shiko)
- Add Copilot Instructions and Setup Steps (#4884 by @harshil21)
- Remove `black`, `isort`, `flake8`, and `pyupgrade` in favor of `ruff` (#4886 by @harshil21)
- Use Renovate to Keep Dependencies Up-To-Date (#4887 by @renovatebot)
- Add and use a `uv.lock` lockfile when setting up the development environment using `uv`. (#4890 by @harshil21 closes #4796)

- Bump `pytest` from 8.4.0 to 8.4.1 (#4892 by @dependabot)
- Bump `pytest-xdist` from 3.6.1 to 3.8.0 (#4893 by @dependabot)
- Bump `astral-sh/setup-uv` from 6.3.1 to 6.4.3 (#4895 by @dependabot)
- Bump `github/codeql-action` from 3.29.2 to 3.29.5 (#4896 by @dependabot)
- Don't update `uv.lock` in copilot runtime environment (#4915 by @harshil21)
- Bump actions/checkout from 4.2.2 to 5.0.0 (#4925 by @dependabot)
- Bump codecov/codecov-action from 5.4.3 to 5.5.0 (#4926 by @dependabot)
- Bump actions/download-artifact from 4.3.0 to 5.0.0 (#4927 by @dependabot)
- Bump astral-sh/setup-uv from 6.4.3 to 6.6.1 (#4928 by @dependabot)
- Bump github/codeql-action from 3.29.7 to 3.30.0 (#4929 by @dependabot)
- Bump pytest from 8.4.1 to 8.4.2 (#4933 by @dependabot)
- Use Tagged Release of *pydantic* in Development Dependencies (#4934 by @harshil21 closes #4932)

6.6.7 22.3

2025-07-20

Highlights

- Full Support for Bot API 9.1 (#4847 by @Bibo-Joshi closes #4845; #4848 by @Bibo-Joshi; #4849 by @harshil21; #4851 by @harshil21; #4857 by @aelkheir)

Breaking Changes

- Remove Functionality Deprecated in API 9.0
  - Remove deprecated argument and attribute `BusinessConnection.can_reply`.
  - Remove deprecated argument and attribute `ChatFullInfo.can_send_gift`
  - Remove deprecated class `constants.StarTransactions`. Please instead use *telegram.constants.Nanostar.VALUE*.
  - Remove deprecated attributes `constants.StarTransactionsLimit.NANOSTAR_MIN_AMOUNT` and `constants.StarTransactionsLimit.NANOSTAR_MAX_AMOUNT`. Please instead use *telegram.constants.NanostarLimit.MIN_AMOUNT* and *telegram.constants.NanostarLimit.MAX_AMOUNT*.

  (#4852 by @aelkheir)

New Features

- New filters based on Bot API 9.1:
  - `filters.StatusUpdate.DIRECT_MESSAGE_PRICE_CHANGED` for `Message.direct_message_price_changed`
  - `filters.StatusUpdate.CHECKLIST_TASKS_ADDED` for `Message.checklist_tasks_added`
  - `filters.StatusUpdate.CHECKLIST_TASKS_DONE` for `Message.checklist_tasks_done`
  - `filters.CHECKLIST` for `Message.checklist`

  (#4847 by @Bibo-Joshi closes #4845; #4848 by @Bibo-Joshi; #4849 by @harshil21; #4851 by @harshil21; #4857 by @aelkheir)

Other Changes

- Make Gender Input Case-Insensitive in `conversationbot.py` (#4855 by @fengxiaohu closes #4846)
- Bump Version to v22.3 (#4870 by @Bibo-Joshi)

Documentation

- Documentation Improvements. Among others, fix links to source code. (#4839 by @aelkheir closes #4838)

Internal Changes

- Update API Token for Local Testing Bot (#4837 by @Bibo-Joshi)
- Bump stefanzweifel/git-auto-commit-action from 5.2.0 to 6.0.1 (#4840 by @dependabot)
- Bump github/codeql-action from 3.28.18 to 3.29.2 (#4841 by @dependabot)
- Bump astral-sh/setup-uv from 5.4.1 to 6.3.1 (#4842 by @dependabot)
- Bump sigstore/gh-action-sigstore-python from 3.0.0 to 3.0.1 (#4843 by @dependabot)
- Bump *pre-commit* Hooks to Latest Versions (#4858 by @pre-commit-ci)

6.6.8 22.2

2025-06-29

Deprecations

- In this release, we're migrating attributes of Telegram objects that represent durations/time periods from having `int` type to Python's native `datetime.timedelta`. This change is opt-in for now to allow for a smooth transition phase. It will become opt-out in future releases.

  Set `PTB_TIMEDELTA=true` or `PTB_TIMEDELTA=1` as an environment variable to make these attributes return `datetime.timedelta` objects instead of integers. Support for `int` values is deprecated and will be removed in a future major version.

  Affected Attributes:

  - *telegram.ChatFullInfo.slow_mode_delay* and *telegram.ChatFullInfo.message_auto_delete_time*
  - *telegram.Animation.duration*
  - *telegram.Audio.duration*
  - *telegram.Video.duration* and *telegram.Video.start_timestamp*
  - *telegram.VideoNote.duration*
  - *telegram.Voice.duration*
  - *telegram.PaidMediaPreview.duration*
  - *telegram.VideoChatEnded.duration*
  - *telegram.InputMediaVideo.duration*
  - *telegram.InputMediaAnimation.duration*
  - *telegram.InputMediaAudio.duration*
  - *telegram.InputPaidMediaVideo.duration*
  - *telegram.InlineQueryResultGif.gif_duration*
  - *telegram.InlineQueryResultMpeg4Gif.mpeg4_duration*
  - *telegram.InlineQueryResultVideo.video_duration*
  - *telegram.InlineQueryResultAudio.audio_duration*

- *telegram.InlineQueryResultVoice.voice_duration*
- *telegram.InlineQueryResultLocation.live_period*
- *telegram.Poll.open_period*
- *telegram.Location.live_period*
- *telegram.MessageAutoDeleteTimerChanged.message_auto_delete_time*
- *telegram.ChatInviteLink.subscription_period*
- *telegram.InputLocationMessageContent.live_period*
- *telegram.error.RetryAfter.retry_after*

(#4750 by @aelkheir closes #4575)

New Features

- Use *timedelta* to represent time periods in class arguments and attributes (#4750 by @aelkheir closes #4575)

Bug Fixes

- Fixed a bug where calling `Application.remove/add_handler` during update handling can cause a `RuntimeError` in `Application.process_update`.

  Hint

  Calling `Application.add/remove_handler` now has no influence on calls to `process_update` that are

  already in progress. The same holds for `Application.add/remove_error_handler` and `Application.process_error`, respectively.

  Warning

  This behavior should currently be considered an implementation detail and not as guaranteed behavior.

  (#4802 by @Bibo-Joshi closes #4803)

- Allow for pattern matching empty inline queries (#4817 by @locobott)
- Correctly parse parameter `allow_sending_without_reply` in `Message.reply_*` when used in combination with `do_quote=True`.

  Hint

  Using `dict` valued input for `do_quote` along with passing `allow_sending_without_reply` is not supported and will raise an error.

  (#4818 by @Bibo-Joshi closes #4807)

Dependencies

- Implement PEP 735 Dependency Groups for Development Dependencies (#4800 by @harshil21 closes #4795)
- Update cachetools requirement from <5.6.0,>=5.3.3 to >=5.3.3,<6.1.0 (#4801 by @dependabot)
- Bump `httpx` from ~=0.27 to >=0.27,<0.29 (#4820 by @Bibo-Joshi closes #4819)

- Update `cachetools` requirement from <6.1.0,>=5.3.3 to >=5.3.3,<6.2.0 (#4830 by @dependabot)

Other Changes

- Improve Informativeness of Network Errors Raised by `BaseRequest.post/retrieve` (#4822 by @Bibo-Joshi)
- Add Python 3.14 Beta To Test Matrix. *Python 3.14 is not officially supported by PTB yet!* (#4825 by @harshil21)
- Bump Version to v22.2 (#4834 by @Bibo-Joshi)

Documentation

- Documentation Improvements. Among other things
  - mention alternative package managers in README and contribution guide
  - remove `furo-sphinx-search`

  (#4810 by @Bibo-Joshi; #4824 by @Aweryc closes #4823; #4826 by @harshil21)

Internal Changes

- Modify *test_official* to handle time periods as timedelta automatically. (#4750 by @aelkheir closes #4575)
- Fix Bug in Automated Channel Announcement (#4792 by @Bibo-Joshi)
- Fix a Failing Test Case (#4793 by @Bibo-Joshi)
- Rework Repository to *src* Layout (#4798 by @Bibo-Joshi closes #4797)
- Bump github/codeql-action from 3.28.16 to 3.28.18 (#4811 by @dependabot)
- Bump actions/setup-python from 5.5.0 to 5.6.0 (#4812 by @dependabot)
- Bump dependabot/fetch-metadata from 2.3.0 to 2.4.0 (#4813 by @dependabot)
- Bump codecov/codecov-action from 5.4.2 to 5.4.3 (#4814 by @dependabot)
- Bump codecov/test-results-action from 1.1.0 to 1.1.1 (#4815 by @dependabot)
- Fix Typo in *TelegramObject._get_attrs* (#4816 by @harshil21)

6.6.9 22.1

2025-05-15

Breaking Changes

- Drop backward compatibility for `user_id` in `send_gift` by updating the order of parameters. Please adapt your code accordingly or use keyword arguments. (#4692 by @Bibo-Joshi)

Deprecations

- This release comes with several deprecations, in line with our *stability policy*.

  This includes the following:

  - Deprecated `telegram.constants.StarTransactionsLimit.NANOSTAR_MIN_AMOUNT` and `telegram.constants.StarTransactionsLimit.NANOSTAR_MAX_AMOUNT`. These members will be replaced by `telegram.constants.NanostarLimit.MIN_AMOUNT` and `telegram.constants.NanostarLimit.MAX_AMOUNT`.
  - Deprecated the class `telegram.constants.StarTransactions`. Its only member `telegram.constants.StarTransactions.NANOSTAR_VALUE` will be replaced by `telegram.constants.Nanostar.VALUE`.

- Bot API 9.0 deprecated `BusinessConnection.can_reply` in favor of `BusinessConnection.rights`
- Bot API 9.0 deprecated `ChatFullInfo.can_send_gift` in favor of `ChatFullInfo.accepted_gift_types`.
- Bot API 9.0 introduced these new required fields to existing classes:
  * `TransactionPartnerUser.transaction_type`
  * `ChatFullInfo.accepted_gift_types`

  Passing these values as positional arguments is deprecated. We encourage you to use keyword arguments instead, as the the signature will be updated in a future release.

These deprecations are backward compatible, but we strongly recommend to update your code to use the new members.

(#4756 by @Bibo-Joshi closes #4754; #4757 by @Bibo-Joshi; #4759 by @Bibo-Joshi; #4763 by @aelkheir; #4766 by @Bibo-Joshi; #4769 by @aelkheir; #4773 by @aelkheir; #4781 by @aelkheir; #4782 by @Bibo-Joshi)

New Features

- Full Support for Bot API 9.0 (#4756 by @Bibo-Joshi closes #4754; #4757 by @Bibo-Joshi; #4759 by @Bibo-Joshi; #4763 by @aelkheir; #4766 by @Bibo-Joshi; #4769 by @aelkheir; #4773 by @aelkheir; #4781 by @aelkheir; #4782 by @Bibo-Joshi)

Bug Fixes

- Ensure execution of `Bot.shutdown()` even if `Bot.get_me()` fails in `Bot.initialize()` (#4733 by @Poolitzer)
- Fix Handling of `Defaults` for `InputPaidMedia` (#4761 by @ngrogolev closes #4753)

Other Changes

- Bump Version to v22.1 (#4791 by @Bibo-Joshi)

Documentation

- Documentation Improvements. Among others, add missing `Returns` field in `User.get_profile_photos` (#4730 by @Bibo-Joshi; #4740 by @aelkheir)
- Update `AUTHORS.rst`, Adding @aelkheir to Active Development Team (#4747 by @Bibo-Joshi)
- Clarify Documentation and Type Hints of `InputMedia` and `InputPaidMedia`. Note that the `media` parameter accepts only objects of type `str` and `InputFile`. The respective subclasses of `Input(Paid)Media` each accept a broader range of input type for the `media` parameter. (#4762 by @Bibo-Joshi)

Internal Changes

- Bump codecov/test-results-action from 1.0.2 to 1.1.0 (#4741 by @dependabot)
- Bump actions/setup-python from 5.4.0 to 5.5.0 (#4742 by @dependabot)
- Bump github/codeql-action from 3.28.10 to 3.28.13 (#4743 by @dependabot)
- Bump astral-sh/setup-uv from 5.3.1 to 5.4.1 (#4744 by @dependabot)
- Bump actions/download-artifact from 4.1.8 to 4.2.1 (#4745 by @dependabot)
- Reenable `test_official` Blocked by Debug Remnant (#4746 by @aelkheir)
- Bump *pre-commit* Hooks to Latest Versions (#4748 by @pre-commit-ci)
- Fine-tune `chango` and release workflows (#4758 by @Bibo-Joshi closes #4720)
- Bump codecov/codecov-action from 5.1.2 to 5.4.2 (#4775 by @dependabot)

- Bump actions/upload-artifact from 4.5.0 to 4.6.2 (#4776 by @dependabot)
- Bump stefanzweifel/git-auto-commit-action from 5.1.0 to 5.2.0 (#4777 by @dependabot)
- Bump github/codeql-action from 3.28.13 to 3.28.16 (#4778 by @dependabot)
- Bump actions/download-artifact from 4.2.1 to 4.3.0 (#4779 by @dependabot)

6.6.10 22.0

2025-03-15

Breaking Changes

- This release removes all functionality that was deprecated in v20.x. This is in line with our *stability policy*.

  This includes the following changes:

  - Removed `filters.CHAT` (all messages have an associated chat) and `filters.StatusUpdate.USER_SHARED` (use `filters.StatusUpdate.USERS_SHARED` instead).
  - Removed `Defaults.disable_web_page_preview` and `Defaults.quote`. Use `Defaults.link_preview_options` and `Defaults.do_quote` instead.
  - Removed `ApplicationBuilder.(get_updates_)proxy_url` and `HTTPXRequest.proxy_url`. Use `ApplicationBuilder.(get_updates_)proxy` and `HTTPXRequest.proxy` instead.
  - Removed the `*_timeout` arguments of `Application.run_polling` and `Updater.start_webhook`. Instead, specify the values via `ApplicationBuilder.get_updates_*_timeout`.
  - Removed `constants.InlineQueryLimit.MIN_SWITCH_PM_TEXT_LENGTH`. Use `constants.InlineQueryResultsButtonLimit.MAX_START_PARAMETER_LENGTH` instead.
  - Removed the argument `quote` of `Message.reply_*`. Use `do_quote` instead.
  - Removed the superfluous `EncryptedPassportElement.credentials` without replacement.
  - Changed attribute value of `PassportFile.file_date` from `int` to `datetime.datetime`. Make sure to adjust your code accordingly.
  - Changed the attribute value of `PassportElementErrors.file_hashes` from `list` to `tuple`. Make sure to adjust your code accordingly.
  - Make `BaseRequest.read_timeout` an abstract property. If you subclass `BaseRequest`, you need to implement this property.
  - The default value for `write_timeout` now defaults to `DEFAULT_NONE` also for bot methods that send media. Previously, it was `20`. If you subclass `BaseRequest`, make sure to use your desired write timeout if `RequestData.multipart_data` is set.

  (#4671 by @Bibo-Joshi closes #4659)

Documentation

- Add chango As Changelog Management Tool (#4672 by @Bibo-Joshi closes #4321)

Internal Changes

- Bump github/codeql-action from 3.28.8 to 3.28.10 (#4697 by @dependabot)
- Bump srvaroa/labeler from 1.12.0 to 1.13.0 (#4698 by @dependabot)
- Bump astral-sh/setup-uv from 5.2.2 to 5.3.1 (#4699 by @dependabot)
- Bump Bibo-Joshi/chango from 0.3.1 to 0.3.2 (#4700 by @dependabot)
- Bump pypa/gh-action-pypi-publish from 1.12.3 to 1.12.4 (#4701 by @dependabot)
- Bump pytest from 8.3.4 to 8.3.5 (#4709 by @dependabot)

- Bump sphinx from 8.1.3 to 8.2.3 (#4710 by @dependabot)
- Bump Bibo-Joshi/chango from 0.3.2 to 0.4.0 (#4712 by @Bibo-Joshi)
- Bump Version to v22.0 (#4719 by @Bibo-Joshi)

6.6.11 Version 21.11.1

Released 2025-03-01

This is the technical changelog for version 21.11. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Documentation Improvements

- Fix ReadTheDocs Build (#4695)

6.6.12 Version 21.11

Released 2025-03-01

This is the technical changelog for version 21.11. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes and New Features

- Full Support for Bot API 8.3 (#4676 closes #4677, #4682 by aelkheir, #4690 by aelkheir, #4691 by aelkheir)
- Make `provider_token` Argument Optional (#4689)
- Remove Deprecated `InlineQueryResultArticle.hide_url` (#4640 closes #4638)
- Accept `datetime.timedelta` Input in `Bot` Method Parameters (#4651)
- Extend Customization Support for `Bot.base_(file_)url` (#4632 closes #3355)
- Support `allow_paid_broadcast` in `AIORateLimiter` (#4627 closes #4578)
- Add `BaseUpdateProcessor.current_concurrent_updates` (#4626 closes #3984)

Minor Changes and Bug Fixes

- Add Bootstrapping Logic to `Application.run_*` (#4673 closes #4657)
- Fix a Bug in `edit_user_star_subscription` (#4681 by vavasik800)
- Simplify Handling of Empty Data in `TelegramObject.de_json` and Friends (#4617 closes #4614)

Documentation Improvements

- Documentation Improvements (#4641)
- Overhaul Admonition Insertion in Documentation (#4462 closes #4414)

Internal Changes

- Stabilize Linkcheck Test (#4693)
- Bump `pre-commit` Hooks to Latest Versions (#4643)
- Refactor Tests for `TelegramObject` Classes with Subclasses (#4654 closes #4652)
- Use Fine Grained Permissions for GitHub Actions Workflows (#4668)

Dependency Updates

- Bump `actions/setup-python` from 5.3.0 to 5.4.0 (#4665)
- Bump `dependabot/fetch-metadata` from 2.2.0 to 2.3.0 (#4666)
- Bump `actions/stale` from 9.0.0 to 9.1.0 (#4667)
- Bump `astral-sh/setup-uv` from 5.1.0 to 5.2.2 (#4664)
- Bump `codecov/test-results-action` from 1.0.1 to 1.0.2 (#4663)

6.6.13 Version 21.10

Released 2025-01-03

This is the technical changelog for version 21.10. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Full Support for Bot API 8.2 (#4633)
- Bump `apscheduler` & Deprecate `pytz` Support (#4582)

New Features

- Add Parameter `pattern` to `JobQueue.jobs()` (#4613 closes #4544)
- Allow Input of Type `Sticker` for Several Methods (#4616 closes #4580)

Bug Fixes

- Ensure Forward Compatibility of `Gift` and `Gifts` (#4634 closes #4637)

Documentation Improvements & Internal Changes

- Use Custom Labels for `dependabot` PRs (#4621)
- Remove Redundant `pylint` Suppressions (#4628)
- Update Copyright to 2025 (#4631)
- Refactor Module Structure and Tests for Star Payments Classes (#4615 closes #4593)
- Unify `datetime` Imports (#4605 by cuevasrja closes #4577)
- Add Static Security Analysis of GitHub Actions Workflows (#4606)

Dependency Updates

- Bump `astral-sh/setup-uv` from 4.2.0 to 5.1.0 (#4625)
- Bump `codecov/codecov-action` from 5.1.1 to 5.1.2 (#4622)
- Bump `actions/upload-artifact` from 4.4.3 to 4.5.0 (#4623)
- Bump `github/codeql-action` from 3.27.9 to 3.28.0 (#4624)

6.6.14 Version 21.9

Released 2024-12-07

This is the technical changelog for version 21.9. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Full Support for Bot API 8.1 (#4594 closes #4592)

Minor Changes

- Use `MessageLimit.DEEP_LINK_LENGTH` in `helpers.create_deep_linked_url` (#4597 by nemacysts)
- Allow `Sequence` Input for `allowed_updates` in `Application` and `Updater` Methods (#4589 by nemacysts)

Dependency Updates

- Update `aiolimiter` requirement from ~=1.1.0 to >=1.1,<1.3 (#4595)
- Bump `pytest` from 8.3.3 to 8.3.4 (#4596)
- Bump `codecov/codecov-action` from 4 to 5 (#4585)
- Bump `pylint` to v3.3.2 to Improve Python 3.13 Support (#4590 by nemacysts)
- Bump `srvaroa/labeler` from 1.11.1 to 1.12.0 (#4586)

6.6.15 Version 21.8

Released 2024-12-01

This is the technical changelog for version 21.8. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Full Support for Bot API 8.0 (#4568, #4566 closes #4567, #4572, #4571, #4570, #4576, #4574)

Documentation Improvements

- Documentation Improvements (#4565 by Snehashish06, #4573)

6.6.16 Version 21.7

Released 2024-11-04

This is the technical changelog for version 21.7. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Full Support for Bot API 7.11 (#4546 closes #4543)
- Add `Message.reply_paid_media` (#4551)
- Drop Support for Python 3.8 (#4398 by elpekenin)

Minor Changes

- Allow `Sequence` in `Application.add_handlers` (#4531 by roast-lord closes #4530)
- Improve Exception Handling in `File.download_*` (#4542)
- Use Stable Python 3.13 Release in Test Suite (#4535)

Documentation Improvements

- Documentation Improvements (#4536 by Ecode2, #4556)
- Fix Linkcheck Workflow (#4545)
- Use `sphinx-build-compatibility` to Keep Sphinx Compatibility (#4492)

Internal Changes

- Improve Test Instability Caused by `Message` Fixtures (#4507)
- Stabilize Some Flaky Tests (#4500)
- Reduce Creation of HTTP Clients in Tests (#4493)
- Update `pytest-xdist` Usage (#4491)
- Fix Failing Tests by Making Them Independent (#4494)
- Introduce Codecov's Test Analysis (#4487)
- Maintenance Work on `Bot` Tests (#4489)
- Introduce `conftest.py` for File Related Tests (#4488)
- Update Issue Templates to Use Issue Types (#4553)
- Update Automation to Label Changes (#4552)

Dependency Updates

- Bump `srvaroa/labeler` from 1.11.0 to 1.11.1 (#4549)
- Bump `sphinx` from 8.0.2 to 8.1.3 (#4532)
- Bump `sphinxcontrib-mermaid` from 0.9.2 to 1.0.0 (#4529)
- Bump `srvaroa/labeler` from 1.10.1 to 1.11.0 (#4509)
- Bump `Bibo-Joshi/pyright-type-completeness` from 1.0.0 to 1.0.1 (#4510)

6.6.17 Version 21.6

Released 2024-09-19

This is the technical changelog for version 21.6. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

New Features

- Full Support for Bot API 7.10 (#4461 closes #4459, #4460, #4463 by aelkheir, #4464)
- Add Parameter `httpx_kwargs` to `HTTPXRequest` (#4451 closes #4424)

Minor Changes

- Improve Type Completeness (#4466)

Internal Changes

- Update Python 3.13 Test Suite to RC2 (#4471)
- Enforce the `offline_bot` Fixture in `Test*WithoutRequest` (#4465)
- Make Tests for `telegram.ext` Independent of Networking (#4454)
- Rename Testing Base Classes (#4453)

Dependency Updates

- Bump `pytest` from 8.3.2 to 8.3.3 (#4475)

6.6.18 Version 21.5

Released 2024-09-01

This is the technical changelog for version 21.5. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Full Support for Bot API 7.9 (#4429)
- Full Support for Bot API 7.8 (#4408)

New Features

- Add `MessageEntity.shift_entities` and `MessageEntity.concatenate` (#4376 closes #4372)
- Add Parameter `game_pattern` to `CallbackQueryHandler` (#4353 by jainamoswal closes #4269)
- Add Parameter `read_file_handle` to `InputFile` (#4388 closes #4339)

Documentation Improvements

- Bugfix for "Available In" Admonitions (#4413)
- Documentation Improvements (#4400 closes #4446, #4448 by Palaptin)
- Document Return Types of `RequestData` Members (#4396)
- Add Introductory Paragraphs to Telegram Types Subsections (#4389 by mohdyusuf2312 closes #4380)
- Start Adapting to RTD Addons (#4386)

Minor and Internal Changes

- Remove Surplus Logging from `Updater` Network Loop (#4432 by MartinHjelmare)
- Add Internal Constants for Encodings (#4378 by elpekenin)
- Improve PyPI Automation (#4375 closes #4373)
- Update Test Suite to New Test Channel Setup (#4435)
- Improve Fixture Usage in `test_message.py` (#4431 by Palaptin)
- Update Python 3.13 Test Suite to RC1 (#4415)
- Bump `ruff` and Add New Rules (#4416)

Dependency Updates

- Update `cachetools` requirement from <5.5.0,>=5.3.3 to >=5.3.3,<5.6.0 (#4437)
- Bump `sphinx` from 7.4.7 to 8.0.2 and `furo` from 2024.7.18 to 2024.8.6 (#4412)
- Bump `test-summary/action` from 2.3 to 2.4 (#4410)
- Bump `pytest` from 8.2.2 to 8.3.2 (#4403)
- Bump `dependabot/fetch-metadata` from 2.1.0 to 2.2.0 (#4411)
- Update `cachetools` requirement from ~=5.3.3 to >=5.3.3,<5.5.0 (#4390)
- Bump `sphinx` from 7.3.7 to 7.4.7 (#4395)
- Bump `furo` from 2024.5.6 to 2024.7.18 (#4392)

6.6.19 Version 21.4

Released 2024-07-12

This is the technical changelog for version 21.4. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Full Support for Bot API 7.5 (#4328, #4316, #4315, #4312 closes #4310, #4311)
- Full Support for Bot API 7.6 (#4333 closes #4331, #4344, #4341, #4334, #4335, #4351, #4342, #4348)
- Full Support for Bot API 7.7 (#4356 closes #4355)
- Drop `python-telegram-bot-raw` And Switch to `pyproject.toml` Based Packaging (#4288 closes #4129 and #4296)
- Deprecate Inclusion of `successful_payment` in `Message.effective_attachment` (#4365 closes #4350)

New Features

- Add Support for Python 3.13 Beta (#4253)
- Add `filters.PAID_MEDIA` (#4357)
- Log Received Data on Deserialization Errors (#4304)
- Add `MessageEntity.adjust_message_entities_to_utf_16` Utility Function (#4323 by Antares0982 closes #4319)
- Make Argument `bot` of `TelegramObject.de_json` Optional (#4320)

Documentation Improvements

- Documentation Improvements (#4303 closes #4301)
- Restructure Readme (#4362)
- Fix Link-Check Workflow (#4332)

Internal Changes

- Automate PyPI Releases (#4364 closes #4318)
- Add `mise-en-place` to `.gitignore` (#4300)
- Use a Composite Action for Testing Type Completeness (#4367)
- Stabilize Some Concurrency Usages in Test Suite (#4360)
- Add a Test Case for `MenuButton` (#4363)
- Extend `SuccessfulPayment` Test (#4349)
- Small Fixes for `test_stars.py` (#4347)
- Use Python 3.13 Beta 3 in Test Suite (#4336)

Dependency Updates

- Bump `ruff` and Add New Rules (#4329)
- Bump `pre-commit` Hooks to Latest Versions (#4337)
- Add Lower Bound for `flaky` Dependency (#4322 by Palaptin)
- Bump `pytest` from 8.2.1 to 8.2.2 (#4294)

6.6.20 Version 21.3

Released 2024-06-07

This is the technical changelog for version 21.3. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Full Support for Bot API 7.4 (#4286, #4276 closes #4275, #4285, #4283, #4280, #4278, #4279)
- Deprecate `python-telegram-bot-raw` (#4270)
- Remove Functionality Deprecated in Bot API 7.3 (#4266 closes #4244)

New Features

- Add Parameter `chat_id` to `ChatMemberHandler` (#4290 by uniquetrij closes #4287)

Documentation Improvements

- Documentation Improvements (#4264 closes #4240)

Internal Changes

- Add `setuptools` to `requirements-dev.txt` (#4282)
- Update Settings for pre-commit.ci (#4265)

Dependency Updates

- Bump `pytest` from 8.2.0 to 8.2.1 (#4272)

6.6.21 Version 21.2

Released 2024-05-20

This is the technical changelog for version 21.2. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Full Support for Bot API 7.3 (#4246, #4260, #4243, #4248, #4242 closes #4236, #4247 by aelkheir)
- Remove Functionality Deprecated by Bot API 7.2 (#4245)

New Features

- Add Version to `PTBDeprecationWarning` (#4262 closes #4261)
- Handle Exceptions in building `CallbackContext` (#4222)

Bug Fixes

- Call `Application.post_stop` Only if `Application.stop` was called (#4211 closes #4210)
- Handle `SystemExit` raised in Handlers (#4157 closes #4155 and #4156)
- Make `Birthdate.to_date` Return a `datetime.date` Object (#4251)

Documentation Improvements

- Documentation Improvements (#4217)

Internal Changes

- Add New Rules to `ruff` Config (#4250)
- Adapt Test Suite to Changes in Error Messages (#4238)

Dependency Updates

- Bump `furo` from 2024.4.27 to 2024.5.6 (#4252)
- `pre-commit` autoupdate (#4239)
- Bump `pytest` from 8.1.1 to 8.2.0 (#4231)
- Bump `dependabot/fetch-metadata` from 2.0.0 to 2.1.0 (#4228)
- Bump `pytest-asyncio` from 0.21.1 to 0.21.2 (#4232)
- Bump `pytest-xdist` from 3.6.0 to 3.6.1 (#4233)
- Bump `furo` from 2024.1.29 to 2024.4.27 (#4230)
- Bump `srvaroa/labeler` from 1.10.0 to 1.10.1 (#4227)
- Bump `pytest` from 7.4.4 to 8.1.1 (#4218)
- Bump `sphinx` from 7.2.6 to 7.3.7 (#4215)
- Bump `pytest-xdist` from 3.5.0 to 3.6.0 (#4215)

6.6.22 Version 21.1.1

Released 2024-04-15

This is the technical changelog for version 21.1.1. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Bug Fixes

- Fix Bug With Parameter `message_thread_id` of `Message.reply_*` (#4207 closes #4205)

Minor Changes

- Remove Deprecation Warning in `JobQueue.run_daily` (#4206 by @Konano)
- Fix Annotation of `EncryptedCredentials.decrypted_secret` (#4199 by @marinelay closes #4198)

6.6.23 Version 21.1

Released 2024-04-12

This is the technical changelog for version 21.1. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- API 7.2 (#4180 closes #4179 and #4181, #4181)
- Make `ChatAdministratorRights/ChatMemberAdministrator.can_*_stories` Required (API 7.1) (#4192)

Minor Changes

- Refactor Debug logging in `Bot` to Improve Type Hinting (#4151 closes #4010)

New Features

- Make `Message.reply_*` Reply in the Same Topic by Default (#4170 by @aelkheir closes #4139)
- Accept Socket Objects for Webhooks (#4161 closes #4078)
- Add `Update.effective_sender` (#4168 by @aelkheir closes #4085)

Documentation Improvements

- Documentation Improvements (#4171, #4158 by @teslaedison)

Internal Changes

- Temporarily Mark Tests with `get_sticker_set` as XFAIL due to API 7.2 Update (#4190)

Dependency Updates

- `pre-commit` autoupdate (#4184)
- Bump `dependabot/fetch-metadata` from 1.6.0 to 2.0.0 (#4185)

6.6.24 Version 21.0.1

Released 2024-03-06

This is the technical changelog for version 21.0.1. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Bug Fixes

- Remove `docs` from Package (#4150)

6.6.25 Version 21.0

Released 2024-03-06

This is the technical changelog for version 21.0. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Remove Functionality Deprecated in API 7.0 (#4114 closes #4099)
- API 7.1 (#4118)

New Features

- Add Parameter `media_write_timeout` to `HTTPXRequest` and Method `ApplicationBuilder.media_write_timeout` (#4120 closes #3864)
- Handle Properties in `TelegramObject.__setstate__` (#4134 closes #4111)

Bug Fixes

- Add Missing Slot to `Updater` (#4130 closes #4127)

Documentation Improvements

- Improve HTML Download of Documentation (#4146 closes #4050)
- Documentation Improvements (#4109, #4116)
- Update Copyright to 2024 (#4121 by @aelkheir closes #4041)

Internal Changes

- Apply `pre-commit` Checks More Widely (#4135)
- Refactor and Overhaul `test_official` (#4087 closes #3874)
- Run Unit Tests in PRs on Requirements Changes (#4144)
- Make `Updater.stop` Independent of `CancelledError` (#4126)

Dependency Updates

- Relax Upper Bound for `httpx` Dependency (#4148)
- Bump `test-summary/action` from 2.2 to 2.3 (#4142)
- Update `cachetools` requirement from ~=5.3.2 to ~=5.3.3 (#4141)
- Update `httpx` requirement from ~=0.26.0 to ~=0.27.0 (#4131)

6.6.26 Version 20.8

Released 2024-02-08

This is the technical changelog for version 20.8. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- API 7.0 (#4034 closes #4033, #4038 by @aelkheir)

Minor Changes

- Fix Type Hint for `filters` Parameter of `MessageHandler` (#4039 by @Palaptin)
- Deprecate `filters.CHAT` (#4083 closes #4062)
- Improve Error Handling in Built-In Webhook Handler (#3987 closes #3979)

New Features

- Add Parameter `pattern` to `PreCheckoutQueryHandler` and `filters.SuccessfulPayment` (#4005 by @aelkheir closes #3752)
- Add Missing Conversions of `type` to Corresponding Enum from `telegram.constants` (#4067)
- Add Support for Unix Sockets to `Updater.start_webhook` (#3986 closes #3978)
- Add `Bot.do_api_request` (#4084 closes #4053)
- Add `AsyncContextManager` as Parent Class to `BaseUpdateProcessor` (#4001)

Documentation Improvements

- Documentation Improvements (#3919)
- Add Docstring to Dunder Methods (#3929 closes #3926)
- Documentation Improvements (#4002, #4079 by @kenjitagawa, #4104 by @xTudoS)

Internal Changes

- Drop Usage of DeepSource (#4100)
- Improve Type Completeness & Corresponding Workflow (#4035)
- Bump `ruff` and Remove `sort-all` (#4075)
- Move Handler Files to `_handlers` Subdirectory (#4064 by @lucasmolinari closes #4060)

- Introduce `sort-all` Hook for `pre-commit` (#4052)
- Use Recommended `pre-commit` Mirror for `black` (#4051)
- Remove Unused `DEFAULT_20` (#3997)
- Migrate From `setup.cfg` to `pyproject.toml` Where Possible (#4088)

Dependency Updates

- Bump `black` and `ruff` (#4089)
- Bump `srvaroa/labeler` from 1.8.0 to 1.10.0 (#4048)
- Update `tornado` requirement from ~=6.3.3 to ~=6.4 (#3992)
- Bump `actions/stale` from 8 to 9 (#4046)
- Bump `actions/setup-python` from 4 to 5 (#4047)
- `pre-commit` autoupdate (#4101)
- Bump `actions/upload-artifact` from 3 to 4 (#4045)
- `pre-commit` autoupdate (#3996)
- Bump `furo` from 2023.9.10 to 2024.1.29 (#4094)
- `pre-commit` autoupdate (#4043)
- Bump `codecov/codecov-action` from 3 to 4 (#4091)
- Bump `EndBug/add-and-commit` from 9.1.3 to 9.1.4 (#4090)
- Update `httpx` requirement from ~=0.25.2 to ~=0.26.0 (#4024)
- Bump `pytest` from 7.4.3 to 7.4.4 (#4056)
- Bump `srvaroa/labeler` from 1.7.0 to 1.8.0 (#3993)
- Bump `test-summary/action` from 2.1 to 2.2 (#4044)
- Bump `dessant/lock-threads` from 4.0.1 to 5.0.1 (#3994)

6.6.27 Version 20.7

Released 2023-11-27

This is the technical changelog for version 20.7. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

New Features

- Add `JobQueue.scheduler_configuration` and Corresponding Warnings (#3913 closes #3837)
- Add Parameter `socket_options` to `HTTPXRequest` (#3935 closes #2965)
- Add `ApplicationBuilder.(get_updates_)socket_options` (#3943)
- Improve `write_timeout` Handling for Media Methods (#3952)
- Add `filters.Mention` (#3941 closes #3799)
- Rename `proxy_url` to `proxy` and Allow `httpx.{Proxy, URL}` as Input (#3939 closes #3844)

Bug Fixes & Changes

- Adjust `read_timeout` Behavior for `Bot.get_updates` (#3963 closes #3893)
- Improve `BaseHandler.__repr__` for Callbacks without `__qualname__` (#3934)
- Fix Persistency Issue with Ended Non-Blocking Conversations (#3962)

- Improve Type Hinting for Arguments with Default Values in `Bot` (#3942)

Documentation Improvements

- Add Documentation for `__aenter__` and `__aexit__` Methods (#3907 closes #3886)
- Improve Insertion of Kwargs into `Bot` Methods (#3965)

Internal Changes

- Adjust Tests to New Error Messages (#3970)

Dependency Updates

- Bump `pytest-xdist` from 3.3.1 to 3.4.0 (#3975)
- `pre-commit` autoupdate (#3967)
- Update `httpx` requirement from ~=0.25.1 to ~=0.25.2 (#3983)
- Bump `pytest-xdist` from 3.4.0 to 3.5.0 (#3982)
- Update `httpx` requirement from ~=0.25.0 to ~=0.25.1 (#3961)
- Bump `srvaroa/labeler` from 1.6.1 to 1.7.0 (#3958)
- Update `cachetools` requirement from ~=5.3.1 to ~=5.3.2 (#3954)
- Bump `pytest` from 7.4.2 to 7.4.3 (#3953)

6.6.28 Version 20.6

Released 2023-10-03

This is the technical changelog for version 20.6. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Drop Backward Compatibility Layer Introduced in #3853 (API 6.8) (#3873)
- Full Support for Bot API 6.9 (#3898)

New Features

- Add Rich Equality Comparison to `WriteAccessAllowed` (#3911 closes #3909)
- Add `__repr__` Methods Added in #3826 closes #3770 to Sphinx Documentation (#3901 closes #3889)
- Add String Representation for Selected Classes (#3826 closes #3770)

Minor Changes

- Add Support Python 3.12 (#3915)
- Documentation Improvements (#3910)

Internal Changes

- Verify Type Hints for Bot Method & Telegram Class Parameters (#3868)
- Move Bot API Tests to Separate Workflow File (#3912)
- Fix Failing `file_size` Tests (#3906)
- Set Threshold for DeepSource's PY-R1000 to High (#3888)
- One-Time Code Formatting Improvement via `--preview` Flag of `black` (#3882)

- Move Dunder Methods to the Top of Class Bodies (#3883)
- Remove Superfluous `Defaults.__ne__` (#3884)

Dependency Updates

- `pre-commit` autoupdate (#3876)
- Update `pre-commit` Dependencies (#3916)
- Bump `actions/checkout` from 3 to 4 (#3914)
- Update `httpx` requirement from ~=0.24.1 to ~=0.25.0 (#3891)
- Bump `furo` from 2023.8.19 to 2023.9.10 (#3890)
- Bump `sphinx` from 7.2.5 to 7.2.6 (#3892)
- Update `tornado` requirement from ~=6.2 to ~=6.3.3 (#3675)
- Bump `pytest` from 7.4.0 to 7.4.2 (#3881)

6.6.29 Version 20.5

Released 2023-09-03

This is the technical changelog for version 20.5. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- API 6.8 (#3853)
- Remove Functionality Deprecated Since Bot API 6.5, 6.6 or 6.7 (#3858)

New Features

- Extend Allowed Values for HTTP Version (#3823 closes #3821)
- Add `has_args` Parameter to `CommandHandler` (#3854 by @thatguylah closes #3798)
- Add `Application.stop_running()` and Improve Marking Updates as Read on `Updater.stop()` (#3804)

Minor Changes

- Type Hinting Fixes for `WebhookInfo` (#3871)
- Test and Document `Exception.__cause__` on `NetworkError` (#3792 closes #3778)
- Add Support for Python 3.12 RC (#3847)

Documentation Improvements

- Remove Version Check from Examples (#3846)
- Documentation Improvements (#3803, #3797, #3816 by @trim21, #3829 by @aelkheir)
- Provide Versions of `customwebhookbot.py` with Different Frameworks (#3820 closes #3717)

Dependency Updates

- `pre-commit` autoupdate (#3824)
- Bump `srvaroa/labeler` from 1.6.0 to 1.6.1 (#3870)
- Bump `sphinx` from 7.0.1 to 7.1.1 (#3818)
- Bump `sphinx` from 7.2.3 to 7.2.5 (#3869)

- Bump `furo` from 2023.5.20 to 2023.7.26 (#3817)
- Update `apscheduler` requirement from ~=3.10.3 to ~=3.10.4 (#3862)
- Bump `sphinx` from 7.2.2 to 7.2.3 (#3861)
- Bump `pytest-asyncio` from 0.21.0 to 0.21.1 (#3801)
- Bump `sphinx-paramlinks` from 0.5.4 to 0.6.0 (#3840)
- Update `apscheduler` requirement from ~=3.10.1 to ~=3.10.3 (#3851)
- Bump `furo` from 2023.7.26 to 2023.8.19 (#3850)
- Bump `sphinx` from 7.1.2 to 7.2.2 (#3852)
- Bump `sphinx` from 7.1.1 to 7.1.2 (#3827)

6.6.30 Version 20.4

Released 2023-07-09

This is the technical changelog for version 20.4. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Drop Support for Python 3.7 (#3728, #3742 by @Trifase, #3749 by @thefunkycat, #3740 closes #3732, #3754 closes #3731, #3753, #3764, #3762, #3759 closes #3733)

New Features

- Make Integration of `APScheduler` into `JobQueue` More Explicit (#3695)
- Introduce `BaseUpdateProcessor` for Customized Concurrent Handling of Updates (#3654 closes #3509)

Minor Changes

- Fix Inconsistent Type Hints for `timeout` Parameter of `Bot.get_updates` (#3709 by @revolter)
- Use Explicit Optionals (#3692 by @MiguelX413)

Bug Fixes

- Fix Wrong Warning Text in `KeyboardButton.__eq__` (#3768)

Documentation Improvements

- Explicitly set `allowed_updates` in Examples (#3741 by @Trifase closes #3726)
- Bump `furo` and `sphinx` (#3719)
- Documentation Improvements (#3698, #3708 by @revolter, #3767)
- Add Quotes for Installation Instructions With Optional Dependencies (#3780)
- Exclude Type Hints from Stability Policy (#3712)
- Set `httpx` Logging Level to Warning in Examples (#3746 closes #3743)

Internal Changes

- Drop a Legacy `pre-commit.ci` Configuration (#3697)
- Add Python 3.12 Beta to the Test Matrix (#3751)
- Use Temporary Files for Testing File Downloads (#3777)
- Auto-Update Changed Version in Other Files After Dependabot PRs (#3716)

- Add More `ruff` Rules (#3763)
- Rename `_handler.py` to `_basehandler.py` (#3761)
- Automatically Label `pre-commit-ci` PRs (#3713)
- Rework `pytest` Integration into GitHub Actions (#3776)
- Fix Two Bugs in GitHub Actions Workflows (#3739)

Dependency Updates

- Update `cachetools` requirement from ~=5.3.0 to ~=5.3.1 (#3738)
- Update `aiolimiter` requirement from ~=1.0.0 to ~=1.1.0 (#3707)
- `pre-commit` autoupdate (#3791)
- Bump `sphinxcontrib-mermaid` from 0.8.1 to 0.9.2 (#3737)
- Bump `pytest-xdist` from 3.2.1 to 3.3.0 (#3705)
- Bump `srvaroa/labeler` from 1.5.0 to 1.6.0 (#3786)
- Bump `dependabot/fetch-metadata` from 1.5.1 to 1.6.0 (#3787)
- Bump `dessant/lock-threads` from 4.0.0 to 4.0.1 (#3785)
- Bump `pytest` from 7.3.2 to 7.4.0 (#3774)
- Update `httpx` requirement from ~=0.24.0 to ~=0.24.1 (#3715)
- Bump `pytest-xdist` from 3.3.0 to 3.3.1 (#3714)
- Bump `pytest` from 7.3.1 to 7.3.2 (#3758)
- `pre-commit` autoupdate (#3747)

6.6.31 Version 20.3

Released 2023-05-07

This is the technical changelog for version 20.3. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Full support for API 6.7 (#3673)
- Add a Stability Policy (#3622)

New Features

- Add `Application.mark_data_for_update_persistence` (#3607)
- Make `Message.link` Point to Thread View Where Possible (#3640)
- Localize Received `datetime` Objects According to `Defaults.tzinfo` (#3632)

Minor Changes, Documentation Improvements and CI

- Empower `ruff` (#3594)
- Drop Usage of `sys.maxunicode` (#3630)
- Add String Representation for `RequestParameter` (#3634)
- Stabilize CI by Rerunning Failed Tests (#3631)
- Give Loggers Better Names (#3623)
- Add Logging for Invalid JSON Data in `BasePersistence.parse_json_payload` (#3668)

- Improve Warning Categories & Stacklevels (#3674)
- Stabilize `test_delete_sticker_set` (#3685)
- Shield Update Fetcher Task in `Application.start` (#3657)
- Recover 100% Type Completeness (#3676)
- Documentation Improvements (#3628, #3636, #3694)

Dependencies

- Bump `actions/stale` from 7 to 8 (#3644)
- Bump `furo` from 2023.3.23 to 2023.3.27 (#3643)
- `pre-commit` autoupdate (#3646, #3688)
- Remove Deprecated `codecov` Package from CI (#3664)
- Bump `sphinx-copybutton` from 0.5.1 to 0.5.2 (#3662)
- Update `httpx` requirement from ~=0.23.3 to ~=0.24.0 (#3660)
- Bump `pytest` from 7.2.2 to 7.3.1 (#3661)

6.6.32 Version 20.2

Released 2023-03-25

This is the technical changelog for version 20.2. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Full Support for API 6.6 (#3584)
- Revert to HTTP/1.1 as Default and make HTTP/2 an Optional Dependency (#3576)

Minor Changes, Documentation Improvements and CI

- Documentation Improvements (#3565, #3600)
- Handle Symbolic Links in `was_called_by` (#3552)
- Tidy Up Tests Directory (#3553)
- Enhance `Application.create_task` (#3543)
- Make Type Completeness Workflow Usable for `PRs` from Forks (#3551)
- Refactor and Overhaul the Test Suite (#3426)

Dependencies

- Bump `pytest-asyncio` from 0.20.3 to 0.21.0 (#3624)
- Bump `furo` from 2022.12.7 to 2023.3.23 (#3625)
- Bump `pytest-xdist` from 3.2.0 to 3.2.1 (#3606)
- `pre-commit` autoupdate (#3577)
- Update `apscheduler` requirement from ~=3.10.0 to ~=3.10.1 (#3572)
- Bump `pytest` from 7.2.1 to 7.2.2 (#3573)
- Bump `pytest-xdist` from 3.1.0 to 3.2.0 (#3550)
- Bump `sphinxcontrib-mermaid` from 0.7.1 to 0.8 (#3549)

6.6.33 Version 20.1

Released 2023-02-09

This is the technical changelog for version 20.1. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Full Support for Bot API 6.5 (#3530)

New Features

- Add `Application(Builder).post_stop` (#3466)
- Add `Chat.effective_name` Convenience Property (#3485)
- Allow to Adjust HTTP Version and Use HTTP/2 by Default (#3506)

Documentation Improvements

- Enhance `chatmemberbot` Example (#3500)
- Automatically Generate Cross-Reference Links (#3501, #3529, #3523)
- Add Some Graphic Elements to Docs (#3535)
- Various Smaller Improvements (#3464, #3483, #3484, #3497, #3512, #3515, #3498)

Minor Changes, Documentation Improvements and CI

- Update Copyright to 2023 (#3459)
- Stabilize Tests on Closing and Hiding the General Forum Topic (#3460)
- Fix Dependency Warning Typo (#3474)
- Cache Dependencies on `GitHub` Actions (#3469)
- Store Documentation Builts as `GitHub` Actions Artifacts (#3468)
- Add `ruff` to `pre-commit` Hooks (#3488)
- Improve Warning for `days` Parameter of `JobQueue.run_daily` (#3503)
- Improve Error Message for `NetworkError` (#3505)
- Lock Inactive Threads Only Once Each Day (#3510)
- Bump `pytest` from 7.2.0 to 7.2.1 (#3513)
- Check for 3D Arrays in `check_keyboard_type` (#3514)
- Explicit Type Annotations (#3508)
- Increase Verbosity of Type Completeness CI Job (#3531)
- Fix CI on Python 3.11 + Windows (#3547)

Dependencies

- Bump `actions/stale` from 6 to 7 (#3461)
- Bump `dessant/lock-threads` from 3.0.0 to 4.0.0 (#3462)
- `pre-commit` autoupdate (#3470)
- Update `httpx` requirement from ~=0.23.1 to ~=0.23.3 (#3489)
- Update `cachetools` requirement from ~=5.2.0 to ~=5.2.1 (#3502)
- Improve Config for `ruff` and Bump to `v0.0.222` (#3507)

- Update `cachetools` requirement from ~=5.2.1 to ~=5.3.0 (#3520)
- Bump `isort` to 5.12.0 (#3525)
- Update `apscheduler` requirement from ~=3.9.1 to ~=3.10.0 (#3532)
- `pre-commit` autoupdate (#3537)
- Update `cryptography` requirement to >=39.0.1 to address Vulnerability (#3539)

6.6.34 Version 20.0

Released 2023-01-01

This is the technical changelog for version 20.0. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Full Support For Bot API 6.4 (#3449)

Minor Changes, Documentation Improvements and CI

- Documentation Improvements (#3428, #3423, #3429, #3441, #3404, #3443)
- Allow `Sequence` Input for Bot Methods (#3412)
- Update Link-Check CI and Replace a Dead Link (#3456)
- Freeze Classes Without Arguments (#3453)
- Add New Constants (#3444)
- Override `Bot.__deepcopy__` to Raise `TypeError` (#3446)
- Add Log Decorator to `Bot.get_webhook_info` (#3442)
- Add Documentation On Verifying Releases (#3436)
- Drop Undocumented `Job.__lt__` (#3432)

Dependencies

- Downgrade `sphinx` to 5.3.0 to Fix Search (#3457)
- Bump `sphinx` from 5.3.0 to 6.0.0 (#3450)

6.6.35 Version 20.0b0

Released 2022-12-15

This is the technical changelog for version 20.0b0. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- Make `TelegramObject` Immutable (#3249)

Minor Changes, Documentation Improvements and CI

- Reduce Code Duplication in Testing `Defaults` (#3419)
- Add Notes and Warnings About Optional Dependencies (#3393)
- Simplify Internals of `Bot` Methods (#3396)
- Reduce Code Duplication in Several `Bot` Methods (#3385)
- Documentation Improvements (#3386, #3395, #3398, #3403)

Dependencies

- Bump `pytest-xdist` from 3.0.2 to 3.1.0 (#3415)
- Bump `pytest-asyncio` from 0.20.2 to 0.20.3 (#3417)
- `pre-commit` autoupdate (#3409)

6.6.36 Version 20.0a6

Released 2022-11-24

This is the technical changelog for version 20.0a6. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Bug Fixes

- Only Persist Arbitrary `callback_data` if `ExtBot.callback_data_cache` is Present (#3384)
- Improve Backwards Compatibility of `TelegramObjects` Pickle Behavior (#3382)
- Fix Naming and Keyword Arguments of `File.download_*` Methods (#3380)
- Fix Return Value Annotation of `Chat.create_forum_topic` (#3381)

6.6.37 Version 20.0a5

Released 2022-11-22

This is the technical changelog for version 20.0a5. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes

- API 6.3 (#3346, #3343, #3342, #3360)
- Explicit `local_mode` Setting (#3154)
- Make Almost All 3rd Party Dependencies Optional (#3267)
- Split `File.download` Into `File.download_to_drive` And `File.download_to_memory` (#3223)

New Features

- Add Properties for API Settings of `Bot` (#3247)
- Add `chat_id` and `username` Parameters to `ChatJoinRequestHandler` (#3261)
- Introduce `TelegramObject.api_kwargs` (#3233)
- Add Two Constants Related to Local Bot API Servers (#3296)
- Add `recursive` Parameter to `TelegramObject.to_dict()` (#3276)
- Overhaul String Representation of `TelegramObject` (#3234)
- Add Methods `Chat.mention_{html, markdown, markdown_v2}` (#3308)
- Add `constants.MessageLimit.DEEP_LINK_LENGTH` (#3315)
- Add Shortcut Parameters `caption`, `parse_mode` and `caption_entities` to `Bot.send_media_group` (#3295)
- Add Several New Enums To Constants (#3351)

Bug Fixes

- Fix `CallbackQueryHandler` Not Handling Non-String Data Correctly With Regex Patterns (#3252)
- Fix Defaults Handling in `Bot.answer_web_app_query` (#3362)

Documentation Improvements

- Update PR Template (#3361)
- Document Dunder Methods of `TelegramObject` (#3319)
- Add Several References to Wiki pages (#3306)
- Overhaul Search bar (#3218)
- Unify Documentation of Arguments and Attributes of Telegram Classes (#3217, #3292, #3303, #3312, #3314)
- Several Smaller Improvements (#3214, #3271, #3289, #3326, #3370, #3376, #3366)

Minor Changes, Documentation Improvements and CI

- Improve Warning About Unknown `ConversationHandler` States (#3242)
- Switch from Stale Bot to `GitHub` Actions (#3243)
- Bump Python 3.11 to RC2 in Test Matrix (#3246)
- Make `Job.job` a Property and Make `Jobs` Hashable (#3250)
- Skip `JobQueue` Tests on Windows Again (#3280)
- Read-Only `CallbackDataCache` (#3266)
- Type Hinting Fix for `Message.effective_attachment` (#3294)
- Run Unit Tests in Parallel (#3283)
- Update Test Matrix to Use Stable Python 3.11 (#3313)
- Don't Edit Objects In-Place When Inserting `ext.Defaults` (#3311)
- Add a Test for `MessageAttachmentType` (#3335)
- Add Three New Test Bots (#3347)
- Improve Unit Tests Regarding `ChatMemberUpdated.difference` (#3352)
- Flaky Unit Tests: Use `pytest` Marker (#3354)
- Fix `DeepSource` Issues (#3357)
- Handle Lists and Tuples and Datetimes Directly in `TelegramObject.to_dict` (#3353)
- Update Meta Config (#3365)
- Merge `ChatDescriptionLimit` Enum Into `ChatLimit` (#3377)

Dependencies

- Bump `pytest` from 7.1.2 to 7.1.3 (#3228)
- `pre-commit` Updates (#3221)
- Bump `sphinx` from 5.1.1 to 5.2.3 (#3269)
- Bump `furo` from 2022.6.21 to 2022.9.29 (#3268)
- Bump `actions/stale` from 5 to 6 (#3277)
- `pre-commit` autoupdate (#3282)
- Bump `sphinx` from 5.2.3 to 5.3.0 (#3300)

- Bump `pytest-asyncio` from 0.19.0 to 0.20.1 (#3299)
- Bump `pytest` from 7.1.3 to 7.2.0 (#3318)
- Bump `pytest-xdist` from 2.5.0 to 3.0.2 (#3317)
- `pre-commit` autoupdate (#3325)
- Bump `pytest-asyncio` from 0.20.1 to 0.20.2 (#3359)
- Update `httpx` requirement from ~=0.23.0 to ~=0.23.1 (#3373)

6.6.38 Version 20.0a4

Released 2022-08-27

This is the technical changelog for version 20.0a4. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Hot Fixes

- Fix a Bug in `setup.py` Regarding Optional Dependencies (#3209)

6.6.39 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)

6.6.40 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)

6.6.41 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,:pr:*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)

6.6.42 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)

6.6.43 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)

6.6.44 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)

6.6.45 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)

6.6.46 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)

6.6.47 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)

6.6.48 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)

6.6.49 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)

6.6.50 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)

6.6.51 Version 13.4.1

Released 2021-03-14

Hot fix release:

- Fixed a bug in `setup.py` (#2431)

6.6.52 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)

6.6.53 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)

6.6.54 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)

6.6.55 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)

6.6.56 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)

6.6.57 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)

6.6.58 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)

6.6.59 Version 12.6.1

Released 2020-04-11

Bug fixes:

- Fix serialization of `reply_markup` in media messages (#1889)

6.6.60 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)

6.6.61 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)

6.6.62 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)

6.6.63 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)

6.6.64 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.

6.6.65 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)

6.6.66 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).

6.6.67 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).

6.6.68 Version 12.1.1

Released 2019-09-18

Hot fix release

Fixed regression in the vendored urllib3 (#1517).

6.6.69 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)

6.6.70 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

6.6.71 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

6.6.72 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)

6.6.73 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)

6.6.74 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.

6.6.75 Version 10.0.1

Released 2018-03-05

Fixes:

- Fix conversationhandler timeout (PR #1032)
- Add missing docs utils (PR #912)

6.6.76 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)

6.6.77 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.

6.6.78 Version 8.1.1

Released 2017-10-15

- Fix Commandhandler crashing on single character messages (PR #873).

6.6.79 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.

6.6.80 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.

6.6.81 Version 7.0.1

Released 2017-07-28

- Fix TypeError exception in RegexHandler (PR #751).
- Small documentation fix (PR #749).

6.6.82 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.

6.6.83 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` - #484
- Download files into file-like objects - #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` - #507
- Add support for Socks5 proxy - #518
- Add support for filters in `CommandHandler` - #536
- Add the ability to invert (not) filters - #552
- Add `Filters.group` and `Filters.private`
- Compatibility with GAE via `urllib3.contrib` package - #583
- Add equality rich comparision operators to telegram objects - #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` - #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

6.7 Contributor Covenant Code of Conduct

6.7.1 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.

6.7.2 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

6.7.3 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.

6.7.4 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.

6.7.5 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.

6.7.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4.

6.8 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.

6.8.1 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
$ 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 the package in development mode as well as optional dependencies and development dependencies. Note that the *–group* argument requires *pip* 25.1 or later.

   Alternatively, you can use your preferred package manager (such as uv, hatch, poetry, etc.) instead of pip.

```
$ pip install -e .[all] --group all
```

   Installing the package itself is necessary because python-telegram-bot uses a src-based layout where the package code is located in the `src/` directory.

5. Install pre-commit hooks:

```
$ prek install -f
```

6.8.2 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.

6.8.3 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. We also have a checklist for PRs *below*.

   - 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 Ruff for linting and formatting.
   - Please ensure that the code you write is well-tested and that all automated tests still pass. We have dedicated an testing page to help you with that.
   - Don't break backward compatibility.
   - Add yourself to the AUTHORS.rst file in an alphabetical fashion.
   - If you want run style & type checks before committing run

     ```
     $ prek 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`!

Check-list for PRs

This checklist is a non-exhaustive reminder of things that should be done before a PR is merged, both for you as contributor and for the maintainers. Feel free to copy (parts of) the checklist to the PR description to remind you or the maintainers of open points or if you have questions on anything.

```
## Check-list for PRs

- [ ] Added `.. versionadded:: NEXT.VERSION`, ``.. versionchanged:: NEXT.VERSION``,␣
↪``.. deprecated:: NEXT.VERSION`` or ``.. versionremoved:: NEXT.VERSION` to the␣
↪docstrings for user facing changes (for methods/class descriptions, arguments and␣
↪attributes)
- [ ] Created new or adapted existing unit tests
- [ ] Documented code changes according to the [CSI standard](https://standards.
↪mousepawmedia.com/en/stable/csi.html)
- [ ] Added myself alphabetically to `AUTHORS.rst` (optional)
- [ ] Added new classes & modules to the docs and all suitable ``__all__`` s
- [ ] Checked the [Stability Policy](https://docs.python-telegram-bot.org/stability_
↪policy.html) in case of deprecations or changes to documented behavior

**If the PR contains API changes (otherwise, you can ignore this passage)**

- [ ] Checked the Bot API specific sections of the [Stability Policy](https://docs.
↪python-telegram-bot.org/stability_policy.html)
- [ ] Created a PR to remove functionality deprecated in the previous Bot API release␣
↪([see here](https://docs.python-telegram-bot.org/en/stable/stability_policy.html
↪#case-2))

- New Classes

    - [ ] Added `self._id_attrs` and corresponding documentation
    - [ ] `__init__` accepts `api_kwargs` as keyword-only

- Added New Shortcuts
```

(continues on next page)

(continued from previous page)

```
    - [ ] In [`telegram.Chat`](https://python-telegram-bot.readthedocs.io/en/stable/
↪telegram.chat.html) \& [`telegram.User`](https://python-telegram-bot.readthedocs.io/
↪en/stable/telegram.user.html) for all methods that accept `chat/user_id`
    - [ ] In [`telegram.Message`](https://python-telegram-bot.readthedocs.io/en/
↪stable/telegram.message.html) for all methods that accept `chat_id` and `message_id`
    - [ ] For new `telegram.Message` shortcuts: Added `quote` argument if methods␣
↪accept `reply_to_message_id`
    - [ ] In [`telegram.CallbackQuery`](https://python-telegram-bot.readthedocs.io/en/
↪stable/telegram.callbackquery.html) for all methods that accept either `chat_id`␣
↪and `message_id` or `inline_message_id`

- If Relevant

    - [ ] Added new constants at `telegram.constants` and shortcuts to them as class␣
↪variables
    - [ ] Linked new and existing constants in docstrings instead of hard-coded␣
↪numbers and strings
    - [ ] Added new message types to `telegram.Message.effective_attachment`
    - [ ] Added new handlers for new update types
        - [ ] Added the handlers to the warning loop in the [`telegram.ext.
↪ConversationHandler`](https://python-telegram-bot.readthedocs.io/en/stable/telegram.
↪ext.conversationhandler.html)
    - [ ] Added new filters for new message (sub)types
    - [ ] Added or updated documentation for the changed class(es) and/or method(s)
    - [ ] Added the new method(s) to `_extbot.py`
    - [ ] Added or updated `bot_methods.rst`
    - [ ] Updated the Bot API version number in all places: `README.rst` (including␣
↪the badge) and `telegram.constants.BOT_API_VERSION_INFO`
    - [ ] Added logic for arbitrary callback data in `telegram.ext.ExtBot` for new␣
↪methods that either accept a `reply_markup` in some form or have a return type that␣
↪is/contains [`telegram.Message`](https://python-telegram-bot.readthedocs.io/en/
↪stable/telegram.message.html)
```

6.8.4 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're running Python 3.10 or above and have the required dependencies installed as explained above. 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:: NEXT.VERSION`, `.. versionchanged:: NEXT.VERSION` or `.. deprecated:: NEXT.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.

6.8.5 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.

6.9 Testing in PTB

PTB uses pytest for testing. To run the tests, you need to have pytest installed along with a few other dependencies. You can find the list of dependencies in the `pyproject.toml` file in the root of the repository.

Since PTB uses a src-based layout, make sure you have installed the package in development mode before running the tests:

```
$ pip install -e .
```

6.9.1 Running tests

To run the entire test suite, you can use the following command:

```
$ pytest
```

This will run all the tests, including the ones which make a request to the Telegram servers, which may take a long time (total > 13 mins). To run only the tests that don't require a connection, you can run the following command:

```
$ pytest -m no_req
```

Or alternatively, you can run the following command to run only the tests that require a connection:

```
$ pytest -m req
```

To further speed up the tests, you can run them in parallel using the `-n` flag (requires pytest-xdist). But beware that this will use multiple CPU cores on your machine. The `--dist` flag is used to specify how the tests will be distributed across the cores. The `loadgroup` option is used to distribute the tests such that tests marked with `@pytest.mark.xdist_group("name")` are run on the same core — important if you want avoid race conditions in some tests:

```
$ pytest -n auto --dist=worksteal
```

This will result in a significant speedup, but may cause some tests to fail. If you want to run the failed tests in isolation, you can use the `--lf` flag:

```
$ pytest --lf
```

6.9.2 Writing tests

PTB has a separate test file for every file in the `telegram.*` namespace. Further, the tests for the `telegram` module are split into two classes, based on whether the test methods in them make a request or not. When writing tests, make sure to split them into these two classes, and make sure to name the test class as: `TestXXXWithoutRequest` for tests that don't make a request, and `TestXXXWithRequest` for tests that do.

Writing tests is a creative process; allowing you to design your test however you'd like, but there are a few conventions that you should follow:

- Each new test class needs a `test_slot_behaviour`, `test_to_dict`, `test_de_json` and `test_equality` (in most cases).
- Make use of pytest's fixtures and parametrize wherever possible. Having knowledge of pytest's tooling can help you as well. You can look at the existing tests for examples and inspiration.
- New fixtures should go into `conftest.py`. New auxiliary functions and classes, used either directly in the tests or in the fixtures, should go into the `tests/auxil` directory.

If you have made some API changes, you may want to run `test_official` to validate that the changes are complete and correct. To run it, export an environment variable first:

```
$ export TEST_OFFICIAL=true
```

and then run `pytest tests/test_official/test_official.py`. Note: You need py 3.10+ to run this test.

We also have another marker, `@pytest.mark.dev`, which you can add to tests that you want to run selectively. Use as follows:

```
$ pytest -m dev
```

6.9.3 Debugging tests

Writing tests can be challenging, and fixing failing tests can be even more so. To help with this, PTB has started to adopt the use of `logging` in the test suite. You can insert debug logging statements in your tests to help you understand what's going on. To enable these logs, you can set `log_level = DEBUG` in `pyproject.toml` or use

the `--log-level=INFO` flag when running the tests. If a test is large and complicated, it is recommended to leave the debug logs for others to use as well.

6.9.4 Bots used in tests

If you run the tests locally, the test setup will use one of the two public bots available. Which bot of the two gets chosen for the test session is random. Whereas when the tests on the Github Actions CI are run, the test setup allocates a different, but the same bot is allocated for every combination of Python version and OS. The operating systems and Python versions the CI runs the tests on can be viewed in the corresponding workflow.

That's it! If you have any questions, feel free to ask them in the PTB dev group.

PYTHON MODULE INDEX

t

telegram, 13
telegram.constants, 960
telegram.error, 1036
telegram.ext, 803
telegram.ext.filters, 888
telegram.helpers, 1040
telegram.warnings, 1049

INDEX

Symbols

__aenter__() (*telegram.Bot method*), 20
__aenter__() (*telegram.ext.Application method*), 806
__aenter__() (*telegram.ext.BaseUpdateProcessor method*), 838
__aenter__() (*telegram.ext.Updater method*), 863
__aenter__() (*telegram.request.BaseRequest method*), 1043
__aexit__() (*telegram.Bot method*), 20
__aexit__() (*telegram.ext.Application method*), 807
__aexit__() (*telegram.ext.BaseUpdateProcessor method*), 838
__aexit__() (*telegram.ext.Updater method*), 864
__aexit__() (*telegram.request.BaseRequest method*), 1043
__and__() (*telegram.ext.filters.BaseFilter method*), 893
__bot_api_version__ (*in module telegram*), 13
__bot_api_version_info__ (*in module telegram*), 13
__deepcopy__() (*telegram.Bot method*), 20
__deepcopy__() (*telegram.TelegramObject method*), 608
__delattr__() (*telegram.TelegramObject method*), 609
__eq__() (*telegram.Bot method*), 20
__eq__() (*telegram.TelegramObject method*), 609
__eq__() (*telegram.ext.Defaults method*), 847
__eq__() (*telegram.ext.Job method*), 853
__getattr__() (*telegram.ext.Job method*), 853
__getitem__() (*telegram.TelegramObject method*), 609
__getstate__() (*telegram.TelegramObject method*), 609
__hash__() (*telegram.Bot method*), 20
__hash__() (*telegram.TelegramObject method*), 610
__hash__() (*telegram.ext.Defaults method*), 847
__hash__() (*telegram.ext.Job method*), 853
__invert__() (*telegram.ext.filters.BaseFilter method*), 893
__or__() (*telegram.ext.filters.BaseFilter method*), 893
__reduce__() (*telegram.Bot method*), 21
__reduce__() (*telegram.error.ChatMigrated method*), 1037
__reduce__() (*telegram.error.Conflict method*), 1037
__reduce__() (*telegram.error.PassportDecryptionError method*), 1038
__reduce__() (*telegram.error.RetryAfter method*), 1039
__reduce__() (*telegram.error.TelegramError method*), 1039
__reduce__() (*telegram.ext.InvalidCallbackData method*), 956
__repr__() (*telegram.Bot method*), 21
__repr__() (*telegram.TelegramObject method*), 610
__repr__() (*telegram.error.TelegramError method*), 1039
__repr__() (*telegram.ext.Application method*), 807
__repr__() (*telegram.ext.BaseHandler method*), 869
__repr__() (*telegram.ext.ConversationHandler method*), 887
__repr__() (*telegram.ext.Job method*), 853
__repr__() (*telegram.ext.JobQueue method*), 856
__repr__() (*telegram.ext.Updater method*), 864
__repr__() (*telegram.ext.filters.BaseFilter method*), 893
__setattr__() (*telegram.TelegramObject method*), 610
__setstate__() (*telegram.TelegramObject method*), 610
__str__() (*telegram.error.TelegramError method*), 1039
__str__() (*telegram.warnings.PTBDeprecationWarning method*), 1050
__version__ (*in module telegram*), 13
__version_info__ (*in module telegram*), 13
__xor__() (*telegram.ext.filters.BaseFilter method*), 893

A

accent_color_id (*telegram.ChatFullInfo attribute*), 308
AccentColor (*class in telegram.constants*), 960
accepted_gift_types (*telegram.ChatFullInfo attribute*), 308
AcceptedGiftTypes (*class in telegram*), 219
active_usernames (*telegram.ChatFullInfo attribute*), 309
ACTIVITY_ONE_DAY (*telegram.constants.StoryLimit attribute*), 1028
ACTIVITY_SIX_HOURS (*telegram.constants.StoryLimit attribute*), 1028

ACTIVITY_TWELVE_HOURS (*telegram.constants.StoryLimit attribute*), 1028
ACTIVITY_TWO_DAYS (*telegram.constants.StoryLimit attribute*), 1028
actor_chat (*telegram.MessageReactionUpdated attribute*), 544
add_bot_ids() (*telegram.ext.filters.ViaBot method*), 917
add_chat_ids() (*telegram.ext.filters.Chat method*), 895
add_chat_ids() (*telegram.ext.filters.ForwardedFrom method*), 903
add_chat_ids() (*telegram.ext.filters.SenderChat method*), 907
add_date (*telegram.ChatBoost attribute*), 298
add_error_handler() (*telegram.ext.Application method*), 807
add_handler() (*telegram.ext.Application method*), 808
add_handlers() (*telegram.ext.Application method*), 808
add_sticker_to_set() (*telegram.Bot method*), 21
add_user_ids() (*telegram.ext.filters.User method*), 915
add_usernames() (*telegram.ext.filters.Chat method*), 896
add_usernames() (*telegram.ext.filters.ForwardedFrom method*), 903
add_usernames() (*telegram.ext.filters.SenderChat method*), 907
add_usernames() (*telegram.ext.filters.User method*), 915
add_usernames() (*telegram.ext.filters.ViaBot method*), 916
added_by_chat (*telegram.PollOption attribute*), 571
added_by_user (*telegram.PollOption attribute*), 571
added_to_attachment_menu (*telegram.User attribute*), 635
added_users (*telegram.BotAccessSettings attribute*), 225
addition_date (*telegram.PollOption attribute*), 571
additional_chat_count (*telegram.GiveawayWinners attribute*), 399
address (*telegram.BusinessLocation attribute*), 237
address (*telegram.ChatLocation attribute*), 349
address (*telegram.InlineQueryResultVenue attribute*), 733
address (*telegram.InputMediaVenue attribute*), 431
address (*telegram.InputVenueMessageContent attribute*), 746
address (*telegram.SecureData attribute*), 801
address (*telegram.StoryAreaTypeLocation attribute*), 597
address (*telegram.Venue attribute*), 664
addStickerToSet() (*telegram.Bot method*), 21
adjust_message_entities_to_utf_16() (*telegram.MessageEntity static method*), 534
ADMINISTRATOR (*telegram.ChatMember attribute*), 350
ADMINISTRATOR (*telegram.constants.ChatMemberStatus attribute*), 972
affiliate (*telegram.TransactionPartnerUser attribute*), 775
affiliate_chat (*telegram.AffiliateInfo attribute*), 753
AFFILIATE_PROGRAM (*telegram.constants.TransactionPartnerType attribute*), 1030
AFFILIATE_PROGRAM (*telegram.TransactionPartner attribute*), 770
affiliate_user (*telegram.AffiliateInfo attribute*), 753
AffiliateInfo (*class in telegram*), 752
AIORateLimiter (*class in telegram.ext*), 958
ALIEN_MONSTER (*telegram.constants.ReactionEmoji attribute*), 1015
ALL (*in module telegram.ext.filters*), 889
ALL (*telegram.ext.filters.Dice attribute*), 898
ALL (*telegram.ext.filters.Document attribute*), 899
ALL (*telegram.ext.filters.SenderChat attribute*), 907
ALL (*telegram.ext.filters.StatusUpdate attribute*), 908
ALL (*telegram.ext.filters.Sticker attribute*), 911
ALL_CHAT_ADMINISTRATORS (*telegram.BotCommandScope attribute*), 227
ALL_CHAT_ADMINISTRATORS (*telegram.constants.BotCommandScopeType attribute*), 965
ALL_EMOJI (*telegram.Dice attribute*), 374
ALL_GROUP_CHATS (*telegram.BotCommandScope attribute*), 227
ALL_GROUP_CHATS (*telegram.constants.BotCommandScopeType attribute*), 965
all_permissions() (*telegram.ChatPermissions class method*), 369
ALL_PRIVATE_CHATS (*telegram.BotCommandScope attribute*), 227
ALL_PRIVATE_CHATS (*telegram.constants.BotCommandScopeType attribute*), 965
all_rights() (*telegram.ChatAdministratorRights class method*), 284
ALL_TYPES (*telegram.MessageEntity attribute*), 533
ALL_TYPES (*telegram.Update attribute*), 628
allow_bot_chats (*telegram.SwitchInlineQueryChosenChat attribute*), 607
allow_channel_chats (*telegram.SwitchInlineQueryChosenChat attribute*), 607
allow_empty (*telegram.ext.filters.Chat attribute*), 895
allow_empty (*telegram.ext.filters.ForwardedFrom attribute*), 903
allow_empty (*telegram.ext.filters.SenderChat at-*

tribute), 906
allow_empty (*telegram.ext.filters.User attribute*), 914
allow_empty (*telegram.ext.filters.ViaBot attribute*), 916
allow_group_chats (*telegram.SwitchInlineQueryChosenChat attribute*), 607
allow_reentry (*telegram.ext.ConversationHandler property*), 887
allow_sending_without_reply (*telegram.ext.Defaults property*), 847
allow_sending_without_reply (*telegram.ReplyParameters attribute*), 589
allow_user_chats (*telegram.SwitchInlineQueryChosenChat attribute*), 607
allowed_updates (*telegram.WebhookInfo attribute*), 677
allows_multiple_answers (*telegram.Poll attribute*), 561
allows_revoting (*telegram.Poll attribute*), 563
allows_users_to_create_topics (*telegram.User attribute*), 636
amount (*telegram.AffiliateInfo attribute*), 753
amount (*telegram.LabeledPrice attribute*), 756
amount (*telegram.StarAmount attribute*), 765
amount (*telegram.StarTransaction attribute*), 766
amount (*telegram.SuggestedPostPaid attribute*), 603
amount (*telegram.SuggestedPostPrice attribute*), 605
ANIMATED (*telegram.constants.InputProfilePhotoType attribute*), 984
ANIMATED (*telegram.constants.StickerFormat attribute*), 1024
ANIMATED (*telegram.ext.filters.Sticker attribute*), 911
ANIMATED (*telegram.InputProfilePhoto attribute*), 440
Animation (*class in telegram*), 220
ANIMATION (*in module telegram.ext.filters*), 889
ANIMATION (*telegram.constants.BaseInputMediaType attribute*), 964
ANIMATION (*telegram.constants.InputMediaType attribute*), 983
ANIMATION (*telegram.constants.MessageAttachmentType attribute*), 992
ANIMATION (*telegram.constants.MessageType attribute*), 1000
animation (*telegram.ExternalReplyInfo attribute*), 380
animation (*telegram.Game attribute*), 778
animation (*telegram.InputProfilePhotoAnimated attribute*), 441
animation (*telegram.Message attribute*), 479
animation (*telegram.PollMedia attribute*), 569
ANONYMOUS_ADMIN (*telegram.constants.ChatID attribute*), 970
answer() (*telegram.CallbackQuery method*), 243
answer() (*telegram.InlineQuery method*), 693
answer() (*telegram.PreCheckoutQuery method*), 758
answer() (*telegram.ShippingQuery method*), 764
answer_callback_query() (*telegram.Bot method*), 22
ANSWER_CALLBACK_QUERY_TEXT_LENGTH (*telegram.constants.CallbackQueryLimit attribute*), 968
answer_guest_query() (*telegram.Bot method*), 23
answer_guest_query() (*telegram.Message method*), 490
answer_inline_query() (*telegram.Bot method*), 24
answer_pre_checkout_query() (*telegram.Bot method*), 25
answer_shipping_query() (*telegram.Bot method*), 26
answer_web_app_query() (*telegram.Bot method*), 27
answerCallbackQuery() (*telegram.Bot method*), 22
answerGuestQuery() (*telegram.Bot method*), 22
answerInlineQuery() (*telegram.Bot method*), 22
answerPreCheckoutQuery() (*telegram.Bot method*), 22
answerShippingQuery() (*telegram.Bot method*), 22
answerWebAppQuery() (*telegram.Bot method*), 22
ANY_CHAT_BOOST (*telegram.ext.ChatBoostHandler attribute*), 876
ANY_CHAT_MEMBER (*telegram.ext.ChatMemberHandler attribute*), 879
api_kwargs (*telegram.TelegramObject attribute*), 608
APK (*telegram.ext.filters.Document attribute*), 901
Application (*class in telegram.ext*), 803
application (*telegram.ext.CallbackContext property*), 841
APPLICATION (*telegram.ext.filters.Document attribute*), 899
application (*telegram.ext.JobQueue property*), 856
application_class() (*telegram.ext.ApplicationBuilder method*), 821
ApplicationBuilder (*class in telegram.ext*), 820
ApplicationHandlerStop (*class in telegram.ext*), 837
approve() (*telegram.ChatJoinRequest method*), 348
approve_chat_join_request() (*telegram.Bot method*), 28
approve_join_request() (*telegram.Chat method*), 251
approve_join_request() (*telegram.ChatFullInfo method*), 314
approve_join_request() (*telegram.User method*), 636
approve_suggested_post() (*telegram.Bot method*), 28
approve_suggested_post() (*telegram.Chat method*), 251
approve_suggested_post() (*telegram.ChatFullInfo method*), 314
approve_suggested_post() (*telegram.Message method*), 491
approveChatJoinRequest() (*telegram.Bot method*), 27
APPROVED (*telegram.constants.SuggestedPostInfoState*

attribute), 1029
APPROVED (*telegram.SuggestedPostInfo attribute*), 602
approveSuggestedPost() (*telegram.Bot method*), 27
arbitrary_callback_data() (*telegram.ext.ApplicationBuilder method*), 821
are_direct_messages_enabled (*telegram.DirectMessagePriceChanged attribute*), 375
args (*telegram.ext.CallbackContext attribute*), 840
ARTICLE (*telegram.constants.InlineQueryResultType attribute*), 980
attach_name (*telegram.InputFile attribute*), 414
attach_uri (*telegram.InputFile property*), 415
ATTACHMENT (*in module telegram.ext.filters*), 889
Audio (*class in telegram*), 222
AUDIO (*in module telegram.ext.filters*), 889
AUDIO (*telegram.constants.BaseInputMediaType attribute*), 964
AUDIO (*telegram.constants.InlineQueryResultType attribute*), 980
AUDIO (*telegram.constants.InputMediaType attribute*), 983
AUDIO (*telegram.constants.MessageAttachmentType attribute*), 992
AUDIO (*telegram.constants.MessageType attribute*), 1000
AUDIO (*telegram.ext.filters.Document attribute*), 900
audio (*telegram.ExternalReplyInfo attribute*), 380
audio (*telegram.Message attribute*), 478
audio (*telegram.PollMedia attribute*), 569
audio_duration (*telegram.InlineQueryResultAudio attribute*), 698
audio_file_id (*telegram.InlineQueryResultCachedAudio attribute*), 699
audio_url (*telegram.InlineQueryResultAudio attribute*), 697
audios (*telegram.UserProfileAudios attribute*), 660
author_signature (*telegram.Message attribute*), 482
author_signature (*telegram.MessageOriginChannel attribute*), 539
author_signature (*telegram.MessageOriginChat attribute*), 540
available_reactions (*telegram.ChatFullInfo attribute*), 309

B

backdrop (*telegram.UniqueGift attribute*), 614
background (*telegram.Gift attribute*), 680
background_color (*telegram.StoryAreaTypeWeather attribute*), 599
background_custom_emoji_id (*telegram.ChatFullInfo attribute*), 309
BackgroundFill (*class in telegram*), 295
BackgroundFillFreeformGradient (*class in telegram*), 297
BackgroundFillGradient (*class in telegram*), 296
BackgroundFillLimit (*class in telegram.constants*), 962
BackgroundFillSolid (*class in telegram*), 296
BackgroundFillType (*class in telegram.constants*), 962
BackgroundType (*class in telegram*), 291
BackgroundTypeChatTheme (*class in telegram*), 294
BackgroundTypeFill (*class in telegram*), 292
BackgroundTypeLimit (*class in telegram.constants*), 962
BackgroundTypePattern (*class in telegram*), 293
BackgroundTypeType (*class in telegram.constants*), 963
BackgroundTypeWallpaper (*class in telegram*), 292
BadRequest, 1036
ban_chat() (*telegram.Chat method*), 251
ban_chat() (*telegram.ChatFullInfo method*), 314
ban_chat_member() (*telegram.Bot method*), 29
ban_chat_sender_chat() (*telegram.Bot method*), 30
ban_member() (*telegram.Chat method*), 252
ban_member() (*telegram.ChatFullInfo method*), 315
ban_sender_chat() (*telegram.Chat method*), 252
ban_sender_chat() (*telegram.ChatFullInfo method*), 315
BANANA (*telegram.constants.ReactionEmoji attribute*), 1015
banChatMember() (*telegram.Bot method*), 29
banChatSenderChat() (*telegram.Bot method*), 29
bank_statement (*telegram.SecureData attribute*), 801
BANNED (*telegram.ChatMember attribute*), 350
BANNED (*telegram.constants.ChatMemberStatus attribute*), 972
base_file_url (*telegram.Bot property*), 31
base_file_url() (*telegram.ext.ApplicationBuilder method*), 822
base_name (*telegram.UniqueGift attribute*), 614
base_url (*telegram.Bot property*), 31
base_url() (*telegram.ext.ApplicationBuilder method*), 822
BaseFilter (*class in telegram.ext.filters*), 892
BaseHandler (*class in telegram.ext*), 867
BaseInputMediaType (*class in telegram.constants*), 963
BasePersistence (*class in telegram.ext*), 937
BaseRateLimiter (*class in telegram.ext*), 956
BaseRequest (*class in telegram.request*), 1042
BaseUpdateProcessor (*class in telegram.ext*), 837
BASKETBALL (*telegram.constants.DiceEmoji attribute*), 974
BASKETBALL (*telegram.Dice attribute*), 374
BASKETBALL (*telegram.ext.filters.Dice attribute*), 898
BIG (*telegram.constants.ChatPhotoSize attribute*), 972
big_file_id (*telegram.ChatPhoto attribute*), 370
big_file_unique_id (*telegram.ChatPhoto attribute*), 370
bio (*telegram.ChatFullInfo attribute*), 310

bio (*telegram.ChatJoinRequest attribute*), 347
birth_date (*telegram.PersonalDetails attribute*), 798
Birthdate (*class in telegram*), 224
birthdate (*telegram.ChatFullInfo attribute*), 309
block (*telegram.ext.BaseHandler attribute*), 869
block (*telegram.ext.BusinessConnectionHandler attribute*), 871
block (*telegram.ext.BusinessMessagesDeletedHandler attribute*), 872
block (*telegram.ext.CallbackQueryHandler attribute*), 874
block (*telegram.ext.ChatBoostHandler attribute*), 875
block (*telegram.ext.ChatJoinRequestHandler attribute*), 877
block (*telegram.ext.ChatMemberHandler attribute*), 879
block (*telegram.ext.ChosenInlineResultHandler attribute*), 880
block (*telegram.ext.CommandHandler attribute*), 883
block (*telegram.ext.ConversationHandler attribute*), 886
block (*telegram.ext.Defaults property*), 847
block (*telegram.ext.InlineQueryHandler attribute*), 919
block (*telegram.ext.ManagedBotUpdatedHandler attribute*), 920
block (*telegram.ext.MessageHandler attribute*), 921
block (*telegram.ext.MessageReactionHandler attribute*), 923
block (*telegram.ext.PaidMediaPurchasedHandler attribute*), 925
block (*telegram.ext.PollAnswerHandler attribute*), 926
block (*telegram.ext.PollHandler attribute*), 927
block (*telegram.ext.PreCheckoutQueryHandler attribute*), 929
block (*telegram.ext.PrefixHandler attribute*), 931
block (*telegram.ext.ShippingQueryHandler attribute*), 932
block (*telegram.ext.StringCommandHandler attribute*), 934
block (*telegram.ext.StringRegexHandler attribute*), 935
block (*telegram.ext.TypeHandler attribute*), 937
BLOCKQUOTE (*telegram.constants.MessageEntityType attribute*), 996
BLOCKQUOTE (*telegram.MessageEntity attribute*), 533
BLUE (*telegram.constants.ForumIconColor attribute*), 977
BLUE (*telegram.constants.KeyboardButtonStyle attribute*), 988
BOLD (*telegram.constants.MessageEntityType attribute*), 996
BOLD (*telegram.MessageEntity attribute*), 533
boost (*telegram.ChatBoostUpdated attribute*), 303
BOOST_ADDED (*in module telegram.ext.filters*), 889
BOOST_ADDED (*telegram.constants.MessageType attribute*), 1000
boost_added (*telegram.Message attribute*), 487
boost_count (*telegram.ChatBoostAdded attribute*), 299
boost_id (*telegram.ChatBoost attribute*), 298
boost_id (*telegram.ChatBoostRemoved attribute*), 300
boosts (*telegram.UserChatBoosts attribute*), 660
Bot (*class in telegram*), 13
bot (*telegram.Bot property*), 31
bot (*telegram.ext.Application attribute*), 804
bot (*telegram.ext.BasePersistence attribute*), 939
bot (*telegram.ext.CallbackContext property*), 841
bot (*telegram.ext.CallbackDataCache attribute*), 953
bot (*telegram.ext.Updater attribute*), 863
bot (*telegram.ManagedBotCreated attribute*), 460
bot (*telegram.ManagedBotUpdated attribute*), 461
bot() (*telegram.ext.ApplicationBuilder method*), 822
bot_administrator_rights (*telegram.KeyboardButtonRequestChat attribute*), 451
BOT_API_VERSION (*in module telegram.constants*), 962
BOT_API_VERSION_INFO (*in module telegram.constants*), 962
BOT_COMMAND (*telegram.constants.MessageEntityType attribute*), 996
BOT_COMMAND (*telegram.MessageEntity attribute*), 533
bot_data (*telegram.ext.Application attribute*), 805
bot_data (*telegram.ext.CallbackContext property*), 841
bot_data (*telegram.ext.ContextTypes property*), 845
bot_data (*telegram.ext.DictPersistence property*), 944
bot_data (*telegram.ext.PersistenceInput attribute*), 948
bot_data_json (*telegram.ext.DictPersistence property*), 944
bot_ids (*telegram.ext.filters.ViaBot property*), 917
bot_is_member (*telegram.KeyboardButtonRequestChat attribute*), 451
bot_username (*telegram.LoginUrl attribute*), 460
BotAccessSettings (*class in telegram*), 225
BotCommand (*class in telegram*), 225
BotCommandLimit (*class in telegram.constants*), 964
BotCommandScope (*class in telegram*), 226
BotCommandScopeAllChatAdministrators (*class in telegram*), 228
BotCommandScopeAllGroupChats (*class in telegram*), 228
BotCommandScopeAllPrivateChats (*class in telegram*), 228
BotCommandScopeChat (*class in telegram*), 229
BotCommandScopeChatAdministrators (*class in telegram*), 229
BotCommandScopeChatMember (*class in telegram*), 230
BotCommandScopeDefault (*class in telegram*), 231
BotCommandScopeType (*class in telegram.constants*), 965
BotDescription (*class in telegram*), 231

BotDescriptionLimit (*class in telegram.constants*), 966
BotName (*class in telegram*), 232
BotNameLimit (*class in telegram.constants*), 966
BotShortDescription (*class in telegram*), 232
BOTTLE_WITH_POPPING_CORK (*telegram.constants.ReactionEmoji attribute*), 1015
bottom_color (*telegram.BackgroundFillGradient attribute*), 297
BOWLING (*telegram.constants.DiceEmoji attribute*), 974
BOWLING (*telegram.Dice attribute*), 374
BOWLING (*telegram.ext.filters.Dice attribute*), 898
BROKEN_HEART (*telegram.constants.ReactionEmoji attribute*), 1015
build() (*telegram.ext.ApplicationBuilder method*), 822
build_reply_arguments() (*telegram.Message method*), 491
builder() (*telegram.ext.Application static method*), 809
BulkRequestLimit (*class in telegram.constants*), 966
BUSINESS_ACCOUNT_TRANSFER (*telegram.constants.TransactionPartnerUser attribute*), 1031
BUSINESS_CONNECTION (*telegram.constants.UpdateType attribute*), 1032
BUSINESS_CONNECTION (*telegram.Update attribute*), 628
business_connection (*telegram.Update attribute*), 627
business_connection_id (*telegram.BusinessMessagesDeleted attribute*), 240
BUSINESS_CONNECTION_ID (*telegram.constants.MessageType attribute*), 1000
business_connection_id (*telegram.Message attribute*), 487
business_intro (*telegram.ChatFullInfo attribute*), 309
business_location (*telegram.ChatFullInfo attribute*), 309
BUSINESS_MESSAGE (*telegram.constants.UpdateType attribute*), 1032
BUSINESS_MESSAGE (*telegram.ext.filters.UpdateType attribute*), 914
BUSINESS_MESSAGE (*telegram.Update attribute*), 628
business_message (*telegram.Update attribute*), 627
BUSINESS_MESSAGES (*telegram.ext.filters.UpdateType attribute*), 914
business_opening_hours (*telegram.ChatFullInfo attribute*), 309
BusinessBotRights (*class in telegram*), 232
BusinessConnection (*class in telegram*), 235
BusinessConnectionHandler (*class in telegram.ext*), 870
BusinessIntro (*class in telegram*), 236
BusinessLimit (*class in telegram.constants*), 967
BusinessLocation (*class in telegram*), 237
BusinessMessagesDeleted (*class in telegram*), 240
BusinessMessagesDeletedHandler (*class in telegram.ext*), 871
BusinessOpeningHours (*class in telegram*), 237
BusinessOpeningHoursInterval (*class in telegram*), 239
button_text (*telegram.WebAppData attribute*), 674
BUTTONS_PER_ROW (*telegram.constants.InlineKeyboardMarkupLimit attribute*), 979

C

callback (*telegram.ext.BaseHandler attribute*), 869
callback (*telegram.ext.BusinessConnectionHandler attribute*), 870
callback (*telegram.ext.BusinessMessagesDeletedHandler attribute*), 872
callback (*telegram.ext.CallbackQueryHandler attribute*), 874
callback (*telegram.ext.ChatBoostHandler attribute*), 875
callback (*telegram.ext.ChatJoinRequestHandler attribute*), 877
callback (*telegram.ext.ChatMemberHandler attribute*), 879
callback (*telegram.ext.ChosenInlineResultHandler attribute*), 880
callback (*telegram.ext.CommandHandler attribute*), 883
callback (*telegram.ext.InlineQueryHandler attribute*), 918
callback (*telegram.ext.Job attribute*), 852
callback (*telegram.ext.ManagedBotUpdatedHandler attribute*), 920
callback (*telegram.ext.MessageHandler attribute*), 921
callback (*telegram.ext.MessageReactionHandler attribute*), 923
callback (*telegram.ext.PaidMediaPurchasedHandler attribute*), 925
callback (*telegram.ext.PollAnswerHandler attribute*), 926
callback (*telegram.ext.PollHandler attribute*), 927
callback (*telegram.ext.PreCheckoutQueryHandler attribute*), 929
callback (*telegram.ext.PrefixHandler attribute*), 931
callback (*telegram.ext.ShippingQueryHandler attribute*), 932
callback (*telegram.ext.StringCommandHandler attribute*), 934
callback (*telegram.ext.StringRegexHandler attribute*), 935
callback (*telegram.ext.TypeHandler attribute*), 937
callback_data (*telegram.ext.DictPersistence property*), 944

callback_data (*telegram.ext.InvalidCallbackData attribute*), 956
callback_data (*telegram.ext.PersistenceInput attribute*), 948
callback_data (*telegram.InlineKeyboardButton attribute*), 404
callback_data_cache (*telegram.ext.ExtBot property*), 850
callback_data_json (*telegram.ext.DictPersistence property*), 944
callback_game (*telegram.InlineKeyboardButton attribute*), 405
CALLBACK_QUERY (*telegram.constants.UpdateType attribute*), 1032
CALLBACK_QUERY (*telegram.Update attribute*), 628
callback_query (*telegram.Update attribute*), 625
CallbackContext (*class in telegram.ext*), 839
CallbackDataCache (*class in telegram.ext*), 952
CallbackGame (*class in telegram*), 777
CallbackQuery (*class in telegram*), 241
CallbackQueryHandler (*class in telegram.ext*), 872
CallbackQueryLimit (*class in telegram.constants*), 968
can_add_web_page_previews (*telegram.ChatMemberRestricted attribute*), 361
can_add_web_page_previews (*telegram.ChatPermissions attribute*), 368
can_be_edited (*telegram.ChatMemberAdministrator attribute*), 353
can_be_transferred (*telegram.OwnedGiftUnique attribute*), 550
can_be_upgraded (*telegram.GiftInfo attribute*), 393
can_be_upgraded (*telegram.OwnedGiftRegular attribute*), 547
can_change_gift_settings (*telegram.BusinessBotRights attribute*), 234
can_change_info (*telegram.ChatAdministratorRights attribute*), 283
can_change_info (*telegram.ChatMemberAdministrator attribute*), 353
can_change_info (*telegram.ChatMemberRestricted attribute*), 360
can_change_info (*telegram.ChatPermissions attribute*), 368
can_connect_to_business (*telegram.User attribute*), 636
can_convert_gifts_to_stars (*telegram.BusinessBotRights attribute*), 234
can_delete_all_messages (*telegram.BusinessBotRights attribute*), 234
can_delete_messages (*telegram.ChatAdministratorRights attribute*), 282
can_delete_messages (*telegram.ChatMemberAdministrator attribute*), 353
can_delete_sent_messages (*telegram.BusinessBotRights attribute*), 234
can_delete_stories (*telegram.ChatAdministratorRights attribute*), 284
can_delete_stories (*telegram.ChatMemberAdministrator attribute*), 354
can_edit_bio (*telegram.BusinessBotRights attribute*), 234
can_edit_messages (*telegram.ChatAdministratorRights attribute*), 283
can_edit_messages (*telegram.ChatMemberAdministrator attribute*), 354
can_edit_name (*telegram.BusinessBotRights attribute*), 234
can_edit_profile_photo (*telegram.BusinessBotRights attribute*), 234
can_edit_stories (*telegram.ChatAdministratorRights attribute*), 284
can_edit_stories (*telegram.ChatMemberAdministrator attribute*), 354
can_edit_tag (*telegram.ChatMemberRestricted attribute*), 362
can_edit_tag (*telegram.ChatPermissions attribute*), 369
can_edit_username (*telegram.BusinessBotRights attribute*), 234
can_invite_users (*telegram.ChatAdministratorRights attribute*), 283
can_invite_users (*telegram.ChatMemberAdministrator attribute*), 354
can_invite_users (*telegram.ChatMemberRestricted attribute*), 360
can_invite_users (*telegram.ChatPermissions attribute*), 368
can_join_groups (*telegram.Bot property*), 32
can_join_groups (*telegram.User attribute*), 635
can_manage_bots (*telegram.User attribute*), 636
can_manage_chat (*telegram.ChatAdministratorRights attribute*), 282
can_manage_chat (*telegram.ChatMemberAdministrator attribute*), 353
can_manage_direct_messages (*telegram.ChatAdministratorRights attribute*), 284
can_manage_direct_messages (*telegram.ChatMemberAdministrator attribute*), 355

can_manage_stories (*telegram.BusinessBotRights attribute*), 235
can_manage_tags (*telegram.ChatAdministratorRights attribute*), 284
can_manage_tags (*telegram.ChatMemberAdministrator attribute*), 355
can_manage_topics (*telegram.ChatAdministratorRights attribute*), 284
can_manage_topics (*telegram.ChatMemberAdministrator attribute*), 355
can_manage_topics (*telegram.ChatMemberRestricted attribute*), 361
can_manage_topics (*telegram.ChatPermissions attribute*), 368
can_manage_video_chats (*telegram.ChatAdministratorRights attribute*), 283
can_manage_video_chats (*telegram.ChatMemberAdministrator attribute*), 353
can_pin_messages (*telegram.ChatAdministratorRights attribute*), 283
can_pin_messages (*telegram.ChatMemberAdministrator attribute*), 354
can_pin_messages (*telegram.ChatMemberRestricted attribute*), 360
can_pin_messages (*telegram.ChatPermissions attribute*), 368
can_post_messages (*telegram.ChatAdministratorRights attribute*), 283
can_post_messages (*telegram.ChatMemberAdministrator attribute*), 354
can_post_stories (*telegram.ChatAdministratorRights attribute*), 283
can_post_stories (*telegram.ChatMemberAdministrator attribute*), 354
can_promote_members (*telegram.ChatAdministratorRights attribute*), 283
can_promote_members (*telegram.ChatMemberAdministrator attribute*), 353
can_react_to_messages (*telegram.ChatMemberRestricted attribute*), 362
can_react_to_messages (*telegram.ChatPermissions attribute*), 369
can_read_all_group_messages (*telegram.Bot property*), 32
can_read_all_group_messages (*telegram.User attribute*), 635
can_read_messages (*telegram.BusinessBotRights attribute*), 234
can_reply (*telegram.BusinessBotRights attribute*), 233
can_restrict_members (*telegram.ChatAdministratorRights attribute*), 283
can_restrict_members (*telegram.ChatMemberAdministrator attribute*), 353
can_send_audios (*telegram.ChatMemberRestricted attribute*), 361
can_send_audios (*telegram.ChatPermissions attribute*), 368
can_send_documents (*telegram.ChatMemberRestricted attribute*), 361
can_send_documents (*telegram.ChatPermissions attribute*), 368
can_send_messages (*telegram.ChatMemberRestricted attribute*), 360
can_send_messages (*telegram.ChatPermissions attribute*), 367
can_send_other_messages (*telegram.ChatMemberRestricted attribute*), 361
can_send_other_messages (*telegram.ChatPermissions attribute*), 367
can_send_paid_media (*telegram.ChatFullInfo attribute*), 313
can_send_photos (*telegram.ChatMemberRestricted attribute*), 361
can_send_photos (*telegram.ChatPermissions attribute*), 368
can_send_polls (*telegram.ChatMemberRestricted attribute*), 360
can_send_polls (*telegram.ChatPermissions attribute*), 367
can_send_video_notes (*telegram.ChatMemberRestricted attribute*), 361
can_send_video_notes (*telegram.ChatPermissions attribute*), 368
can_send_videos (*telegram.ChatMemberRestricted attribute*), 361
can_send_videos (*telegram.ChatPermissions attribute*), 368
can_send_voice_notes (*telegram.ChatMemberRestricted attribute*), 362
can_send_voice_notes (*telegram.ChatPermissions attribute*), 369
can_set_sticker_set (*telegram.ChatFullInfo*

attribute), 312
can_transfer_and_upgrade_gifts (*telegram.BusinessBotRights attribute*), 235
can_transfer_stars (*telegram.BusinessBotRights attribute*), 235
can_view_gifts_and_stars (*telegram.BusinessBotRights attribute*), 234
Caption (*class in telegram.ext.filters*), 894
CAPTION (*in module telegram.ext.filters*), 889
caption (*telegram.InlineQueryResultAudio attribute*), 698
caption (*telegram.InlineQueryResultCachedAudio attribute*), 700
caption (*telegram.InlineQueryResultCachedDocument attribute*), 701
caption (*telegram.InlineQueryResultCachedGif attribute*), 703
caption (*telegram.InlineQueryResultCachedMpeg4Gif attribute*), 705
caption (*telegram.InlineQueryResultCachedPhoto attribute*), 708
caption (*telegram.InlineQueryResultCachedVideo attribute*), 711
caption (*telegram.InlineQueryResultCachedVoice attribute*), 713
caption (*telegram.InlineQueryResultDocument attribute*), 717
caption (*telegram.InlineQueryResultGif attribute*), 721
caption (*telegram.InlineQueryResultMpeg4Gif attribute*), 727
caption (*telegram.InlineQueryResultPhoto attribute*), 730
caption (*telegram.InlineQueryResultVideo attribute*), 736
caption (*telegram.InlineQueryResultVoice attribute*), 739
caption (*telegram.InputMedia attribute*), 416
caption (*telegram.InputMediaAnimation attribute*), 418
caption (*telegram.InputMediaAudio attribute*), 421
caption (*telegram.InputMediaDocument attribute*), 423
caption (*telegram.InputMediaLivePhoto attribute*), 425
caption (*telegram.InputMediaPhoto attribute*), 428
caption (*telegram.InputMediaVideo attribute*), 433
caption (*telegram.Message attribute*), 480
caption_entities (*telegram.InlineQueryResultAudio attribute*), 698
caption_entities (*telegram.InlineQueryResultCachedAudio attribute*), 700
caption_entities (*telegram.InlineQueryResultCachedDocument attribute*), 702
caption_entities (*telegram.InlineQueryResultCachedGif attribute*), 704
caption_entities (*telegram.InlineQueryResultCachedMpeg4Gif attribute*), 706
caption_entities (*telegram.InlineQueryResultCachedPhoto attribute*), 708
caption_entities (*telegram.InlineQueryResultCachedVideo attribute*), 711
caption_entities (*telegram.InlineQueryResultCachedVoice attribute*), 713
caption_entities (*telegram.InlineQueryResultDocument attribute*), 717
caption_entities (*telegram.InlineQueryResultGif attribute*), 721
caption_entities (*telegram.InlineQueryResultMpeg4Gif attribute*), 727
caption_entities (*telegram.InlineQueryResultPhoto attribute*), 730
caption_entities (*telegram.InlineQueryResultVideo attribute*), 736
caption_entities (*telegram.InlineQueryResultVoice attribute*), 739
caption_entities (*telegram.InputMedia attribute*), 416
caption_entities (*telegram.InputMediaAnimation attribute*), 418
caption_entities (*telegram.InputMediaAudio attribute*), 421
caption_entities (*telegram.InputMediaDocument attribute*), 423
caption_entities (*telegram.InputMediaLivePhoto attribute*), 425
caption_entities (*telegram.InputMediaPhoto attribute*), 428
caption_entities (*telegram.InputMediaVideo attribute*), 433
caption_entities (*telegram.Message attribute*), 478
caption_html (*telegram.Message property*), 492
caption_html_urled (*telegram.Message property*), 492
CAPTION_LENGTH (*telegram.constants.MessageLimit attribute*), 998
CAPTION_LENGTH (*telegram.constants.StoryLimit attribute*), 1028
caption_markdown (*telegram.Message property*), 493
caption_markdown_urled (*telegram.Message property*), 493
caption_markdown_v2 (*telegram.Message property*), 494

caption_markdown_v2_urled (*telegram.Message property*), 494
CaptionEntity (*class in telegram.ext.filters*), 894
CaptionRegex (*class in telegram.ext.filters*), 894
CASHTAG (*telegram.constants.MessageEntityType attribute*), 996
CASHTAG (*telegram.MessageEntity attribute*), 533
center_color (*telegram.GiftBackground attribute*), 391
center_color (*telegram.UniqueGiftBackdropColors attribute*), 617
CHANNEL (*telegram.Chat attribute*), 250
CHANNEL (*telegram.ChatFullInfo attribute*), 314
CHANNEL (*telegram.constants.ChatType attribute*), 973
CHANNEL (*telegram.constants.MessageOriginType attribute*), 999
CHANNEL (*telegram.ext.filters.ChatType attribute*), 896
CHANNEL (*telegram.ext.filters.SenderChat attribute*), 907
CHANNEL (*telegram.MessageOrigin attribute*), 538
CHANNEL_CHAT_CREATED (*telegram.constants.MessageType attribute*), 1000
channel_chat_created (*telegram.Message attribute*), 481
CHANNEL_POST (*telegram.constants.UpdateType attribute*), 1033
CHANNEL_POST (*telegram.ext.filters.UpdateType attribute*), 913
CHANNEL_POST (*telegram.Update attribute*), 628
channel_post (*telegram.Update attribute*), 625
CHANNEL_POSTS (*telegram.ext.filters.UpdateType attribute*), 913
Chat (*class in telegram*), 248
Chat (*class in telegram.ext.filters*), 895
CHAT (*telegram.BotCommandScope attribute*), 227
chat (*telegram.BusinessMessagesDeleted attribute*), 240
chat (*telegram.ChatBoostRemoved attribute*), 299
chat (*telegram.ChatBoostUpdated attribute*), 303
chat (*telegram.ChatJoinRequest attribute*), 347
chat (*telegram.ChatMemberUpdated attribute*), 363
CHAT (*telegram.constants.BotCommandScopeType attribute*), 965
CHAT (*telegram.constants.MessageOriginType attribute*), 999
CHAT (*telegram.constants.TransactionPartnerType attribute*), 1030
chat (*telegram.ExternalReplyInfo attribute*), 380
chat (*telegram.GiveawayWinners attribute*), 398
chat (*telegram.InaccessibleMessage attribute*), 400
chat (*telegram.MaybeInaccessibleMessage attribute*), 462
chat (*telegram.Message attribute*), 476
CHAT (*telegram.MessageOrigin attribute*), 538
chat (*telegram.MessageOriginChannel attribute*), 539
chat (*telegram.MessageReactionCountUpdated attribute*), 542
chat (*telegram.MessageReactionUpdated attribute*), 543
chat (*telegram.Story attribute*), 593
CHAT (*telegram.TransactionPartner attribute*), 770
chat (*telegram.TransactionPartnerChat attribute*), 772
CHAT_ACTIVITY_TIMEOUT (*telegram.constants.BusinessLimit attribute*), 967
CHAT_ADMINISTRATOR_CUSTOM_TITLE_LENGTH (*telegram.constants.ChatLimit attribute*), 971
CHAT_ADMINISTRATORS (*telegram.BotCommandScope attribute*), 227
CHAT_ADMINISTRATORS (*telegram.constants.BotCommandScopeType attribute*), 965
CHAT_BACKGROUND_SET (*telegram.constants.MessageType attribute*), 1000
CHAT_BACKGROUND_SET (*telegram.ext.filters.StatusUpdate attribute*), 908
chat_background_set (*telegram.Message attribute*), 488
CHAT_BOOST (*telegram.constants.UpdateType attribute*), 1033
CHAT_BOOST (*telegram.ext.ChatBoostHandler attribute*), 876
CHAT_BOOST (*telegram.Update attribute*), 628
chat_boost (*telegram.Update attribute*), 626
chat_boost_types (*telegram.ext.ChatBoostHandler attribute*), 875
CHAT_CREATED (*telegram.ext.filters.StatusUpdate attribute*), 908
chat_data (*telegram.ext.Application attribute*), 805
chat_data (*telegram.ext.CallbackContext property*), 841
chat_data (*telegram.ext.ContextTypes property*), 845
chat_data (*telegram.ext.DictPersistence property*), 944
chat_data (*telegram.ext.PersistenceInput attribute*), 948
chat_data_json (*telegram.ext.DictPersistence property*), 945
CHAT_DESCRIPTION_LENGTH (*telegram.constants.ChatLimit attribute*), 971
chat_has_username (*telegram.KeyboardButtonRequestChat attribute*), 450
chat_id (*telegram.BotCommandScopeChat attribute*), 229
chat_id (*telegram.BotCommandScopeChatAdministrators attribute*), 230
chat_id (*telegram.BotCommandScopeChatMember attribute*), 230
chat_id (*telegram.ChatShared attribute*), 371
chat_id (*telegram.ext.Job attribute*), 853
chat_id (*telegram.Message property*), 495
chat_id (*telegram.ReplyParameters attribute*), 589

chat_ids (*telegram.ext.filters.Chat attribute*), 895
chat_ids (*telegram.ext.filters.ForwardedFrom attribute*), 903
chat_ids (*telegram.ext.filters.SenderChat attribute*), 906
chat_instance (*telegram.CallbackQuery attribute*), 242
chat_is_channel (*telegram.KeyboardButtonRequestChat attribute*), 450
chat_is_created (*telegram.KeyboardButtonRequestChat attribute*), 450
chat_is_forum (*telegram.KeyboardButtonRequestChat attribute*), 450
CHAT_JOIN_REQUEST (*telegram.constants.UpdateType attribute*), 1033
CHAT_JOIN_REQUEST (*telegram.Update attribute*), 628
chat_join_request (*telegram.Update attribute*), 626
CHAT_MEMBER (*telegram.BotCommandScope attribute*), 227
CHAT_MEMBER (*telegram.constants.BotCommandScopeType attribute*), 966
CHAT_MEMBER (*telegram.constants.UpdateType attribute*), 1033
CHAT_MEMBER (*telegram.ext.ChatMemberHandler attribute*), 879
CHAT_MEMBER (*telegram.Update attribute*), 628
chat_member (*telegram.Update attribute*), 626
chat_member_types (*telegram.ext.ChatMemberHandler attribute*), 879
CHAT_OWNER_CHANGED (*telegram.constants.MessageType attribute*), 1000
CHAT_OWNER_CHANGED (*telegram.ext.filters.StatusUpdate attribute*), 908
chat_owner_changed (*telegram.Message attribute*), 489
CHAT_OWNER_LEFT (*telegram.constants.MessageType attribute*), 1001
CHAT_OWNER_LEFT (*telegram.ext.filters.StatusUpdate attribute*), 908
chat_owner_left (*telegram.Message attribute*), 489
CHAT_SHARED (*telegram.constants.MessageType attribute*), 1001
CHAT_SHARED (*telegram.ext.filters.StatusUpdate attribute*), 908
chat_shared (*telegram.Message attribute*), 485
CHAT_THEME (*telegram.BackgroundType attribute*), 291
CHAT_THEME (*telegram.constants.BackgroundTypeType attribute*), 963
chat_type (*telegram.InlineQuery attribute*), 692
chat_types (*telegram.ext.InlineQueryHandler attribute*), 919
ChatAction (*class in telegram.constants*), 968
ChatAdministratorRights (*class in telegram*), 280
ChatBackground (*class in telegram*), 285
ChatBoost (*class in telegram*), 298
ChatBoostAdded (*class in telegram*), 299
ChatBoostHandler (*class in telegram.ext*), 874
ChatBoostRemoved (*class in telegram*), 299
ChatBoostSource (*class in telegram*), 300
ChatBoostSourceGiftCode (*class in telegram*), 301
ChatBoostSourceGiveaway (*class in telegram*), 301
ChatBoostSourcePremium (*class in telegram*), 302
ChatBoostSources (*class in telegram.constants*), 969
ChatBoostUpdated (*class in telegram*), 303
ChatFullInfo (*class in telegram*), 304
ChatID (*class in telegram.constants*), 970
ChatInviteLink (*class in telegram*), 343
ChatInviteLinkLimit (*class in telegram.constants*), 970
ChatJoinRequest (*class in telegram*), 346
ChatJoinRequestHandler (*class in telegram.ext*), 876
ChatLimit (*class in telegram.constants*), 971
ChatLocation (*class in telegram*), 349
ChatMember (*class in telegram*), 349
ChatMemberAdministrator (*class in telegram*), 351
ChatMemberBanned (*class in telegram*), 355
ChatMemberHandler (*class in telegram.ext*), 878
ChatMemberLeft (*class in telegram*), 356
ChatMemberMember (*class in telegram*), 357
ChatMemberOwner (*class in telegram*), 358
ChatMemberRestricted (*class in telegram*), 358
ChatMemberStatus (*class in telegram.constants*), 971
ChatMemberUpdated (*class in telegram*), 362
ChatMigrated, 1036
ChatOwnerChanged (*class in telegram*), 365
ChatOwnerLeft (*class in telegram*), 365
ChatPermissions (*class in telegram*), 366
ChatPhoto (*class in telegram*), 369
ChatPhotoSize (*class in telegram.constants*), 972
chats (*telegram.Giveaway attribute*), 395
ChatShared (*class in telegram*), 371
ChatSubscriptionLimit (*class in telegram.constants*), 972
ChatType (*class in telegram.constants*), 973
ChatType (*class in telegram.ext.filters*), 896
check_update() (*telegram.ext.BaseHandler method*), 869
check_update() (*telegram.ext.BusinessConnectionHandler method*), 871
check_update() (*telegram.ext.BusinessMessagesDeletedHandler method*), 872
check_update() (*telegram.ext.CallbackQueryHandler method*), 874
check_update() (*telegram.ext.ChatBoostHandler method*), 876

check_update() (*telegram.ext.ChatJoinRequestHandler method*), 877
check_update() (*telegram.ext.ChatMemberHandler method*), 879
check_update() (*telegram.ext.ChosenInlineResultHandler method*), 881
check_update() (*telegram.ext.CommandHandler method*), 883
check_update() (*telegram.ext.ConversationHandler method*), 887
check_update() (*telegram.ext.filters.BaseFilter method*), 893
check_update() (*telegram.ext.filters.MessageFilter method*), 904
check_update() (*telegram.ext.filters.UpdateFilter method*), 913
check_update() (*telegram.ext.InlineQueryHandler method*), 919
check_update() (*telegram.ext.ManagedBotUpdatedHandler method*), 920
check_update() (*telegram.ext.MessageHandler method*), 921
check_update() (*telegram.ext.MessageReactionHandler method*), 924
check_update() (*telegram.ext.PaidMediaPurchasedHandler method*), 925
check_update() (*telegram.ext.PollAnswerHandler method*), 926
check_update() (*telegram.ext.PollHandler method*), 927
check_update() (*telegram.ext.PreCheckoutQueryHandler method*), 929
check_update() (*telegram.ext.PrefixHandler method*), 931
check_update() (*telegram.ext.ShippingQueryHandler method*), 932
check_update() (*telegram.ext.StringCommandHandler method*), 934
check_update() (*telegram.ext.StringRegexHandler method*), 936
check_update() (*telegram.ext.TypeHandler method*), 937
Checklist (*class in telegram*), 285
CHECKLIST (*in module telegram.ext.filters*), 889
CHECKLIST (*telegram.constants.MessageType attribute*), 1001
checklist (*telegram.ExternalReplyInfo attribute*), 381
checklist (*telegram.Message attribute*), 485
checklist_message (*telegram.ChecklistTasksAdded attribute*), 289
checklist_message (*telegram.ChecklistTasksDone attribute*), 290
checklist_task_id (*telegram.ReplyParameters attribute*), 589
CHECKLIST_TASKS_ADDED (*telegram.constants.MessageType attribute*), 1001
CHECKLIST_TASKS_ADDED (*telegram.ext.filters.StatusUpdate attribute*), 908
checklist_tasks_added (*telegram.Message attribute*), 488
CHECKLIST_TASKS_DONE (*telegram.constants.MessageType attribute*), 1001
CHECKLIST_TASKS_DONE (*telegram.ext.filters.StatusUpdate attribute*), 908
checklist_tasks_done (*telegram.Message attribute*), 488
ChecklistTask (*class in telegram*), 287
ChecklistTasksAdded (*class in telegram*), 289
ChecklistTasksDone (*class in telegram*), 290
CHIN (*telegram.constants.MaskPosition attribute*), 991
CHIN (*telegram.MaskPosition attribute*), 683
CHOOSE_STICKER (*telegram.constants.ChatAction attribute*), 968
CHOSEN_INLINE_RESULT (*telegram.constants.UpdateType attribute*), 1033
CHOSEN_INLINE_RESULT (*telegram.Update attribute*), 628
chosen_inline_result (*telegram.Update attribute*), 625
ChosenInlineResult (*class in telegram*), 689
ChosenInlineResultHandler (*class in telegram.ext*), 879
CHRISTMAS_TREE (*telegram.constants.ReactionEmoji attribute*), 1016
city (*telegram.LocationAddress attribute*), 459
city (*telegram.ResidentialAddress attribute*), 799
city (*telegram.ShippingAddress attribute*), 762
CLAPPING_HANDS (*telegram.constants.ReactionEmoji attribute*), 1016
clear_callback_data() (*telegram.ext.CallbackDataCache method*), 953
clear_callback_queries() (*telegram.ext.CallbackDataCache method*), 953
close() (*telegram.Bot method*), 32
close_date (*telegram.Poll attribute*), 562
close_forum_topic() (*telegram.Bot method*), 33
close_forum_topic() (*telegram.Chat method*), 252
close_forum_topic() (*telegram.ChatFullInfo method*), 315
close_forum_topic() (*telegram.Message method*), 495

close_general_forum_topic() (*telegram.Bot method*), 33
close_general_forum_topic() (*telegram.Chat method*), 252
close_general_forum_topic() (*telegram.ChatFullInfo method*), 315
closeForumTopic() (*telegram.Bot method*), 32
closeGeneralForumTopic() (*telegram.Bot method*), 32
closing_minute (*telegram.BusinessOpeningHoursInterval attribute*), 239
closing_time (*telegram.BusinessOpeningHoursInterval property*), 240
CLOWN_FACE (*telegram.constants.ReactionEmoji attribute*), 1016
CODE (*telegram.constants.MessageEntityType attribute*), 996
CODE (*telegram.MessageEntity attribute*), 533
codec (*telegram.VideoQuality attribute*), 672
collect_additional_context() (*telegram.ext.BaseHandler method*), 869
collect_additional_context() (*telegram.ext.CallbackQueryHandler method*), 874
collect_additional_context() (*telegram.ext.ChosenInlineResultHandler method*), 881
collect_additional_context() (*telegram.ext.CommandHandler method*), 883
collect_additional_context() (*telegram.ext.InlineQueryHandler method*), 919
collect_additional_context() (*telegram.ext.MessageHandler method*), 922
collect_additional_context() (*telegram.ext.PrefixHandler method*), 931
collect_additional_context() (*telegram.ext.StringCommandHandler method*), 934
collect_additional_context() (*telegram.ext.StringRegexHandler method*), 936
color (*telegram.BackgroundFillSolid attribute*), 296
COLOR_000 (*telegram.constants.AccentColor attribute*), 960
COLOR_000 (*telegram.constants.ProfileAccentColor attribute*), 1014
COLOR_001 (*telegram.constants.AccentColor attribute*), 960
COLOR_001 (*telegram.constants.ProfileAccentColor attribute*), 1014
COLOR_002 (*telegram.constants.AccentColor attribute*), 960
COLOR_002 (*telegram.constants.ProfileAccentColor attribute*), 1014
COLOR_003 (*telegram.constants.AccentColor attribute*), 960
COLOR_003 (*telegram.constants.ProfileAccentColor attribute*), 1014
COLOR_004 (*telegram.constants.AccentColor attribute*), 960
COLOR_004 (*telegram.constants.ProfileAccentColor attribute*), 1014
COLOR_005 (*telegram.constants.AccentColor attribute*), 960
COLOR_005 (*telegram.constants.ProfileAccentColor attribute*), 1014
COLOR_006 (*telegram.constants.AccentColor attribute*), 960
COLOR_006 (*telegram.constants.ProfileAccentColor attribute*), 1014
COLOR_007 (*telegram.constants.AccentColor attribute*), 961
COLOR_007 (*telegram.constants.ProfileAccentColor attribute*), 1014
COLOR_008 (*telegram.constants.AccentColor attribute*), 961
COLOR_008 (*telegram.constants.ProfileAccentColor attribute*), 1014
COLOR_009 (*telegram.constants.AccentColor attribute*), 961
COLOR_009 (*telegram.constants.ProfileAccentColor attribute*), 1015
COLOR_010 (*telegram.constants.AccentColor attribute*), 961
COLOR_010 (*telegram.constants.ProfileAccentColor attribute*), 1015
COLOR_011 (*telegram.constants.AccentColor attribute*), 961
COLOR_011 (*telegram.constants.ProfileAccentColor attribute*), 1015
COLOR_012 (*telegram.constants.AccentColor attribute*), 961
COLOR_012 (*telegram.constants.ProfileAccentColor attribute*), 1015
COLOR_013 (*telegram.constants.AccentColor attribute*), 961
COLOR_013 (*telegram.constants.ProfileAccentColor attribute*), 1015
COLOR_014 (*telegram.constants.AccentColor attribute*), 961
COLOR_014 (*telegram.constants.ProfileAccentColor attribute*), 1015
COLOR_015 (*telegram.constants.AccentColor attribute*), 961
COLOR_015 (*telegram.constants.ProfileAccentColor attribute*), 1015
COLOR_016 (*telegram.constants.AccentColor attribute*), 961
COLOR_017 (*telegram.constants.AccentColor attribute*), 961
COLOR_018 (*telegram.constants.AccentColor attribute*), 961
COLOR_019 (*telegram.constants.AccentColor at-*

tribute), 961
COLOR_020 (*telegram.constants.AccentColor attribute*), 962
colors (*telegram.BackgroundFillFreeformGradient attribute*), 298
colors (*telegram.UniqueGift attribute*), 615
colors (*telegram.UniqueGiftBackdrop attribute*), 617
Command (*class in telegram.ext.filters*), 897
COMMAND (*in module telegram.ext.filters*), 889
command (*telegram.BotCommand attribute*), 226
command (*telegram.ext.StringCommandHandler attribute*), 934
CommandHandler (*class in telegram.ext*), 881
COMMANDS (*telegram.constants.MenuButtonType attribute*), 992
commands (*telegram.ext.CommandHandler attribute*), 883
commands (*telegram.ext.PrefixHandler attribute*), 931
COMMANDS (*telegram.MenuButton attribute*), 463
comment (*telegram.SuggestedPostDeclined attribute*), 601
commission_per_mille (*telegram.AffiliateInfo attribute*), 753
commission_per_mille (*telegram.TransactionPartnerAffiliateProgram attribute*), 771
completed_by_chat (*telegram.ChecklistTask attribute*), 288
completed_by_user (*telegram.ChecklistTask attribute*), 288
completion_date (*telegram.ChecklistTask attribute*), 288
compute_quote_position_and_entities() (*telegram.Message method*), 495
concatenate() (*telegram.MessageEntity class method*), 535
concurrent_updates (*telegram.ext.Application property*), 809
concurrent_updates() (*telegram.ext.ApplicationBuilder method*), 823
Conflict, 1037
connect_timeout() (*telegram.ext.ApplicationBuilder method*), 823
CONNECTED_WEBSITE (*telegram.constants.MessageType attribute*), 1001
CONNECTED_WEBSITE (*telegram.ext.filters.StatusUpdate attribute*), 908
connected_website (*telegram.Message attribute*), 482
connection_pool_size() (*telegram.ext.ApplicationBuilder method*), 824
Contact (*class in telegram*), 372
CONTACT (*in module telegram.ext.filters*), 889
CONTACT (*telegram.constants.InlineQueryResultType attribute*), 981
CONTACT (*telegram.constants.MessageAttachmentType attribute*), 992
CONTACT (*telegram.constants.MessageType attribute*), 1001
contact (*telegram.ExternalReplyInfo attribute*), 381
contact (*telegram.Message attribute*), 480
ContactLimit (*class in telegram.constants*), 974
contains_files (*telegram.request.RequestData attribute*), 1046
context (*telegram.ext.ContextTypes property*), 845
context_types (*telegram.ext.Application attribute*), 806
context_types (*telegram.ext.PicklePersistence attribute*), 950
context_types() (*telegram.ext.ApplicationBuilder method*), 824
ContextTypes (*class in telegram.ext*), 844
conversation_timeout (*telegram.ext.ConversationHandler property*), 887
ConversationHandler (*class in telegram.ext*), 883
conversations (*telegram.ext.DictPersistence property*), 945
conversations_json (*telegram.ext.DictPersistence property*), 945
convert_gift_to_stars() (*telegram.Bot method*), 34
convert_star_count (*telegram.GiftInfo attribute*), 392
convert_star_count (*telegram.OwnedGiftRegular attribute*), 547
convertGiftToStars() (*telegram.Bot method*), 34
copy() (*telegram.Message method*), 496
copy_message() (*telegram.Bot method*), 35
copy_message() (*telegram.CallbackQuery method*), 243
copy_message() (*telegram.Chat method*), 253
copy_message() (*telegram.ChatFullInfo method*), 316
copy_message() (*telegram.User method*), 637
copy_messages() (*telegram.Bot method*), 37
copy_messages() (*telegram.Chat method*), 253
copy_messages() (*telegram.ChatFullInfo method*), 316
copy_messages() (*telegram.User method*), 637
copy_text (*telegram.InlineKeyboardButton attribute*), 405
copyMessage() (*telegram.Bot method*), 35
copyMessages() (*telegram.Bot method*), 35
CopyTextButton (*class in telegram*), 290
corner_radius_percentage (*telegram.StoryAreaPosition attribute*), 595
coroutine (*telegram.ext.CallbackContext attribute*), 840
correct_option_id (*telegram.Poll property*), 564
correct_option_ids (*telegram.Poll attribute*), 563
country_code (*telegram.LocationAddress attribute*),

459
country_code (*telegram.PersonalDetails attribute*), 798
country_code (*telegram.ResidentialAddress attribute*), 799
country_code (*telegram.ShippingAddress attribute*), 762
country_codes (*telegram.Giveaway attribute*), 396
country_codes (*telegram.Poll attribute*), 563
cover (*telegram.InputMediaVideo attribute*), 434
cover (*telegram.InputPaidMediaVideo attribute*), 439
cover (*telegram.Video attribute*), 667
cover_frame_timestamp (*telegram.InputStoryContentVideo attribute*), 445
create_chat_invite_link() (*telegram.Bot method*), 39
create_chat_subscription_invite_link() (*telegram.Bot method*), 40
create_deep_linked_url() (*in module telegram.helpers*), 1040
create_forum_topic() (*telegram.Bot method*), 41
create_forum_topic() (*telegram.Chat method*), 253
create_forum_topic() (*telegram.ChatFullInfo method*), 317
create_invite_link() (*telegram.Chat method*), 254
create_invite_link() (*telegram.ChatFullInfo method*), 317
create_invoice_link() (*telegram.Bot method*), 42
create_new_sticker_set() (*telegram.Bot method*), 44
create_subscription_invite_link() (*telegram.Chat method*), 254
create_subscription_invite_link() (*telegram.ChatFullInfo method*), 317
create_task() (*telegram.ext.Application method*), 809
createChatInviteLink() (*telegram.Bot method*), 39
createChatSubscriptionInviteLink() (*telegram.Bot method*), 39
createForumTopic() (*telegram.Bot method*), 39
createInvoiceLink() (*telegram.Bot method*), 39
createNewStickerSet() (*telegram.Bot method*), 39
creates_join_request (*telegram.ChatInviteLink attribute*), 345
creator (*telegram.ChatInviteLink attribute*), 345
Credentials (*class in telegram*), 780
credentials (*telegram.PassportData attribute*), 787
CRYING_FACE (*telegram.constants.ReactionEmoji attribute*), 1016
currency (*telegram.InputInvoiceMessageContent attribute*), 750
currency (*telegram.Invoice attribute*), 754
currency (*telegram.PreCheckoutQuery attribute*), 758
currency (*telegram.RefundedPayment attribute*), 759
currency (*telegram.SuccessfulPayment attribute*), 768
currency (*telegram.SuggestedPostPaid attribute*), 603
currency (*telegram.SuggestedPostPrice attribute*), 605
current_concurrent_updates (*telegram.ext.BaseUpdateProcessor property*), 838
current_level_rating (*telegram.UserRating attribute*), 662
CUSTOM_EMOJI (*telegram.constants.MessageEntityType attribute*), 997
CUSTOM_EMOJI (*telegram.constants.ReactionType attribute*), 1022
CUSTOM_EMOJI (*telegram.constants.StickerType attribute*), 1026
CUSTOM_EMOJI (*telegram.MessageEntity attribute*), 533
CUSTOM_EMOJI (*telegram.ReactionType attribute*), 578
CUSTOM_EMOJI (*telegram.Sticker attribute*), 687
custom_emoji_id (*telegram.MessageEntity attribute*), 532
custom_emoji_id (*telegram.ReactionTypeCustomEmoji attribute*), 579
custom_emoji_id (*telegram.Sticker attribute*), 687
CUSTOM_EMOJI_IDENTIFIER_LIMIT (*telegram.constants.CustomEmojiStickerLimit attribute*), 974
custom_emoji_sticker_set_name (*telegram.ChatFullInfo attribute*), 312
custom_title (*telegram.ChatMemberAdministrator attribute*), 355
custom_title (*telegram.ChatMemberOwner attribute*), 358
CustomEmojiStickerLimit (*class in telegram.constants*), 974

D

DANGER (*telegram.constants.KeyboardButtonStyle attribute*), 988
dark_theme_dimming (*telegram.BackgroundTypeFill attribute*), 292
dark_theme_dimming (*telegram.BackgroundTypeWallpaper attribute*), 293
dark_theme_main_color (*telegram.UniqueGiftColors attribute*), 616
dark_theme_other_colors (*telegram.UniqueGiftColors attribute*), 616
DARTS (*telegram.constants.DiceEmoji attribute*), 974
DARTS (*telegram.Dice attribute*), 374
DARTS (*telegram.ext.filters.Dice attribute*), 898
data (*telegram.CallbackQuery attribute*), 242
data (*telegram.EncryptedCredentials attribute*), 781
data (*telegram.EncryptedPassportElement attribute*), 783
data (*telegram.ext.Job attribute*), 852
data (*telegram.PassportData attribute*), 786
data (*telegram.SecureValue attribute*), 802
data (*telegram.WebAppData attribute*), 674

data_filter (*telegram.ext.filters.BaseFilter property*), 893
data_hash (*telegram.PassportElementErrorDataField attribute*), 788
DataCredentials (*class in telegram*), 780
date (*telegram.BusinessConnection attribute*), 236
date (*telegram.ChatJoinRequest attribute*), 347
date (*telegram.ChatMemberUpdated attribute*), 363
date (*telegram.InaccessibleMessage attribute*), 400
date (*telegram.MaybeInaccessibleMessage attribute*), 462
date (*telegram.Message attribute*), 476
date (*telegram.MessageOrigin attribute*), 538
date (*telegram.MessageOriginChannel attribute*), 539
date (*telegram.MessageOriginChat attribute*), 540
date (*telegram.MessageOriginHiddenUser attribute*), 541
date (*telegram.MessageOriginUser attribute*), 541
date (*telegram.MessageReactionCountUpdated attribute*), 542
date (*telegram.MessageReactionUpdated attribute*), 543
date (*telegram.RevenueWithdrawalStateSucceeded attribute*), 761
date (*telegram.StarTransaction attribute*), 766
DATE_TIME (*telegram.constants.MessageEntityType attribute*), 997
DATE_TIME (*telegram.MessageEntity attribute*), 533
date_time_format (*telegram.MessageEntity attribute*), 533
day (*telegram.Birthdate attribute*), 224
de_json() (*telegram.AffiliateInfo class method*), 753
de_json() (*telegram.Animation class method*), 221
de_json() (*telegram.Audio class method*), 224
de_json() (*telegram.BackgroundFill class method*), 296
de_json() (*telegram.BackgroundType class method*), 291
de_json() (*telegram.BotAccessSettings class method*), 225
de_json() (*telegram.BotCommandScope class method*), 227
de_json() (*telegram.BusinessConnection class method*), 236
de_json() (*telegram.BusinessIntro class method*), 237
de_json() (*telegram.BusinessLocation class method*), 237
de_json() (*telegram.BusinessMessagesDeleted class method*), 241
de_json() (*telegram.BusinessOpeningHours class method*), 238
de_json() (*telegram.CallbackQuery class method*), 243
de_json() (*telegram.ChatBackground class method*), 285
de_json() (*telegram.ChatBoost class method*), 299
de_json() (*telegram.ChatBoostRemoved class method*), 300
de_json() (*telegram.ChatBoostSource class method*), 301
de_json() (*telegram.ChatBoostUpdated class method*), 303
de_json() (*telegram.ChatFullInfo class method*), 317
de_json() (*telegram.ChatInviteLink class method*), 346
de_json() (*telegram.ChatJoinRequest class method*), 348
de_json() (*telegram.ChatLocation class method*), 349
de_json() (*telegram.ChatMember class method*), 351
de_json() (*telegram.ChatMemberUpdated class method*), 364
de_json() (*telegram.ChatOwnerChanged class method*), 365
de_json() (*telegram.ChatOwnerLeft class method*), 365
de_json() (*telegram.ChatPermissions class method*), 369
de_json() (*telegram.ChatShared class method*), 372
de_json() (*telegram.Checklist class method*), 286
de_json() (*telegram.ChecklistTask class method*), 288
de_json() (*telegram.ChecklistTasksAdded class method*), 289
de_json() (*telegram.ChecklistTasksDone class method*), 290
de_json() (*telegram.ChosenInlineResult class method*), 690
de_json() (*telegram.Credentials class method*), 780
de_json() (*telegram.DirectMessagesTopic class method*), 377
de_json() (*telegram.Document class method*), 378
de_json() (*telegram.EncryptedPassportElement class method*), 785
de_json() (*telegram.ExternalReplyInfo class method*), 382
de_json() (*telegram.Game class method*), 778
de_json() (*telegram.GameHighScore class method*), 780
de_json() (*telegram.Gift class method*), 681
de_json() (*telegram.GiftInfo class method*), 393
de_json() (*telegram.Gifts class method*), 681
de_json() (*telegram.Giveaway class method*), 396
de_json() (*telegram.GiveawayCompleted class method*), 397
de_json() (*telegram.GiveawayWinners class method*), 400
de_json() (*telegram.InlineKeyboardButton class method*), 406
de_json() (*telegram.InlineKeyboardMarkup class method*), 409
de_json() (*telegram.InlineQuery class method*), 693
de_json() (*telegram.InlineQueryResultsButton class method*), 731
de_json() (*telegram.InputInvoiceMessageContent class method*), 752
de_json() (*telegram.InputPollOption class method*), 443

de_json() (*telegram.KeyboardButton class method*), 448
de_json() (*telegram.KeyboardButtonRequestChat class method*), 451
de_json() (*telegram.LivePhoto class method*), 456
de_json() (*telegram.ManagedBotCreated class method*), 460
de_json() (*telegram.ManagedBotUpdated class method*), 461
de_json() (*telegram.MenuButton class method*), 463
de_json() (*telegram.MenuButtonWebApp class method*), 465
de_json() (*telegram.Message class method*), 496
de_json() (*telegram.MessageEntity class method*), 536
de_json() (*telegram.MessageOrigin class method*), 539
de_json() (*telegram.MessageReactionCountUpdated class method*), 543
de_json() (*telegram.MessageReactionUpdated class method*), 544
de_json() (*telegram.OrderInfo class method*), 757
de_json() (*telegram.OwnedGift class method*), 544
de_json() (*telegram.OwnedGiftRegular class method*), 547
de_json() (*telegram.OwnedGifts class method*), 549
de_json() (*telegram.OwnedGiftUnique class method*), 550
de_json() (*telegram.PaidMedia class method*), 551
de_json() (*telegram.PaidMediaInfo class method*), 552
de_json() (*telegram.PaidMediaLivePhoto class method*), 553
de_json() (*telegram.PaidMediaPhoto class method*), 554
de_json() (*telegram.PaidMediaPurchased class method*), 555
de_json() (*telegram.PaidMediaVideo class method*), 556
de_json() (*telegram.PassportData class method*), 787
de_json() (*telegram.PassportFile class method*), 796
de_json() (*telegram.Poll class method*), 564
de_json() (*telegram.PollAnswer class method*), 568
de_json() (*telegram.PollMedia class method*), 570
de_json() (*telegram.PollOption class method*), 572
de_json() (*telegram.PollOptionAdded class method*), 573
de_json() (*telegram.PollOptionDeleted class method*), 575
de_json() (*telegram.PreCheckoutQuery class method*), 758
de_json() (*telegram.PreparedInlineMessage class method*), 752
de_json() (*telegram.ProximityAlertTriggered class method*), 577
de_json() (*telegram.ReactionCount class method*), 578
de_json() (*telegram.ReactionType class method*), 578
de_json() (*telegram.ReplyParameters class method*), 590
de_json() (*telegram.RevenueWithdrawalState class method*), 760
de_json() (*telegram.RevenueWithdrawalStateSucceeded class method*), 762
de_json() (*telegram.SecureData class method*), 801
de_json() (*telegram.SecureValue class method*), 803
de_json() (*telegram.SharedUser class method*), 592
de_json() (*telegram.ShippingQuery class method*), 765
de_json() (*telegram.StarTransaction class method*), 767
de_json() (*telegram.StarTransactions class method*), 767
de_json() (*telegram.Sticker class method*), 687
de_json() (*telegram.StickerSet class method*), 689
de_json() (*telegram.Story class method*), 593
de_json() (*telegram.SuccessfulPayment class method*), 769
de_json() (*telegram.SuggestedPostApprovalFailed class method*), 600
de_json() (*telegram.SuggestedPostApproved class method*), 601
de_json() (*telegram.SuggestedPostDeclined class method*), 601
de_json() (*telegram.SuggestedPostInfo class method*), 602
de_json() (*telegram.SuggestedPostPaid class method*), 603
de_json() (*telegram.SuggestedPostParameters class method*), 605
de_json() (*telegram.SuggestedPostRefunded class method*), 606
de_json() (*telegram.TelegramObject class method*), 610
de_json() (*telegram.TextQuote class method*), 612
de_json() (*telegram.TransactionPartner class method*), 770
de_json() (*telegram.TransactionPartnerAffiliateProgram class method*), 771
de_json() (*telegram.TransactionPartnerChat class method*), 772
de_json() (*telegram.TransactionPartnerFragment class method*), 772
de_json() (*telegram.TransactionPartnerUser class method*), 776
de_json() (*telegram.UniqueGift class method*), 615
de_json() (*telegram.UniqueGiftBackdrop class method*), 617
de_json() (*telegram.UniqueGiftInfo class method*), 620
de_json() (*telegram.UniqueGiftModel class method*), 621
de_json() (*telegram.UniqueGiftSymbol class method*), 621
de_json() (*telegram.Update class method*), 630
de_json() (*telegram.UserChatBoosts class method*),

660
de_json() (*telegram.UserProfileAudios class method*), 660
de_json() (*telegram.UserProfilePhotos class method*), 661
de_json() (*telegram.UsersShared class method*), 663
de_json() (*telegram.Venue class method*), 664
de_json() (*telegram.Video class method*), 667
de_json() (*telegram.VideoChatParticipantsInvited class method*), 668
de_json() (*telegram.VideoChatScheduled class method*), 669
de_json() (*telegram.VideoNote class method*), 671
de_json() (*telegram.WebhookInfo class method*), 677
de_json_decrypted() (*telegram.EncryptedPassportElement class method*), 785
de_json_decrypted() (*telegram.PassportFile class method*), 796
de_list() (*telegram.TelegramObject class method*), 610
de_list_decrypted() (*telegram.PassportFile class method*), 796
decline() (*telegram.ChatJoinRequest method*), 348
decline_chat_join_request() (*telegram.Bot method*), 45
decline_join_request() (*telegram.Chat method*), 254
decline_join_request() (*telegram.ChatFullInfo method*), 317
decline_join_request() (*telegram.User method*), 638
decline_suggested_post() (*telegram.Bot method*), 46
decline_suggested_post() (*telegram.Chat method*), 255
decline_suggested_post() (*telegram.ChatFullInfo method*), 318
decline_suggested_post() (*telegram.Message method*), 496
declineChatJoinRequest() (*telegram.Bot method*), 45
DECLINED (*telegram.constants.SuggestedPostInfoState attribute*), 1029
DECLINED (*telegram.SuggestedPostInfo attribute*), 602
declineSuggestedPost() (*telegram.Bot method*), 45
decrypted_credentials (*telegram.PassportData property*), 787
decrypted_data (*telegram.EncryptedCredentials property*), 782
decrypted_data (*telegram.PassportData property*), 787
decrypted_secret (*telegram.EncryptedCredentials property*), 782
DEEP_LINK_LENGTH (*telegram.constants.MessageLimit attribute*), 998
DEFAULT (*telegram.BotCommandScope attribute*), 227
DEFAULT (*telegram.constants.BotCommandScopeType attribute*), 966
DEFAULT (*telegram.constants.MenuButtonType attribute*), 992
DEFAULT (*telegram.MenuButton attribute*), 463
DEFAULT_NONE (*telegram.request.BaseRequest attribute*), 1042
DEFAULT_TYPE (*telegram.ext.ContextTypes attribute*), 845
Defaults (*class in telegram.ext*), 845
defaults (*telegram.ext.ExtBot property*), 850
defaults() (*telegram.ext.ApplicationBuilder method*), 824
delete() (*telegram.Message method*), 497
delete_all_message_reactions() (*telegram.Bot method*), 48
delete_all_reactions() (*telegram.Chat method*), 255
delete_all_reactions() (*telegram.ChatFullInfo method*), 318
delete_all_reactions() (*telegram.User method*), 638
delete_business_messages() (*telegram.Bot method*), 49
DELETE_CHAT_PHOTO (*telegram.constants.MessageType attribute*), 1001
DELETE_CHAT_PHOTO (*telegram.ext.filters.StatusUpdate attribute*), 908
delete_chat_photo (*telegram.Message attribute*), 481
delete_chat_photo() (*telegram.Bot method*), 49
delete_chat_sticker_set() (*telegram.Bot method*), 50
delete_forum_topic() (*telegram.Bot method*), 51
delete_forum_topic() (*telegram.Chat method*), 255
delete_forum_topic() (*telegram.ChatFullInfo method*), 318
delete_forum_topic() (*telegram.Message method*), 497
delete_message() (*telegram.Bot method*), 51
delete_message() (*telegram.CallbackQuery method*), 243
delete_message() (*telegram.Chat method*), 255
delete_message() (*telegram.ChatFullInfo method*), 319
delete_message() (*telegram.User method*), 638
delete_message_reaction() (*telegram.Bot method*), 52
delete_messages() (*telegram.Bot method*), 53
delete_messages() (*telegram.Chat method*), 256
delete_messages() (*telegram.ChatFullInfo method*), 319
delete_messages() (*telegram.User method*), 639
delete_my_commands() (*telegram.Bot method*), 54
delete_photo() (*telegram.Chat method*), 256
delete_photo() (*telegram.ChatFullInfo method*), 319

delete_reaction() (*telegram.Chat method*), 256
delete_reaction() (*telegram.ChatFullInfo method*), 319
delete_reaction() (*telegram.Message method*), 497
delete_reaction() (*telegram.User method*), 639
delete_sticker_from_set() (*telegram.Bot method*), 55
delete_sticker_set() (*telegram.Bot method*), 55
delete_story() (*telegram.Bot method*), 56
delete_webhook() (*telegram.Bot method*), 57
deleteAllMessageReactions() (*telegram.Bot method*), 47
deleteBusinessMessages() (*telegram.Bot method*), 47
deleteChatPhoto() (*telegram.Bot method*), 47
deleteChatStickerSet() (*telegram.Bot method*), 47
DELETED_BUSINESS_MESSAGES (*telegram.constants.UpdateType attribute*), 1033
DELETED_BUSINESS_MESSAGES (*telegram.Update attribute*), 629
deleted_business_messages (*telegram.Update attribute*), 627
deleteForumTopic() (*telegram.Bot method*), 47
deleteMessage() (*telegram.Bot method*), 47
deleteMessageReaction() (*telegram.Bot method*), 47
deleteMessages() (*telegram.Bot method*), 47
deleteMyCommands() (*telegram.Bot method*), 47
deleteStickerFromSet() (*telegram.Bot method*), 48
deleteStickerSet() (*telegram.Bot method*), 48
deleteStory() (*telegram.Bot method*), 48
deleteWebhook() (*telegram.Bot method*), 48
description (*telegram.BotCommand attribute*), 226
description (*telegram.BotDescription attribute*), 231
description (*telegram.ChatFullInfo attribute*), 311
description (*telegram.Game attribute*), 778
description (*telegram.InlineQueryResultArticle attribute*), 696
description (*telegram.InlineQueryResultCachedDocument attribute*), 701
description (*telegram.InlineQueryResultCachedPhoto attribute*), 708
description (*telegram.InlineQueryResultCachedVideo attribute*), 711
description (*telegram.InlineQueryResultDocument attribute*), 717
description (*telegram.InlineQueryResultPhoto attribute*), 730
description (*telegram.InlineQueryResultVideo attribute*), 737
description (*telegram.InputInvoiceMessageContent attribute*), 749
description (*telegram.Invoice attribute*), 754
description (*telegram.Poll attribute*), 563
description_entities (*telegram.Poll attribute*), 563
Dice (*class in telegram*), 373
Dice (*class in telegram.ext.filters*), 897
DICE (*telegram.constants.DiceEmoji attribute*), 974
DICE (*telegram.constants.MessageAttachmentType attribute*), 992
DICE (*telegram.constants.MessageType attribute*), 1001
DICE (*telegram.Dice attribute*), 374
DICE (*telegram.ext.filters.Dice attribute*), 898
dice (*telegram.ExternalReplyInfo attribute*), 381
dice (*telegram.Message attribute*), 483
Dice.Basketball (*class in telegram.ext.filters*), 898
Dice.Bowling (*class in telegram.ext.filters*), 898
Dice.Darts (*class in telegram.ext.filters*), 898
Dice.Dice (*class in telegram.ext.filters*), 898
Dice.Football (*class in telegram.ext.filters*), 898
Dice.SlotMachine (*class in telegram.ext.filters*), 899
DiceEmoji (*class in telegram.constants*), 974
DiceLimit (*class in telegram.constants*), 975
DictPersistence (*class in telegram.ext*), 943
difference() (*telegram.ChatMemberUpdated method*), 364
DIRECT_MESSAGE_PRICE_CHANGED (*telegram.constants.MessageType attribute*), 1002
DIRECT_MESSAGE_PRICE_CHANGED (*telegram.ext.filters.StatusUpdate attribute*), 908
direct_message_price_changed (*telegram.Message attribute*), 488
direct_message_star_count (*telegram.DirectMessagePriceChanged attribute*), 376
DIRECT_MESSAGES (*in module telegram.ext.filters*), 889
direct_messages_topic (*telegram.Message attribute*), 489
DirectMessagePriceChanged (*class in telegram*), 375
DirectMessagesTopic (*class in telegram*), 376
disable_content_type_detection (*telegram.InputMediaDocument attribute*), 424
disable_notification (*telegram.ext.Defaults property*), 847
distance (*telegram.ProximityAlertTriggered attribute*), 577
do_api_request() (*telegram.Bot method*), 57
do_process_update() (*telegram.ext.BaseUpdateProcessor method*), 838
do_process_update() (*telegram.ext.SimpleUpdateProcessor method*), 862
do_quote (*telegram.ext.Defaults property*), 848
do_request() (*telegram.request.BaseRequest method*), 1043
do_request() (*telegram.request.HTTPXRequest method*), 1049
DOC (*telegram.ext.filters.Document attribute*), 901
Document (*class in telegram*), 377

Document (*class in telegram.ext.filters*), 899
document (*telegram.BackgroundTypePattern attribute*), 294
document (*telegram.BackgroundTypeWallpaper attribute*), 293
DOCUMENT (*telegram.constants.BaseInputMediaType attribute*), 964
DOCUMENT (*telegram.constants.InlineQueryResultType attribute*), 981
DOCUMENT (*telegram.constants.InputMediaType attribute*), 983
DOCUMENT (*telegram.constants.MessageAttachmentType attribute*), 993
DOCUMENT (*telegram.constants.MessageType attribute*), 1002
document (*telegram.ExternalReplyInfo attribute*), 380
document (*telegram.Message attribute*), 478
document (*telegram.PollMedia attribute*), 569
Document.Category (*class in telegram.ext.filters*), 899
Document.FileExtension (*class in telegram.ext.filters*), 900
Document.MimeType (*class in telegram.ext.filters*), 900
document_file_id (*telegram.InlineQueryResultCachedDocument attribute*), 701
document_no (*telegram.IdDocumentData attribute*), 786
document_url (*telegram.InlineQueryResultDocument attribute*), 717
DOCX (*telegram.ext.filters.Document attribute*), 901
DOVE_OF_PEACE (*telegram.constants.ReactionEmoji attribute*), 1016
download_as_bytearray() (*telegram.File method*), 383
download_to_drive() (*telegram.File method*), 384
download_to_memory() (*telegram.File method*), 385
driver_license (*telegram.SecureData attribute*), 800
drop_callback_data() (*telegram.ext.CallbackContext method*), 842
drop_chat_data() (*telegram.ext.Application method*), 810
drop_chat_data() (*telegram.ext.BasePersistence method*), 939
drop_chat_data() (*telegram.ext.DictPersistence method*), 945
drop_chat_data() (*telegram.ext.PicklePersistence method*), 950
drop_data() (*telegram.ext.CallbackDataCache method*), 953
drop_user_data() (*telegram.ext.Application method*), 810
drop_user_data() (*telegram.ext.BasePersistence method*), 939
drop_user_data() (*telegram.ext.DictPersistence method*), 945
drop_user_data() (*telegram.ext.PicklePersistence method*), 950
duration (*telegram.Animation attribute*), 221
duration (*telegram.Audio attribute*), 223
duration (*telegram.InputMediaAnimation attribute*), 419
duration (*telegram.InputMediaAudio attribute*), 421
duration (*telegram.InputMediaVideo attribute*), 434
duration (*telegram.InputPaidMediaVideo attribute*), 439
duration (*telegram.InputStoryContentVideo attribute*), 445
duration (*telegram.LivePhoto attribute*), 456
duration (*telegram.PaidMediaPreview attribute*), 555
duration (*telegram.Video attribute*), 666
duration (*telegram.VideoChatEnded attribute*), 668
duration (*telegram.VideoNote attribute*), 670
duration (*telegram.Voice attribute*), 673

E

edge_color (*telegram.GiftBackground attribute*), 391
edge_color (*telegram.UniqueGiftBackdropColors attribute*), 617
edit_caption() (*telegram.Message method*), 498
edit_chat_invite_link() (*telegram.Bot method*), 59
edit_chat_subscription_invite_link() (*telegram.Bot method*), 60
edit_checklist() (*telegram.Message method*), 498
edit_date (*telegram.Message attribute*), 477
edit_forum_topic() (*telegram.Bot method*), 61
edit_forum_topic() (*telegram.Chat method*), 257
edit_forum_topic() (*telegram.ChatFullInfo method*), 320
edit_forum_topic() (*telegram.Message method*), 499
edit_general_forum_topic() (*telegram.Bot method*), 62
edit_general_forum_topic() (*telegram.Chat method*), 257
edit_general_forum_topic() (*telegram.ChatFullInfo method*), 320
edit_invite_link() (*telegram.Chat method*), 257
edit_invite_link() (*telegram.ChatFullInfo method*), 320
edit_live_location() (*telegram.Message method*), 499
edit_media() (*telegram.Message method*), 500
edit_message_caption() (*telegram.Bot method*), 63
edit_message_caption() (*telegram.CallbackQuery method*), 244
edit_message_checklist() (*telegram.Bot method*), 64
edit_message_checklist() (*telegram.CallbackQuery method*), 244
edit_message_live_location() (*telegram.Bot method*), 65

edit_message_live_location() (*telegram.CallbackQuery method*), 244
edit_message_media() (*telegram.Bot method*), 66
edit_message_media() (*telegram.CallbackQuery method*), 245
edit_message_reply_markup() (*telegram.Bot method*), 68
edit_message_reply_markup() (*telegram.CallbackQuery method*), 245
edit_message_text() (*telegram.Bot method*), 69
edit_message_text() (*telegram.CallbackQuery method*), 246
edit_reply_markup() (*telegram.Message method*), 500
edit_story() (*telegram.Bot method*), 70
edit_subscription_invite_link() (*telegram.Chat method*), 257
edit_subscription_invite_link() (*telegram.ChatFullInfo method*), 320
edit_text() (*telegram.Message method*), 501
edit_user_star_subscription() (*telegram.Bot method*), 71
editChatInviteLink() (*telegram.Bot method*), 58
editChatSubscriptionInviteLink() (*telegram.Bot method*), 58
EDITED (*telegram.ext.filters.UpdateType attribute*), 913
EDITED_BUSINESS_MESSAGE (*telegram.constants.UpdateType attribute*), 1033
EDITED_BUSINESS_MESSAGE (*telegram.ext.filters.UpdateType attribute*), 914
EDITED_BUSINESS_MESSAGE (*telegram.Update attribute*), 629
edited_business_message (*telegram.Update attribute*), 627
EDITED_CHANNEL_POST (*telegram.constants.UpdateType attribute*), 1033
EDITED_CHANNEL_POST (*telegram.ext.filters.UpdateType attribute*), 914
EDITED_CHANNEL_POST (*telegram.Update attribute*), 629
edited_channel_post (*telegram.Update attribute*), 625
EDITED_MESSAGE (*telegram.constants.UpdateType attribute*), 1033
EDITED_MESSAGE (*telegram.ext.filters.UpdateType attribute*), 914
EDITED_MESSAGE (*telegram.Update attribute*), 629
edited_message (*telegram.Update attribute*), 624
editForumTopic() (*telegram.Bot method*), 58
editGeneralForumTopic() (*telegram.Bot method*), 58
editMessageCaption() (*telegram.Bot method*), 58
editMessageChecklist() (*telegram.Bot method*), 58
editMessageLiveLocation() (*telegram.Bot method*), 58
editMessageMedia() (*telegram.Bot method*), 59
editMessageReplyMarkup() (*telegram.Bot method*), 59
editMessageText() (*telegram.Bot method*), 59
editStory() (*telegram.Bot method*), 59
editUserStarSubscription() (*telegram.Bot method*), 59
EFFECT_ID (*in module telegram.ext.filters*), 889
EFFECT_ID (*telegram.constants.MessageType attribute*), 1002
effect_id (*telegram.Message attribute*), 478
effective_attachment (*telegram.Message property*), 501
effective_chat (*telegram.Update property*), 630
effective_message (*telegram.Update property*), 630
effective_message_type() (*in module telegram.helpers*), 1041
effective_name (*telegram.Chat property*), 258
effective_name (*telegram.ChatFullInfo property*), 321
effective_sender (*telegram.Update property*), 631
effective_user (*telegram.Update property*), 631
element_hash (*telegram.PassportElementErrorUnspecified attribute*), 795
EMAIL (*telegram.constants.MessageEntityType attribute*), 997
email (*telegram.EncryptedPassportElement attribute*), 784
EMAIL (*telegram.MessageEntity attribute*), 533
email (*telegram.OrderInfo attribute*), 757
EMOJI (*telegram.constants.ReactionType attribute*), 1022
emoji (*telegram.Dice attribute*), 374
emoji (*telegram.InputMediaSticker attribute*), 429
EMOJI (*telegram.ReactionType attribute*), 578
emoji (*telegram.ReactionTypeEmoji attribute*), 580
emoji (*telegram.Sticker attribute*), 686
emoji (*telegram.StoryAreaTypeWeather attribute*), 599
emoji_list (*telegram.InputSticker attribute*), 682
emoji_status_custom_emoji_id (*telegram.ChatFullInfo attribute*), 310
emoji_status_expiration_date (*telegram.ChatFullInfo attribute*), 310
enabled (*telegram.ext.Job property*), 853
EncryptedCredentials (*class in telegram*), 781
EncryptedPassportElement (*class in telegram*), 782
END (*telegram.ext.ConversationHandler attribute*), 886
EndPointNotFound, 1037
entities (*telegram.GiftInfo attribute*), 393
entities (*telegram.InputTextMessageContent attribute*), 742
entities (*telegram.Message attribute*), 477
entities (*telegram.OwnedGiftRegular attribute*), 546
entities (*telegram.TextQuote attribute*), 612
Entity (*class in telegram.ext.filters*), 901
entry_points (*telegram.ext.ConversationHandler*

property), 887
EPIC (*telegram.constants.UniqueGiftModelRarity attribute*), 1032
error (*telegram.ext.CallbackContext attribute*), 841
error_handlers (*telegram.ext.Application attribute*), 806
escape_markdown() (*in module telegram.helpers*), 1041
EXE (*telegram.ext.filters.Document attribute*), 901
EXPANDABLE_BLOCKQUOTE (*telegram.constants.MessageEntityType attribute*), 997
EXPANDABLE_BLOCKQUOTE (*telegram.MessageEntity attribute*), 534
expiration_date (*telegram.ChatBoost attribute*), 298
expiration_date (*telegram.PreparedInlineMessage attribute*), 752
expire_date (*telegram.ChatInviteLink attribute*), 345
expiry_date (*telegram.IdDocumentData attribute*), 786
explanation (*telegram.Poll attribute*), 562
explanation_entities (*telegram.Poll attribute*), 562
explanation_media (*telegram.Poll attribute*), 562
explanation_parse_mode (*telegram.ext.Defaults property*), 848
export_chat_invite_link() (*telegram.Bot method*), 72
export_invite_link() (*telegram.Chat method*), 258
export_invite_link() (*telegram.ChatFullInfo method*), 321
exportChatInviteLink() (*telegram.Bot method*), 72
ExtBot (*class in telegram.ext*), 849
external_reply (*telegram.Message attribute*), 487
ExternalReplyInfo (*class in telegram*), 378
extract_uuids() (*telegram.ext.CallbackDataCache static method*), 954
EYES (*telegram.constants.MaskPosition attribute*), 991
EYES (*telegram.constants.ReactionEmoji attribute*), 1016
EYES (*telegram.MaskPosition attribute*), 683

F

FACE_SCREAMING_IN_FEAR (*telegram.constants.ReactionEmoji attribute*), 1016
FACE_THROWING_A_KISS (*telegram.constants.ReactionEmoji attribute*), 1016
FACE_WITH_ONE_EYEBROW_RAISED (*telegram.constants.ReactionEmoji attribute*), 1016
FACE_WITH_OPEN_MOUTH_VOMITING (*telegram.constants.ReactionEmoji attribute*), 1016
FACE_WITH_UNEVEN_EYES_AND_WAVY_MOUTH (*telegram.constants.ReactionEmoji attribute*), 1016
FAILED (*telegram.constants.RevenueWithdrawalStateType attribute*), 1023
FAILED (*telegram.RevenueWithdrawalState attribute*), 760
FAKE_CHANNEL (*telegram.constants.ChatID attribute*), 970
fallbacks (*telegram.ext.ConversationHandler property*), 887
FATHER_CHRISTMAS (*telegram.constants.ReactionEmoji attribute*), 1017
FEARFUL_FACE (*telegram.constants.ReactionEmoji attribute*), 1017
field_name (*telegram.PassportElementErrorDataField attribute*), 788
field_tuple (*telegram.InputFile property*), 415
File (*class in telegram*), 382
file_date (*telegram.PassportFile attribute*), 796
file_hash (*telegram.PassportElementErrorFile attribute*), 789
file_hash (*telegram.PassportElementErrorFrontSide attribute*), 791
file_hash (*telegram.PassportElementErrorReverseSide attribute*), 792
file_hash (*telegram.PassportElementErrorSelfie attribute*), 792
file_hash (*telegram.PassportElementErrorTranslationFile attribute*), 793
file_hashes (*telegram.PassportElementErrorFiles attribute*), 790
file_hashes (*telegram.PassportElementErrorTranslationFiles attribute*), 794
file_id (*telegram.Animation attribute*), 221
file_id (*telegram.Audio attribute*), 223
file_id (*telegram.Document attribute*), 377
file_id (*telegram.File attribute*), 383
file_id (*telegram.LivePhoto attribute*), 456
file_id (*telegram.PassportFile attribute*), 795
file_id (*telegram.PhotoSize attribute*), 558
file_id (*telegram.Sticker attribute*), 685
file_id (*telegram.Video attribute*), 666
file_id (*telegram.VideoNote attribute*), 670
file_id (*telegram.VideoQuality attribute*), 671
file_id (*telegram.Voice attribute*), 673
file_name (*telegram.Animation attribute*), 221
file_name (*telegram.Audio attribute*), 223
file_name (*telegram.Document attribute*), 378
file_name (*telegram.Video attribute*), 666
file_path (*telegram.File attribute*), 383
file_size (*telegram.Animation attribute*), 221
file_size (*telegram.Audio attribute*), 223
file_size (*telegram.Document attribute*), 378
file_size (*telegram.File attribute*), 383
file_size (*telegram.LivePhoto attribute*), 456
file_size (*telegram.PassportFile attribute*), 796
file_size (*telegram.PhotoSize attribute*), 558
file_size (*telegram.Sticker attribute*), 686

file_size (*telegram.Video attribute*), 666
file_size (*telegram.VideoNote attribute*), 670
file_size (*telegram.VideoQuality attribute*), 672
file_size (*telegram.Voice attribute*), 673
file_unique_id (*telegram.Animation attribute*), 221
file_unique_id (*telegram.Audio attribute*), 223
file_unique_id (*telegram.Document attribute*), 377
file_unique_id (*telegram.File attribute*), 383
file_unique_id (*telegram.LivePhoto attribute*), 456
file_unique_id (*telegram.PassportFile attribute*), 795
file_unique_id (*telegram.PhotoSize attribute*), 558
file_unique_id (*telegram.Sticker attribute*), 685
file_unique_id (*telegram.Video attribute*), 666
file_unique_id (*telegram.VideoNote attribute*), 670
file_unique_id (*telegram.VideoQuality attribute*), 671
file_unique_id (*telegram.Voice attribute*), 673
FileCredentials (*class in telegram*), 785
filename (*telegram.InputFile attribute*), 415
filepath (*telegram.ext.PicklePersistence attribute*), 949
files (*telegram.EncryptedPassportElement attribute*), 784
files (*telegram.SecureValue attribute*), 803
FILESIZE_DOWNLOAD (*telegram.constants.FileSizeLimit attribute*), 976
FILESIZE_DOWNLOAD_LOCAL_MODE (*telegram.constants.FileSizeLimit attribute*), 976
FILESIZE_UPLOAD (*telegram.constants.FileSizeLimit attribute*), 976
FILESIZE_UPLOAD_LOCAL_MODE (*telegram.constants.FileSizeLimit attribute*), 976
FileSizeLimit (*class in telegram.constants*), 975
FILL (*telegram.BackgroundType attribute*), 291
fill (*telegram.BackgroundTypeFill attribute*), 292
fill (*telegram.BackgroundTypePattern attribute*), 294
FILL (*telegram.constants.BackgroundTypeType attribute*), 963
filter() (*telegram.ext.filters.MessageFilter method*), 905
filter() (*telegram.ext.filters.UpdateFilter method*), 913
filters (*telegram.ext.CommandHandler attribute*), 883
filters (*telegram.ext.MessageHandler attribute*), 921
filters (*telegram.ext.PrefixHandler attribute*), 931
FIND_LOCATION (*telegram.constants.ChatAction attribute*), 968
FIRE (*telegram.constants.ReactionEmoji attribute*), 1017
first_name (*telegram.Bot property*), 73
first_name (*telegram.Chat attribute*), 250
first_name (*telegram.ChatFullInfo attribute*), 308
first_name (*telegram.Contact attribute*), 373
first_name (*telegram.InlineQueryResultContact attribute*), 714
first_name (*telegram.InputContactMessageContent attribute*), 747
first_name (*telegram.PersonalDetails attribute*), 797
first_name (*telegram.SharedUser attribute*), 591
first_name (*telegram.User attribute*), 635
first_name_native (*telegram.PersonalDetails attribute*), 798
first_profile_audio (*telegram.ChatFullInfo attribute*), 313
FloodLimit (*class in telegram.constants*), 976
flush() (*telegram.ext.BasePersistence method*), 939
flush() (*telegram.ext.DictPersistence method*), 945
flush() (*telegram.ext.PicklePersistence method*), 950
FOOTBALL (*telegram.constants.DiceEmoji attribute*), 975
FOOTBALL (*telegram.Dice attribute*), 374
FOOTBALL (*telegram.ext.filters.Dice attribute*), 899
Forbidden, 1037
force_reply (*telegram.ForceReply attribute*), 387
ForceReply (*class in telegram*), 386
FOREHEAD (*telegram.constants.MaskPosition attribute*), 991
FOREHEAD (*telegram.MaskPosition attribute*), 683
format (*telegram.InputSticker attribute*), 682
FORUM (*in module telegram.ext.filters*), 889
FORUM_TOPIC_CLOSED (*telegram.constants.MessageType attribute*), 1002
FORUM_TOPIC_CLOSED (*telegram.ext.filters.StatusUpdate attribute*), 908
forum_topic_closed (*telegram.Message attribute*), 484
FORUM_TOPIC_CREATED (*telegram.constants.MessageType attribute*), 1002
FORUM_TOPIC_CREATED (*telegram.ext.filters.StatusUpdate attribute*), 908
forum_topic_created (*telegram.Message attribute*), 484
FORUM_TOPIC_EDITED (*telegram.constants.MessageType attribute*), 1002
FORUM_TOPIC_EDITED (*telegram.ext.filters.StatusUpdate attribute*), 909
forum_topic_edited (*telegram.Message attribute*), 484
FORUM_TOPIC_REOPENED (*telegram.constants.MessageType attribute*), 1002
FORUM_TOPIC_REOPENED (*telegram.ext.filters.StatusUpdate attribute*), 909
forum_topic_reopened (*telegram.Message at-*

tribute), 484
ForumIconColor (*class in telegram.constants*), 977
ForumTopic (*class in telegram*), 387
ForumTopicClosed (*class in telegram*), 388
ForumTopicCreated (*class in telegram*), 389
ForumTopicEdited (*class in telegram*), 390
ForumTopicLimit (*class in telegram.constants*), 977
ForumTopicReopened (*class in telegram*), 390
forward() (*telegram.Message method*), 502
forward_from() (*telegram.Chat method*), 258
forward_from() (*telegram.ChatFullInfo method*), 321
forward_from() (*telegram.User method*), 639
forward_message() (*telegram.Bot method*), 73
forward_messages() (*telegram.Bot method*), 75
forward_messages_from() (*telegram.Chat method*), 258
forward_messages_from() (*telegram.ChatFullInfo method*), 322
forward_messages_from() (*telegram.User method*), 640
forward_messages_to() (*telegram.Chat method*), 259
forward_messages_to() (*telegram.ChatFullInfo method*), 322
forward_messages_to() (*telegram.User method*), 640
forward_origin (*telegram.Message attribute*), 487
forward_text (*telegram.LoginUrl attribute*), 460
forward_to() (*telegram.Chat method*), 259
forward_to() (*telegram.ChatFullInfo method*), 322
forward_to() (*telegram.User method*), 640
FORWARDED (*in module telegram.ext.filters*), 890
ForwardedFrom (*class in telegram.ext.filters*), 902
forwardMessage() (*telegram.Bot method*), 73
forwardMessages() (*telegram.Bot method*), 73
foursquare_id (*telegram.InlineQueryResultVenue attribute*), 733
foursquare_id (*telegram.InputMediaVenue attribute*), 431
foursquare_id (*telegram.InputVenueMessageContent attribute*), 746
foursquare_id (*telegram.Venue attribute*), 664
foursquare_type (*telegram.InlineQueryResultVenue attribute*), 733
foursquare_type (*telegram.InputMediaVenue attribute*), 431
foursquare_type (*telegram.InputVenueMessageContent attribute*), 746
foursquare_type (*telegram.Venue attribute*), 664
FRAGMENT (*telegram.constants.TransactionPartnerType attribute*), 1030
FRAGMENT (*telegram.TransactionPartner attribute*), 770
FREEFORM_GRADIENT (*telegram.BackgroundFill attribute*), 296
FREEFORM_GRADIENT (*telegram.constants.BackgroundFillType attribute*), 962
from_aps_job() (*telegram.ext.Job class method*), 854
from_attachment_menu (*telegram.WriteAccessAllowed attribute*), 678
from_button() (*telegram.InlineKeyboardMarkup class method*), 409
from_button() (*telegram.ReplyKeyboardMarkup class method*), 584
from_column() (*telegram.InlineKeyboardMarkup class method*), 410
from_column() (*telegram.ReplyKeyboardMarkup class method*), 584
from_error() (*telegram.ext.CallbackContext class method*), 842
from_job() (*telegram.ext.CallbackContext class method*), 842
from_request (*telegram.WriteAccessAllowed attribute*), 678
from_row() (*telegram.InlineKeyboardMarkup class method*), 410
from_row() (*telegram.ReplyKeyboardMarkup class method*), 585
from_update() (*telegram.ext.CallbackContext class method*), 843
from_user (*telegram.CallbackQuery attribute*), 242
from_user (*telegram.ChatJoinRequest attribute*), 347
from_user (*telegram.ChatMemberUpdated attribute*), 363
from_user (*telegram.ChosenInlineResult attribute*), 690
from_user (*telegram.InlineQuery attribute*), 692
from_user (*telegram.Message attribute*), 476
from_user (*telegram.PaidMediaPurchased attribute*), 555
from_user (*telegram.PreCheckoutQuery attribute*), 758
from_user (*telegram.ShippingQuery attribute*), 764
front_side (*telegram.EncryptedPassportElement attribute*), 784
front_side (*telegram.SecureValue attribute*), 802
full_name (*telegram.Chat property*), 260
full_name (*telegram.ChatFullInfo property*), 323
full_name (*telegram.SharedUser property*), 592
full_name (*telegram.User property*), 641

G

Game (*class in telegram*), 777
GAME (*in module telegram.ext.filters*), 890
GAME (*telegram.constants.InlineQueryResultType attribute*), 981
GAME (*telegram.constants.MessageAttachmentType attribute*), 993
GAME (*telegram.constants.MessageType attribute*), 1002
game (*telegram.ExternalReplyInfo attribute*), 381
game (*telegram.Message attribute*), 479
game_pattern (*telegram.ext.CallbackQueryHandler attribute*), 874

game_short_name (*telegram.CallbackQuery attribute*), 242
game_short_name (*telegram.InlineQueryResultGame attribute*), 719
GameHighScore (*class in telegram*), 779
gender (*telegram.PersonalDetails attribute*), 798
GENERAL_FORUM_TOPIC_HIDDEN (*telegram.constants.MessageType attribute*), 1002
GENERAL_FORUM_TOPIC_HIDDEN (*telegram.ext.filters.StatusUpdate attribute*), 909
general_forum_topic_hidden (*telegram.Message attribute*), 484
GENERAL_FORUM_TOPIC_UNHIDDEN (*telegram.constants.MessageType attribute*), 1003
GENERAL_FORUM_TOPIC_UNHIDDEN (*telegram.ext.filters.StatusUpdate attribute*), 909
general_forum_topic_unhidden (*telegram.Message attribute*), 485
GeneralForumTopicHidden (*class in telegram*), 390
GeneralForumTopicUnhidden (*class in telegram*), 391
get_administrators() (*telegram.Chat method*), 260
get_administrators() (*telegram.ChatFullInfo method*), 323
get_available_gifts() (*telegram.Bot method*), 78
get_big_file() (*telegram.ChatPhoto method*), 370
get_bot() (*telegram.TelegramObject method*), 611
get_bot_data() (*telegram.ext.BasePersistence method*), 939
get_bot_data() (*telegram.ext.DictPersistence method*), 945
get_bot_data() (*telegram.ext.PicklePersistence method*), 950
get_business_account_gifts() (*telegram.Bot method*), 79
get_business_account_star_balance() (*telegram.Bot method*), 80
get_business_connection() (*telegram.Bot method*), 81
get_callback_data() (*telegram.ext.BasePersistence method*), 940
get_callback_data() (*telegram.ext.DictPersistence method*), 945
get_callback_data() (*telegram.ext.PicklePersistence method*), 950
get_chat() (*telegram.Bot method*), 81
get_chat_administrators() (*telegram.Bot method*), 82
get_chat_boosts() (*telegram.User method*), 641
get_chat_data() (*telegram.ext.BasePersistence method*), 940
get_chat_data() (*telegram.ext.DictPersistence method*), 946
get_chat_data() (*telegram.ext.PicklePersistence method*), 950
get_chat_gifts() (*telegram.Bot method*), 82
get_chat_member() (*telegram.Bot method*), 84
get_chat_member_count() (*telegram.Bot method*), 84
get_chat_menu_button() (*telegram.Bot method*), 85
get_conversations() (*telegram.ext.BasePersistence method*), 940
get_conversations() (*telegram.ext.DictPersistence method*), 946
get_conversations() (*telegram.ext.PicklePersistence method*), 951
get_custom_emoji_stickers() (*telegram.Bot method*), 86
get_file() (*telegram.Animation method*), 222
get_file() (*telegram.Audio method*), 224
get_file() (*telegram.Bot method*), 86
get_file() (*telegram.Document method*), 378
get_file() (*telegram.PassportFile method*), 797
get_file() (*telegram.PhotoSize method*), 559
get_file() (*telegram.Sticker method*), 687
get_file() (*telegram.Video method*), 667
get_file() (*telegram.VideoNote method*), 671
get_file() (*telegram.VideoQuality method*), 672
get_file() (*telegram.Voice method*), 673
get_forum_topic_icon_stickers() (*telegram.Bot method*), 87
get_game_high_scores() (*telegram.Bot method*), 88
get_game_high_scores() (*telegram.CallbackQuery method*), 246
get_game_high_scores() (*telegram.Message method*), 502
get_gifts() (*telegram.Chat method*), 260
get_gifts() (*telegram.ChatFullInfo method*), 323
get_gifts() (*telegram.User method*), 641
get_jobs_by_name() (*telegram.ext.JobQueue method*), 856
get_managed_bot_access_settings() (*telegram.Bot method*), 89
get_managed_bot_access_settings() (*telegram.User method*), 642
get_managed_bot_token() (*telegram.Bot method*), 89
get_me() (*telegram.Bot method*), 90
get_member() (*telegram.Chat method*), 260
get_member() (*telegram.ChatFullInfo method*), 323
get_member_count() (*telegram.Chat method*), 261
get_member_count() (*telegram.ChatFullInfo method*), 324
get_menu_button() (*telegram.Chat method*), 261
get_menu_button() (*telegram.ChatFullInfo method*), 324
get_menu_button() (*telegram.User method*), 642
get_my_commands() (*telegram.Bot method*), 90
get_my_default_administrator_rights() (*telegram.Bot method*), 91
get_my_description() (*telegram.Bot method*), 92
get_my_name() (*telegram.Bot method*), 92

get_my_short_description() (*telegram.Bot method*), 93
get_my_star_balance() (*telegram.Bot method*), 93
get_opening_hours_for_day() (*telegram.BusinessOpeningHours method*), 238
get_personal_chat_messages() (*telegram.User method*), 642
get_profile_audios() (*telegram.User method*), 643
get_profile_photos() (*telegram.User method*), 643
get_small_file() (*telegram.ChatPhoto method*), 371
get_star_transactions() (*telegram.Bot method*), 94
get_sticker_set() (*telegram.Bot method*), 94
get_updates() (*telegram.Bot method*), 95
get_updates_connect_timeout() (*telegram.ext.ApplicationBuilder method*), 825
get_updates_connection_pool_size() (*telegram.ext.ApplicationBuilder method*), 825
get_updates_http_version() (*telegram.ext.ApplicationBuilder method*), 825
get_updates_pool_timeout() (*telegram.ext.ApplicationBuilder method*), 826
get_updates_proxy() (*telegram.ext.ApplicationBuilder method*), 826
get_updates_read_timeout() (*telegram.ext.ApplicationBuilder method*), 827
get_updates_request() (*telegram.ext.ApplicationBuilder method*), 827
get_updates_socket_options() (*telegram.ext.ApplicationBuilder method*), 827
get_updates_write_timeout() (*telegram.ext.ApplicationBuilder method*), 828
get_user_chat_boosts() (*telegram.Bot method*), 96
get_user_chat_boosts() (*telegram.Chat method*), 261
get_user_chat_boosts() (*telegram.ChatFullInfo method*), 324
get_user_data() (*telegram.ext.BasePersistence method*), 940
get_user_data() (*telegram.ext.DictPersistence method*), 946
get_user_data() (*telegram.ext.PicklePersistence method*), 951
get_user_gifts() (*telegram.Bot method*), 97
get_user_personal_chat_messages() (*telegram.Bot method*), 98
get_user_profile_audios() (*telegram.Bot method*), 98
get_user_profile_photos() (*telegram.Bot method*), 99
get_webhook_info() (*telegram.Bot method*), 100
getAvailableGifts() (*telegram.Bot method*), 76
getBusinessAccountGifts() (*telegram.Bot method*), 76
getBusinessAccountStarBalance() (*telegram.Bot method*), 76
getBusinessConnection() (*telegram.Bot method*), 76
getChat() (*telegram.Bot method*), 76
getChatAdministrators() (*telegram.Bot method*), 76
getChatGifts() (*telegram.Bot method*), 76
getChatMember() (*telegram.Bot method*), 77
getChatMemberCount() (*telegram.Bot method*), 77
getChatMenuButton() (*telegram.Bot method*), 77
getCustomEmojiStickers() (*telegram.Bot method*), 77
getFile() (*telegram.Bot method*), 77
getForumTopicIconStickers() (*telegram.Bot method*), 77
getGameHighScores() (*telegram.Bot method*), 77
getManagedBotAccessSettings() (*telegram.Bot method*), 77
getManagedBotToken() (*telegram.Bot method*), 77
getMe() (*telegram.Bot method*), 77
getMyCommands() (*telegram.Bot method*), 77
getMyDefaultAdministratorRights() (*telegram.Bot method*), 77
getMyDescription() (*telegram.Bot method*), 77
getMyName() (*telegram.Bot method*), 78
getMyShortDescription() (*telegram.Bot method*), 78
getMyStarBalance() (*telegram.Bot method*), 78
getStarTransactions() (*telegram.Bot method*), 78
getStickerSet() (*telegram.Bot method*), 78
getUpdates() (*telegram.Bot method*), 78
getUserChatBoosts() (*telegram.Bot method*), 78
getUserGifts() (*telegram.Bot method*), 78
getUserPersonalChatMessages() (*telegram.Bot method*), 78
getUserProfileAudios() (*telegram.Bot method*), 78
getUserProfilePhotos() (*telegram.Bot method*), 78
getWebhookInfo() (*telegram.Bot method*), 78
GHOST (*telegram.constants.ReactionEmoji attribute*), 1017
GIF (*telegram.constants.InlineQueryResultType attribute*), 981
GIF (*telegram.ext.filters.Document attribute*), 901
gif_duration (*telegram.InlineQueryResultGif attribute*), 721
gif_file_id (*telegram.InlineQueryResultCachedGif attribute*), 703
gif_height (*telegram.InlineQueryResultGif attribute*), 720
gif_url (*telegram.InlineQueryResultGif attribute*), 720

gif_width (*telegram.InlineQueryResultGif attribute*), 720
Gift (*class in telegram*), 678
GIFT (*telegram.constants.MessageType attribute*), 1003
GIFT (*telegram.ext.filters.StatusUpdate attribute*), 909
gift (*telegram.GiftInfo attribute*), 392
gift (*telegram.Message attribute*), 485
gift (*telegram.OwnedGiftRegular attribute*), 546
gift (*telegram.OwnedGiftUnique attribute*), 550
gift (*telegram.TransactionPartnerChat attribute*), 772
gift (*telegram.TransactionPartnerUser attribute*), 776
gift (*telegram.UniqueGiftInfo attribute*), 619
GIFT_CODE (*telegram.ChatBoostSource attribute*), 300
GIFT_CODE (*telegram.constants.ChatBoostSources attribute*), 969
gift_id (*telegram.UniqueGift attribute*), 614
gift_premium_subscription() (*telegram.Bot method*), 100
gift_premium_subscription() (*telegram.User method*), 643
GIFT_PURCHASE (*telegram.constants.TransactionPartnerUser attribute*), 1031
GIFT_UPGRADE_SENT (*telegram.constants.MessageType attribute*), 1003
GIFT_UPGRADE_SENT (*telegram.ext.filters.StatusUpdate attribute*), 909
gift_upgrade_sent (*telegram.Message attribute*), 485
GiftBackground (*class in telegram*), 391
GIFTED_UPGRADE (*telegram.constants.UniqueGiftInfoOrigin attribute*), 1031
GIFTED_UPGRADE (*telegram.UniqueGiftInfo attribute*), 619
GiftInfo (*class in telegram*), 392
GiftLimit (*class in telegram.constants*), 978
giftPremiumSubscription() (*telegram.Bot method*), 100
Gifts (*class in telegram*), 681
gifts (*telegram.Gifts attribute*), 681
gifts (*telegram.OwnedGifts attribute*), 549
gifts_from_channels (*telegram.AcceptedGiftTypes attribute*), 220
Giveaway (*class in telegram*), 394
GIVEAWAY (*in module telegram.ext.filters*), 890
GIVEAWAY (*telegram.ChatBoostSource attribute*), 301
GIVEAWAY (*telegram.constants.ChatBoostSources attribute*), 969
GIVEAWAY (*telegram.constants.MessageType attribute*), 1003
giveaway (*telegram.ExternalReplyInfo attribute*), 381
giveaway (*telegram.Message attribute*), 486
GIVEAWAY_COMPLETED (*telegram.constants.MessageType attribute*), 1003
GIVEAWAY_COMPLETED (*telegram.ext.filters.StatusUpdate attribute*), 909
giveaway_completed (*telegram.Message attribute*), 486
GIVEAWAY_CREATED (*telegram.constants.MessageType attribute*), 1003
GIVEAWAY_CREATED (*telegram.ext.filters.StatusUpdate attribute*), 909
giveaway_created (*telegram.Message attribute*), 486
giveaway_message (*telegram.GiveawayCompleted attribute*), 397
giveaway_message_id (*telegram.ChatBoostSourceGiveaway attribute*), 302
giveaway_message_id (*telegram.GiveawayWinners attribute*), 398
GIVEAWAY_WINNERS (*in module telegram.ext.filters*), 890
GIVEAWAY_WINNERS (*telegram.constants.MessageType attribute*), 1003
giveaway_winners (*telegram.ExternalReplyInfo attribute*), 382
giveaway_winners (*telegram.Message attribute*), 486
GiveawayCompleted (*class in telegram*), 396
GiveawayCreated (*class in telegram*), 397
GiveawayLimit (*class in telegram.constants*), 978
GiveawayWinners (*class in telegram*), 398
google_place_id (*telegram.InlineQueryResultVenue attribute*), 733
google_place_id (*telegram.InputMediaVenue attribute*), 431
google_place_id (*telegram.InputVenueMessageContent attribute*), 746
google_place_id (*telegram.Venue attribute*), 664
google_place_type (*telegram.InlineQueryResultVenue attribute*), 733
google_place_type (*telegram.InputMediaVenue attribute*), 431
google_place_type (*telegram.InputVenueMessageContent attribute*), 746
google_place_type (*telegram.Venue attribute*), 664
GRADIENT (*telegram.BackgroundFill attribute*), 296
GRADIENT (*telegram.constants.BackgroundFillType attribute*), 962
GREEN (*telegram.constants.ForumIconColor attribute*), 977
GREEN (*telegram.constants.KeyboardButtonStyle attribute*), 988
GRINNING_FACE_WITH_ONE_LARGE_AND_ONE_SMALL_EYE (*telegram.constants.ReactionEmoji attribute*), 1017
GRINNING_FACE_WITH_SMILING_EYES (*telegram.constants.ReactionEmoji attribute*), 1017

GRINNING_FACE_WITH_STAR_EYES (*telegram.constants.ReactionEmoji attribute*), 1017
GROUP (*telegram.Chat attribute*), 251
GROUP (*telegram.ChatFullInfo attribute*), 314
GROUP (*telegram.constants.ChatType attribute*), 973
GROUP (*telegram.ext.filters.ChatType attribute*), 897
GROUP_CHAT_CREATED (*telegram.constants.MessageType attribute*), 1003
group_chat_created (*telegram.Message attribute*), 481
GROUPS (*telegram.ext.filters.ChatType attribute*), 897
guest_bot_caller_chat (*telegram.Message attribute*), 490
guest_bot_caller_user (*telegram.Message attribute*), 490
GUEST_MESSAGE (*telegram.constants.UpdateType attribute*), 1033
GUEST_MESSAGE (*telegram.ext.filters.UpdateType attribute*), 914
GUEST_MESSAGE (*telegram.Update attribute*), 629
guest_message (*telegram.Update attribute*), 627
guest_query_id (*telegram.Message attribute*), 490

H

handle_update() (*telegram.ext.BaseHandler method*), 869
handle_update() (*telegram.ext.ConversationHandler method*), 887
handlers (*telegram.ext.Application attribute*), 806
HANDSHAKE (*telegram.constants.ReactionEmoji attribute*), 1017
has_aggressive_anti_spam_enabled (*telegram.ChatFullInfo attribute*), 312
has_args (*telegram.ext.CommandHandler attribute*), 883
has_colors (*telegram.Gift attribute*), 680
has_custom_certificate (*telegram.WebhookInfo attribute*), 676
has_hidden_members (*telegram.ChatFullInfo attribute*), 312
has_main_web_app (*telegram.User attribute*), 636
HAS_MEDIA_SPOILER (*in module telegram.ext.filters*), 890
has_media_spoiler (*telegram.ExternalReplyInfo attribute*), 381
has_media_spoiler (*telegram.Message attribute*), 485
has_private_forwards (*telegram.ChatFullInfo attribute*), 310
HAS_PROTECTED_CONTENT (*in module telegram.ext.filters*), 890
has_protected_content (*telegram.ChatFullInfo attribute*), 312
has_protected_content (*telegram.Message attribute*), 477
has_public_winners (*telegram.Giveaway attribute*), 395
has_restricted_voice_and_video_messages (*telegram.ChatFullInfo attribute*), 310
has_spoiler (*telegram.InputMediaAnimation attribute*), 419
has_spoiler (*telegram.InputMediaLivePhoto attribute*), 426
has_spoiler (*telegram.InputMediaPhoto attribute*), 428
has_spoiler (*telegram.InputMediaVideo attribute*), 434
has_topics_enabled (*telegram.User attribute*), 636
has_visible_history (*telegram.ChatFullInfo attribute*), 312
hash (*telegram.DataCredentials attribute*), 781
hash (*telegram.EncryptedCredentials attribute*), 781
hash (*telegram.EncryptedPassportElement attribute*), 783
hash (*telegram.FileCredentials attribute*), 785
HASHTAG (*telegram.constants.MessageEntityType attribute*), 997
HASHTAG (*telegram.MessageEntity attribute*), 534
heading (*telegram.InlineQueryResultLocation attribute*), 723
heading (*telegram.InputLocationMessageContent attribute*), 743
heading (*telegram.Location attribute*), 458
HEAR_NO_EVIL_MONKEY (*telegram.constants.ReactionEmoji attribute*), 1017
HEART_ON_FIRE (*telegram.constants.ReactionEmoji attribute*), 1017
HEART_WITH_ARROW (*telegram.constants.ReactionEmoji attribute*), 1017
height (*telegram.Animation attribute*), 221
height (*telegram.InputMediaAnimation attribute*), 419
height (*telegram.InputMediaVideo attribute*), 434
height (*telegram.InputPaidMediaVideo attribute*), 439
height (*telegram.LivePhoto attribute*), 456
height (*telegram.PaidMediaPreview attribute*), 555
height (*telegram.PhotoSize attribute*), 558
height (*telegram.Sticker attribute*), 686
height (*telegram.Video attribute*), 666
height (*telegram.VideoQuality attribute*), 672
height_percentage (*telegram.StoryAreaPosition attribute*), 595
HIDDEN_USER (*telegram.constants.MessageOriginType attribute*), 999
HIDDEN_USER (*telegram.MessageOrigin attribute*), 538
hide_general_forum_topic() (*telegram.Bot method*), 101
hide_general_forum_topic() (*telegram.Chat method*), 262
hide_general_forum_topic() (*telegram.ChatFullInfo method*), 325
hideGeneralForumTopic() (*telegram.Bot method*),

101
HIGH_VOLTAGE_SIGN (*telegram.constants.ReactionEmoji attribute*), 1018
HORIZONTAL_ACCURACY (*telegram.constants.LocationLimit attribute*), 989
HORIZONTAL_ACCURACY (*telegram.InlineQueryResultLocation attribute*), 724
horizontal_accuracy (*telegram.InlineQueryResultLocation attribute*), 723
HORIZONTAL_ACCURACY (*telegram.InputLocationMessageContent attribute*), 744
horizontal_accuracy (*telegram.InputLocationMessageContent attribute*), 743
horizontal_accuracy (*telegram.InputMediaLocation attribute*), 427
HORIZONTAL_ACCURACY (*telegram.Location attribute*), 458
horizontal_accuracy (*telegram.Location attribute*), 457
HOT_DOG (*telegram.constants.ReactionEmoji attribute*), 1018
HTML (*telegram.constants.ParseMode attribute*), 1010
http_version (*telegram.request.HTTPXRequest property*), 1049
http_version() (*telegram.ext.ApplicationBuilder method*), 828
HTTPXRequest (*class in telegram.request*), 1047
HUGGING_FACE (*telegram.constants.ReactionEmoji attribute*), 1018
HUNDRED_POINTS_SYMBOL (*telegram.constants.ReactionEmoji attribute*), 1018

I

icon_color (*telegram.ForumTopic attribute*), 388
icon_color (*telegram.ForumTopicCreated attribute*), 389
icon_custom_emoji_id (*telegram.ForumTopic attribute*), 388
icon_custom_emoji_id (*telegram.ForumTopicCreated attribute*), 389
icon_custom_emoji_id (*telegram.ForumTopicEdited attribute*), 390
icon_custom_emoji_id (*telegram.InlineKeyboardButton attribute*), 406
icon_custom_emoji_id (*telegram.KeyboardButton attribute*), 448
id (*telegram.Bot property*), 102
id (*telegram.BusinessConnection attribute*), 235
id (*telegram.CallbackQuery attribute*), 242
id (*telegram.Chat attribute*), 250
id (*telegram.ChatFullInfo attribute*), 307
id (*telegram.ChecklistTask attribute*), 287
id (*telegram.Gift attribute*), 679
id (*telegram.InlineQuery attribute*), 692
id (*telegram.InlineQueryResult attribute*), 694
id (*telegram.InlineQueryResultArticle attribute*), 695
id (*telegram.InlineQueryResultAudio attribute*), 697
id (*telegram.InlineQueryResultCachedAudio attribute*), 699
id (*telegram.InlineQueryResultCachedDocument attribute*), 701
id (*telegram.InlineQueryResultCachedGif attribute*), 703
id (*telegram.InlineQueryResultCachedMpeg4Gif attribute*), 705
id (*telegram.InlineQueryResultCachedPhoto attribute*), 707
id (*telegram.InlineQueryResultCachedSticker attribute*), 709
id (*telegram.InlineQueryResultCachedVideo attribute*), 710
id (*telegram.InlineQueryResultCachedVoice attribute*), 712
id (*telegram.InlineQueryResultContact attribute*), 714
id (*telegram.InlineQueryResultDocument attribute*), 717
id (*telegram.InlineQueryResultGame attribute*), 718
id (*telegram.InlineQueryResultGif attribute*), 720
id (*telegram.InlineQueryResultLocation attribute*), 723
id (*telegram.InlineQueryResultMpeg4Gif attribute*), 726
id (*telegram.InlineQueryResultPhoto attribute*), 729
id (*telegram.InlineQueryResultVenue attribute*), 733
id (*telegram.InlineQueryResultVideo attribute*), 736
id (*telegram.InlineQueryResultVoice attribute*), 738
id (*telegram.InputChecklistTask attribute*), 412
id (*telegram.Message property*), 503
id (*telegram.Poll attribute*), 561
id (*telegram.PreCheckoutQuery attribute*), 758
id (*telegram.PreparedInlineMessage attribute*), 752
id (*telegram.PreparedKeyboardButton attribute*), 576
id (*telegram.ShippingOption attribute*), 763
id (*telegram.ShippingQuery attribute*), 764
id (*telegram.StarTransaction attribute*), 766
id (*telegram.Story attribute*), 593
id (*telegram.User attribute*), 634
IdDocumentData (*class in telegram*), 786
identity_card (*telegram.SecureData attribute*), 800
IMAGE (*telegram.ext.filters.Document attribute*), 900
InaccessibleMessage (*class in telegram*), 400
initialize() (*telegram.Bot method*), 102
initialize() (*telegram.ext.AIORateLimiter method*), 959
initialize() (*telegram.ext.Application method*), 810
initialize() (*telegram.ext.BaseRateLimiter method*), 956
initialize() (*telegram.ext.BaseUpdateProcessor method*), 838

initialize() (*telegram.ext.ExtBot method*), 850
initialize() (*telegram.ext.SimpleUpdateProcessor method*), 862
initialize() (*telegram.ext.Updater method*), 864
initialize() (*telegram.request.BaseRequest method*), 1044
initialize() (*telegram.request.HTTPXRequest method*), 1049
inline_keyboard (*telegram.InlineKeyboardMarkup attribute*), 409
inline_message_id (*telegram.CallbackQuery attribute*), 242
inline_message_id (*telegram.ChosenInlineResult attribute*), 690
inline_message_id (*telegram.SentGuestMessage attribute*), 590
inline_message_id (*telegram.SentWebAppMessage attribute*), 590
INLINE_QUERY (*telegram.constants.UpdateType attribute*), 1034
INLINE_QUERY (*telegram.Update attribute*), 629
inline_query (*telegram.Update attribute*), 625
InlineKeyboardButton (*class in telegram*), 400
InlineKeyboardButtonLimit (*class in telegram.constants*), 978
InlineKeyboardMarkup (*class in telegram*), 406
InlineKeyboardMarkupLimit (*class in telegram.constants*), 979
InlineQuery (*class in telegram*), 690
InlineQueryHandler (*class in telegram.ext*), 917
InlineQueryLimit (*class in telegram.constants*), 979
InlineQueryResult (*class in telegram*), 693
InlineQueryResultArticle (*class in telegram*), 694
InlineQueryResultAudio (*class in telegram*), 696
InlineQueryResultCachedAudio (*class in telegram*), 699
InlineQueryResultCachedDocument (*class in telegram*), 700
InlineQueryResultCachedGif (*class in telegram*), 702
InlineQueryResultCachedMpeg4Gif (*class in telegram*), 704
InlineQueryResultCachedPhoto (*class in telegram*), 706
InlineQueryResultCachedSticker (*class in telegram*), 708
InlineQueryResultCachedVideo (*class in telegram*), 710
InlineQueryResultCachedVoice (*class in telegram*), 712
InlineQueryResultContact (*class in telegram*), 713
InlineQueryResultDocument (*class in telegram*), 715
InlineQueryResultGame (*class in telegram*), 718
InlineQueryResultGif (*class in telegram*), 719
InlineQueryResultLimit (*class in telegram.constants*), 980
InlineQueryResultLocation (*class in telegram*), 722
InlineQueryResultMpeg4Gif (*class in telegram*), 725
InlineQueryResultPhoto (*class in telegram*), 728
InlineQueryResultsButton (*class in telegram*), 730
InlineQueryResultsButtonLimit (*class in telegram.constants*), 982
InlineQueryResultType (*class in telegram.constants*), 980
InlineQueryResultVenue (*class in telegram*), 732
InlineQueryResultVideo (*class in telegram*), 734
InlineQueryResultVoice (*class in telegram*), 737
input_field_placeholder (*telegram.ForceReply attribute*), 387
input_field_placeholder (*telegram.ReplyKeyboardMarkup attribute*), 583
input_file_content (*telegram.InputFile attribute*), 414
input_message_content (*telegram.InlineQueryResultArticle attribute*), 696
input_message_content (*telegram.InlineQueryResultAudio attribute*), 698
input_message_content (*telegram.InlineQueryResultCachedAudio attribute*), 700
input_message_content (*telegram.InlineQueryResultCachedDocument attribute*), 702
input_message_content (*telegram.InlineQueryResultCachedGif attribute*), 704
input_message_content (*telegram.InlineQueryResultCachedMpeg4Gif attribute*), 706
input_message_content (*telegram.InlineQueryResultCachedPhoto attribute*), 708
input_message_content (*telegram.InlineQueryResultCachedSticker attribute*), 709
input_message_content (*telegram.InlineQueryResultCachedVideo attribute*), 711
input_message_content (*telegram.InlineQueryResultCachedVoice attribute*), 713
input_message_content (*telegram.InlineQueryResultContact attribute*), 715
input_message_content (*telegram.InlineQueryResultDocument attribute*), 717
input_message_content (*telegram.InlineQueryResultGif attribute*), 722

input_message_content (*telegram.InlineQueryResultLocation attribute*), 724
input_message_content (*telegram.InlineQueryResultMpeg4Gif attribute*), 728
input_message_content (*telegram.InlineQueryResultPhoto attribute*), 730
input_message_content (*telegram.InlineQueryResultVenue attribute*), 734
input_message_content (*telegram.InlineQueryResultVideo attribute*), 737
input_message_content (*telegram.InlineQueryResultVoice attribute*), 739
InputChecklist (*class in telegram*), 410
InputChecklistLimit (*class in telegram.constants*), 982
InputChecklistTask (*class in telegram*), 411
InputContactMessageContent (*class in telegram*), 746
InputFile (*class in telegram*), 412
InputInvoiceMessageContent (*class in telegram*), 747
InputLocationMessageContent (*class in telegram*), 742
InputMedia (*class in telegram*), 415
InputMediaAnimation (*class in telegram*), 416
InputMediaAudio (*class in telegram*), 419
InputMediaDocument (*class in telegram*), 422
InputMediaLivePhoto (*class in telegram*), 424
InputMediaLocation (*class in telegram*), 426
InputMediaPhoto (*class in telegram*), 427
InputMediaSticker (*class in telegram*), 429
InputMediaType (*class in telegram.constants*), 983
InputMediaVenue (*class in telegram*), 430
InputMediaVideo (*class in telegram*), 431
InputMessageContent (*class in telegram*), 739
InputPaidMedia (*class in telegram*), 435
InputPaidMediaLivePhoto (*class in telegram*), 436
InputPaidMediaPhoto (*class in telegram*), 437
InputPaidMediaType (*class in telegram.constants*), 983
InputPaidMediaVideo (*class in telegram*), 438
InputPollMedia (*in module telegram*), 440
InputPollOption (*class in telegram*), 442
InputPollOptionMedia (*in module telegram*), 443
InputProfilePhoto (*class in telegram*), 440
InputProfilePhotoAnimated (*class in telegram*), 441
InputProfilePhotoStatic (*class in telegram*), 441
InputProfilePhotoType (*class in telegram.constants*), 984
InputSticker (*class in telegram*), 681
InputStoryContent (*class in telegram*), 443
InputStoryContentLimit (*class in telegram.constants*), 984
InputStoryContentPhoto (*class in telegram*), 444
InputStoryContentType (*class in telegram.constants*), 985
InputStoryContentVideo (*class in telegram*), 444
InputTextMessageContent (*class in telegram*), 740
InputVenueMessageContent (*class in telegram*), 744
insert_callback_data() (*telegram.ext.ExtBot method*), 851
intensity (*telegram.BackgroundTypePattern attribute*), 294
internal_passport (*telegram.SecureData attribute*), 800
InvalidCallbackData (*class in telegram.ext*), 955
InvalidToken, 1037
invite_link (*telegram.ChatFullInfo attribute*), 311
invite_link (*telegram.ChatInviteLink attribute*), 345
invite_link (*telegram.ChatJoinRequest attribute*), 348
invite_link (*telegram.ChatMemberUpdated attribute*), 364
Invoice (*class in telegram*), 754
INVOICE (*in module telegram.ext.filters*), 890
INVOICE (*telegram.constants.MessageAttachmentType attribute*), 993
INVOICE (*telegram.constants.MessageType attribute*), 1003
invoice (*telegram.ExternalReplyInfo attribute*), 382
invoice (*telegram.Message attribute*), 482
invoice_payload (*telegram.PreCheckoutQuery attribute*), 758
invoice_payload (*telegram.RefundedPayment attribute*), 759
invoice_payload (*telegram.ShippingQuery attribute*), 764
invoice_payload (*telegram.SuccessfulPayment attribute*), 768
invoice_payload (*telegram.TransactionPartnerUser attribute*), 775
INVOICE_PAYMENT (*telegram.constants.TransactionPartnerUser attribute*), 1031
InvoiceLimit (*class in telegram.constants*), 985
ip_address (*telegram.WebhookInfo attribute*), 676
is_access_restricted (*telegram.BotAccessSettings attribute*), 225
is_accessible (*telegram.MaybeInaccessibleMessage property*), 462
is_animated (*telegram.Sticker attribute*), 686
is_animation (*telegram.InputStoryContentVideo attribute*), 445
is_anonymous (*telegram.ChatAdministratorRights attribute*), 282
is_anonymous (*telegram.ChatMemberAdministrator attribute*), 353
is_anonymous (*telegram.ChatMemberOwner at-*

tribute), 358
is_anonymous (*telegram.Poll attribute*), 561
IS_AUTOMATIC_FORWARD (*in module telegram.ext.filters*), 890
is_automatic_forward (*telegram.Message attribute*), 476
is_blurred (*telegram.BackgroundTypeWallpaper attribute*), 293
is_bot (*telegram.User attribute*), 635
is_burned (*telegram.UniqueGift attribute*), 615
is_closed (*telegram.Poll attribute*), 561
is_dark (*telegram.StoryAreaTypeSuggestedReaction attribute*), 598
is_direct_messages (*telegram.Chat attribute*), 250
is_direct_messages (*telegram.ChatFullInfo attribute*), 313
is_disabled (*telegram.LinkPreviewOptions attribute*), 454
is_enabled (*telegram.BusinessConnection attribute*), 236
is_first_recurring (*telegram.SuccessfulPayment attribute*), 769
is_flexible (*telegram.InputInvoiceMessageContent attribute*), 751
is_flipped (*telegram.StoryAreaTypeSuggestedReaction attribute*), 598
is_forum (*telegram.Chat attribute*), 250
is_forum (*telegram.ChatFullInfo attribute*), 308
is_from_blockchain (*telegram.UniqueGift attribute*), 614
IS_FROM_OFFLINE (*in module telegram.ext.filters*), 890
is_from_offline (*telegram.Message attribute*), 477
is_inverted (*telegram.BackgroundTypePattern attribute*), 294
is_manual (*telegram.TextQuote attribute*), 612
is_member (*telegram.ChatMemberRestricted attribute*), 360
is_moving (*telegram.BackgroundTypePattern attribute*), 294
is_moving (*telegram.BackgroundTypeWallpaper attribute*), 293
is_name_implicit (*telegram.ForumTopic attribute*), 388
is_name_implicit (*telegram.ForumTopicCreated attribute*), 389
is_open() (*telegram.BusinessOpeningHours method*), 238
is_paid_post (*telegram.Message attribute*), 488
is_persistent (*telegram.ReplyKeyboardMarkup attribute*), 583
is_premium (*telegram.Gift attribute*), 680
is_premium (*telegram.UniqueGift attribute*), 614
is_premium (*telegram.User attribute*), 635
is_primary (*telegram.ChatInviteLink attribute*), 345
is_private (*telegram.GiftInfo attribute*), 393
is_private (*telegram.OwnedGiftRegular attribute*), 546
is_recurring (*telegram.SuccessfulPayment attribute*), 769
is_revoked (*telegram.ChatInviteLink attribute*), 345
is_saved (*telegram.OwnedGiftRegular attribute*), 546
is_saved (*telegram.OwnedGiftUnique attribute*), 550
is_star_giveaway (*telegram.GiveawayCompleted attribute*), 397
IS_TOPIC_MESSAGE (*in module telegram.ext.filters*), 890
is_topic_message (*telegram.Message attribute*), 484
is_unclaimed (*telegram.ChatBoostSourceGiveaway attribute*), 302
is_upgrade_separate (*telegram.GiftInfo attribute*), 393
is_upgrade_separate (*telegram.OwnedGiftRegular attribute*), 547
is_video (*telegram.Sticker attribute*), 686
ITALIC (*telegram.constants.MessageEntityType attribute*), 997
ITALIC (*telegram.MessageEntity attribute*), 534

J

JACK_O_LANTERN (*telegram.constants.ReactionEmoji attribute*), 1018
Job (*class in telegram.ext*), 851
job (*telegram.ext.CallbackContext attribute*), 841
job (*telegram.ext.Job property*), 854
job_callback() (*telegram.ext.JobQueue static method*), 856
job_queue (*telegram.ext.Application property*), 811
job_queue (*telegram.ext.CallbackContext property*), 843
job_queue() (*telegram.ext.ApplicationBuilder method*), 828
JobQueue (*class in telegram.ext*), 855
jobs() (*telegram.ext.JobQueue method*), 856
join_by_request (*telegram.ChatFullInfo attribute*), 311
join_to_send_messages (*telegram.ChatFullInfo attribute*), 311
JPG (*telegram.ext.filters.Document attribute*), 901
json_parameters (*telegram.request.RequestData property*), 1046
json_payload (*telegram.request.RequestData property*), 1046

K

keyboard (*telegram.ReplyKeyboardMarkup attribute*), 583
KeyboardButton (*class in telegram*), 445
KeyboardButtonPollType (*class in telegram*), 448
KeyboardButtonRequestChat (*class in telegram*), 449
KeyboardButtonRequestManagedBot (*class in telegram*), 451
KeyboardButtonRequestUsers (*class in telegram*), 452
KeyboardButtonRequestUsersLimit (*class in telegram.constants*), 987

KeyboardButtonStyle (*class in telegram.constants*), 988
keywords (*telegram.InputSticker attribute*), 682
KISS_MARK (*telegram.constants.ReactionEmoji attribute*), 1018

L

label (*telegram.LabeledPrice attribute*), 756
LabeledPrice (*class in telegram*), 755
Language (*class in telegram.ext.filters*), 904
language (*telegram.MessageEntity attribute*), 532
language_code (*telegram.User attribute*), 635
last_error_date (*telegram.WebhookInfo attribute*), 676
last_error_message (*telegram.WebhookInfo attribute*), 676
last_name (*telegram.Bot property*), 102
last_name (*telegram.Chat attribute*), 250
last_name (*telegram.ChatFullInfo attribute*), 308
last_name (*telegram.Contact attribute*), 373
last_name (*telegram.InlineQueryResultContact attribute*), 715
last_name (*telegram.InputContactMessageContent attribute*), 747
last_name (*telegram.PersonalDetails attribute*), 798
last_name (*telegram.SharedUser attribute*), 591
last_name (*telegram.User attribute*), 635
last_name_native (*telegram.PersonalDetails attribute*), 798
last_resale_amount (*telegram.UniqueGiftInfo attribute*), 619
last_resale_currency (*telegram.UniqueGiftInfo attribute*), 619
last_synchronization_error_date (*telegram.WebhookInfo attribute*), 677
latitude (*telegram.InlineQueryResultLocation attribute*), 723
latitude (*telegram.InlineQueryResultVenue attribute*), 733
latitude (*telegram.InputLocationMessageContent attribute*), 743
latitude (*telegram.InputMediaLocation attribute*), 426
latitude (*telegram.InputMediaVenue attribute*), 430
latitude (*telegram.InputVenueMessageContent attribute*), 745
latitude (*telegram.Location attribute*), 457
latitude (*telegram.StoryAreaTypeLocation attribute*), 597
leave() (*telegram.Chat method*), 262
leave() (*telegram.ChatFullInfo method*), 325
leave_chat() (*telegram.Bot method*), 102
leaveChat() (*telegram.Bot method*), 102
LEFT (*telegram.ChatMember attribute*), 350
LEFT (*telegram.constants.ChatMemberStatus attribute*), 972
LEFT_CHAT_MEMBER (*telegram.constants.MessageType attribute*), 1004
LEFT_CHAT_MEMBER (*telegram.ext.filters.StatusUpdate attribute*), 909
left_chat_member (*telegram.Message attribute*), 481
LEGENDARY (*telegram.constants.UniqueGiftModelRarity attribute*), 1032
length (*telegram.MessageEntity attribute*), 532
length (*telegram.VideoNote attribute*), 670
level (*telegram.UserRating attribute*), 662
light_theme_main_color (*telegram.UniqueGiftColors attribute*), 616
light_theme_other_colors (*telegram.UniqueGiftColors attribute*), 616
limited_gifts (*telegram.AcceptedGiftTypes attribute*), 219
link (*telegram.Bot property*), 103
link (*telegram.Chat property*), 262
link (*telegram.ChatFullInfo property*), 325
link (*telegram.ChatShared property*), 372
LINK (*telegram.constants.StoryAreaTypeType attribute*), 1027
link (*telegram.Message property*), 503
link (*telegram.SharedUser property*), 592
LINK (*telegram.StoryAreaType attribute*), 596
link (*telegram.User property*), 643
link_preview_options (*telegram.ext.Defaults property*), 848
link_preview_options (*telegram.ExternalReplyInfo attribute*), 380
link_preview_options (*telegram.InputTextMessageContent attribute*), 742
link_preview_options (*telegram.Message attribute*), 477
linked_chat_id (*telegram.ChatFullInfo attribute*), 312
LinkPreviewOptions (*class in telegram*), 453
live_period (*telegram.InlineQueryResultLocation attribute*), 723
live_period (*telegram.InputLocationMessageContent attribute*), 743
live_period (*telegram.Location attribute*), 458
LIVE_PERIOD_FOREVER (*telegram.constants.LocationLimit attribute*), 989
LIVE_PHOTO (*in module telegram.ext.filters*), 890
LIVE_PHOTO (*telegram.constants.BaseInputMediaType attribute*), 964
LIVE_PHOTO (*telegram.constants.InputMediaType attribute*), 983
LIVE_PHOTO (*telegram.constants.InputPaidMediaType attribute*), 984
LIVE_PHOTO (*telegram.constants.MessageAttachmentType attribute*), 993
LIVE_PHOTO (*telegram.constants.MessageType attribute*), 1004
LIVE_PHOTO (*telegram.constants.PaidMediaType attribute*), 1009
live_photo (*telegram.ExternalReplyInfo attribute*),

382
LIVE_PHOTO (*telegram.InputPaidMedia attribute*), 436
live_photo (*telegram.Message attribute*), 490
LIVE_PHOTO (*telegram.PaidMedia attribute*), 551
live_photo (*telegram.PaidMediaLivePhoto attribute*), 553
live_photo (*telegram.PollMedia attribute*), 569
LivePhoto (*class in telegram*), 455
load_persistence_data() (*telegram.ext.CallbackDataCache method*), 954
local_mode (*telegram.Bot property*), 103
local_mode() (*telegram.ext.ApplicationBuilder method*), 829
LOCALIZED_WEEKDAY (*telegram.constants.MessageEntityDateTimeFormats attribute*), 994
LOCALIZED_WEEKDAY_LONG_DATE (*telegram.constants.MessageEntityDateTimeFormats attribute*), 994
LOCALIZED_WEEKDAY_LONG_DATE_LONG_TIME (*telegram.constants.MessageEntityDateTimeFormats attribute*), 994
LOCALIZED_WEEKDAY_LONG_DATE_SHORT_TIME (*telegram.constants.MessageEntityDateTimeFormats attribute*), 995
LOCALIZED_WEEKDAY_LONG_TIME (*telegram.constants.MessageEntityDateTimeFormats attribute*), 995
LOCALIZED_WEEKDAY_SHORT_DATE (*telegram.constants.MessageEntityDateTimeFormats attribute*), 995
LOCALIZED_WEEKDAY_SHORT_DATE_LONG_TIME (*telegram.constants.MessageEntityDateTimeFormats attribute*), 995
LOCALIZED_WEEKDAY_SHORT_DATE_SHORT_TIME (*telegram.constants.MessageEntityDateTimeFormats attribute*), 995
LOCALIZED_WEEKDAY_SHORT_TIME (*telegram.constants.MessageEntityDateTimeFormats attribute*), 995
Location (*class in telegram*), 456
LOCATION (*in module telegram.ext.filters*), 890
location (*telegram.BusinessLocation attribute*), 237
location (*telegram.ChatFullInfo attribute*), 313
location (*telegram.ChatLocation attribute*), 349
location (*telegram.ChosenInlineResult attribute*), 690
LOCATION (*telegram.constants.BaseInputMediaType attribute*), 964
LOCATION (*telegram.constants.InlineQueryResultType attribute*), 981
LOCATION (*telegram.constants.MessageAttachmentType attribute*), 993
LOCATION (*telegram.constants.MessageType attribute*), 1004
LOCATION (*telegram.constants.StoryAreaTypeType attribute*), 1027
location (*telegram.ExternalReplyInfo attribute*), 382
location (*telegram.InlineQuery attribute*), 693
location (*telegram.Message attribute*), 480
location (*telegram.PollMedia attribute*), 569
LOCATION (*telegram.StoryAreaType attribute*), 596
location (*telegram.Venue attribute*), 664
LocationAddress (*class in telegram*), 458
LocationLimit (*class in telegram.constants*), 988
log_out() (*telegram.Bot method*), 103
login_url (*telegram.InlineKeyboardButton attribute*), 404
LoginUrl (*class in telegram*), 459
logOut() (*telegram.Bot method*), 103
LONG_DATE (*telegram.constants.MessageEntityDateTimeFormats attribute*), 995
LONG_DATE_LONG_TIME (*telegram.constants.MessageEntityDateTimeFormats attribute*), 995
LONG_DATE_SHORT_TIME (*telegram.constants.MessageEntityDateTimeFormats attribute*), 995
LONG_TIME (*telegram.constants.MessageEntityDateTimeFormats attribute*), 995
longitude (*telegram.InlineQueryResultLocation attribute*), 723
longitude (*telegram.InlineQueryResultVenue attribute*), 733
longitude (*telegram.InputLocationMessageContent attribute*), 743
longitude (*telegram.InputMediaLocation attribute*), 426
longitude (*telegram.InputMediaVenue attribute*), 430
longitude (*telegram.InputVenueMessageContent attribute*), 745
longitude (*telegram.Location attribute*), 457
longitude (*telegram.StoryAreaTypeLocation attribute*), 597
LOUDLY_CRYING_FACE (*telegram.constants.ReactionEmoji attribute*), 1018

M

main_frame_timestamp (*telegram.InputProfilePhotoAnimated attribute*), 441
MAN_SHRUGGING (*telegram.constants.ReactionEmoji attribute*), 1018
MAN_TECHNOLOGIST (*telegram.constants.ReactionEmoji attribute*), 1018
MANAGED_BOT (*telegram.constants.UpdateType attribute*), 1034
MANAGED_BOT (*telegram.Update attribute*), 629
managed_bot (*telegram.Update attribute*), 627
MANAGED_BOT_CREATED (*telegram.constants.MessageType attribute*), 1004
MANAGED_BOT_CREATED (*telegram.ext.filters.StatusUpdate attribute*),

909
managed_bot_created (*telegram.Message attribute*), 490
ManagedBotAccessLimit (*class in telegram.constants*), 991
ManagedBotCreated (*class in telegram*), 460
ManagedBotUpdated (*class in telegram*), 461
ManagedBotUpdatedHandler (*class in telegram.ext*), 919
map_to_parent (*telegram.ext.ConversationHandler property*), 888
mark_data_for_update_persistence() (*telegram.ext.Application method*), 811
MARKDOWN (*telegram.constants.ParseMode attribute*), 1010
MARKDOWN_V2 (*telegram.constants.ParseMode attribute*), 1010
marked_as_done_task_ids (*telegram.ChecklistTasksDone attribute*), 290
marked_as_not_done_task_ids (*telegram.ChecklistTasksDone attribute*), 290
MASK (*telegram.constants.StickerType attribute*), 1026
MASK (*telegram.Sticker attribute*), 687
mask_position (*telegram.InputSticker attribute*), 682
mask_position (*telegram.Sticker attribute*), 686
MaskPosition (*class in telegram*), 682
MaskPosition (*class in telegram.constants*), 991
match (*telegram.ext.CallbackContext property*), 843
matches (*telegram.ext.CallbackContext attribute*), 840
MAX_ADDRESS (*telegram.ChatLocation attribute*), 349
MAX_ALLOWED_USERS (*telegram.constants.ManagedBotAccessLimit attribute*), 991
MAX_AMOUNT (*telegram.constants.NanostarLimit attribute*), 1009
MAX_ANIMATED_STICKERS (*telegram.constants.StickerSetLimit attribute*), 1025
MAX_ANIMATED_THUMBNAIL_SIZE (*telegram.constants.StickerSetLimit attribute*), 1025
MAX_ANSWER_TEXT_LENGTH (*telegram.CallbackQuery attribute*), 242
MAX_BIO_LENGTH (*telegram.constants.BusinessLimit attribute*), 967
MAX_CALLBACK_DATA (*telegram.constants.InlineKeyboardButtonLimit attribute*), 979
MAX_CALLBACK_DATA (*telegram.InlineKeyboardButton attribute*), 406
MAX_CHAT_LOCATION_ADDRESS (*telegram.constants.LocationLimit attribute*), 989
MAX_CHAT_TITLE_LENGTH (*telegram.constants.ChatLimit attribute*), 971
MAX_COMMAND (*telegram.BotCommand attribute*), 226
MAX_COMMAND (*telegram.constants.BotCommandLimit attribute*), 964
MAX_COMMAND_NUMBER (*telegram.constants.BotCommandLimit attribute*), 965
MAX_COMMENT_LENGTH (*telegram.constants.SuggestedPost attribute*), 1028
max_concurrent_updates (*telegram.ext.BaseUpdateProcessor property*), 839
max_connections (*telegram.WebhookInfo attribute*), 676
MAX_CONNECTIONS_LIMIT (*telegram.constants.WebhookLimit attribute*), 1036
MAX_COPY_TEXT (*telegram.constants.InlineKeyboardButtonLimit attribute*), 979
MAX_COUNTRY_CODES (*telegram.constants.PollLimit attribute*), 1011
MAX_DESCRIPTION (*telegram.BotCommand attribute*), 226
MAX_DESCRIPTION (*telegram.constants.BotCommandLimit attribute*), 965
MAX_DESCRIPTION_CHARACTERS (*telegram.constants.PollLimit attribute*), 1011
MAX_DESCRIPTION_CHARACTERS (*telegram.Poll attribute*), 563
MAX_DESCRIPTION_LENGTH (*telegram.constants.BotDescriptionLimit attribute*), 966
MAX_DESCRIPTION_LENGTH (*telegram.constants.InvoiceLimit attribute*), 986
MAX_DESCRIPTION_LENGTH (*telegram.Invoice attribute*), 755
MAX_DIMMING (*telegram.constants.BackgroundTypeLimit attribute*), 963
MAX_EMOJI_STICKERS (*telegram.constants.StickerSetLimit attribute*), 1025
MAX_EXPLANATION_LENGTH (*telegram.constants.PollLimit attribute*), 1011
MAX_EXPLANATION_LENGTH (*telegram.Poll attribute*), 563
MAX_EXPLANATION_LINE_FEEDS (*telegram.constants.PollLimit attribute*), 1011
MAX_EXPLANATION_LINE_FEEDS (*telegram.Poll attribute*), 563
MAX_GIFT_RESULTS (*telegram.constants.BusinessLimit attribute*), 967
MAX_HEADING (*telegram.constants.LocationLimit attribute*), 989
MAX_HEADING (*telegram.InlineQueryResultLocation attribute*), 724
MAX_HEADING (*telegram.InputLocationMessageContent attribute*), 744

MAX_HEADING (*telegram.Location attribute*), 458
MAX_ID_LENGTH (*telegram.constants.InlineQueryResultLimit attribute*), 980
MAX_ID_LENGTH (*telegram.InlineQueryResult attribute*), 694
MAX_INITIAL_STICKERS (*telegram.constants.StickerSetLimit attribute*), 1025
MAX_INPUT_FIELD_PLACEHOLDER (*telegram.constants.ReplyLimit attribute*), 1022
MAX_INPUT_FIELD_PLACEHOLDER (*telegram.ForceReply attribute*), 387
MAX_INPUT_FIELD_PLACEHOLDER (*telegram.ReplyKeyboardMarkup attribute*), 583
MAX_INTENSITY (*telegram.constants.BackgroundTypeLimit attribute*), 963
MAX_KEYWORD_LENGTH (*telegram.constants.StickerLimit attribute*), 1024
MAX_LENGTH (*telegram.BotName attribute*), 232
MAX_LENGTH (*telegram.PollOption attribute*), 572
MAX_LIMIT (*telegram.constants.BulkRequestLimit attribute*), 966
MAX_LIMIT (*telegram.constants.PersonalChatMessagesLimit attribute*), 1010
MAX_LIMIT (*telegram.constants.PollingLimit attribute*), 1013
MAX_LIMIT (*telegram.constants.StarTransactionsLimit attribute*), 1023
MAX_LIMIT (*telegram.constants.UserProfileAudiosLimit attribute*), 1035
MAX_LIMIT (*telegram.constants.UserProfilePhotosLimit attribute*), 1035
MAX_LINK_AREAS (*telegram.constants.StoryAreaTypeLimit attribute*), 1027
MAX_LIVE_PERIOD (*telegram.constants.LocationLimit attribute*), 989
MAX_LIVE_PERIOD (*telegram.InlineQueryResultLocation attribute*), 724
MAX_LIVE_PERIOD (*telegram.InputLocationMessageContent attribute*), 744
MAX_LOCATION_AREAS (*telegram.constants.StoryAreaTypeLimit attribute*), 1027
MAX_MEDIA_LENGTH (*telegram.constants.MediaGroupLimit attribute*), 992
MAX_MEMBER_LIMIT (*telegram.constants.ChatInviteLinkLimit attribute*), 971
MAX_NAME_AND_TITLE (*telegram.constants.StickerLimit attribute*), 1024
MAX_NAME_LENGTH (*telegram.constants.BotNameLimit attribute*), 966
MAX_NAME_LENGTH (*telegram.constants.BusinessLimit attribute*), 967
MAX_NAME_LENGTH (*telegram.constants.ForumTopicLimit attribute*), 978
MAX_OFFSET_LENGTH (*telegram.constants.InlineQueryLimit attribute*), 980
MAX_OFFSET_LENGTH (*telegram.InlineQuery attribute*), 693
MAX_OPEN_PERIOD (*telegram.constants.PollLimit attribute*), 1011
MAX_OPEN_PERIOD (*telegram.Poll attribute*), 563
MAX_OPTION_LENGTH (*telegram.constants.PollLimit attribute*), 1011
MAX_OPTION_LENGTH (*telegram.Poll attribute*), 564
MAX_OPTION_NUMBER (*telegram.constants.PollLimit attribute*), 1011
MAX_OPTION_NUMBER (*telegram.Poll attribute*), 564
MAX_PAYLOAD_LENGTH (*telegram.constants.InvoiceLimit attribute*), 986
MAX_PAYLOAD_LENGTH (*telegram.Invoice attribute*), 755
MAX_PRICE (*telegram.constants.ChatSubscriptionLimit attribute*), 973
MAX_PRICE_NANOTONCOINS (*telegram.constants.SuggestedPost attribute*), 1028
MAX_PRICE_STARS (*telegram.constants.SuggestedPost attribute*), 1029
MAX_PROXIMITY_ALERT_RADIUS (*telegram.constants.LocationLimit attribute*), 990
MAX_PROXIMITY_ALERT_RADIUS (*telegram.InlineQueryResultLocation attribute*), 724
MAX_PROXIMITY_ALERT_RADIUS (*telegram.InputLocationMessageContent attribute*), 744
MAX_QUANTITY (*telegram.constants.KeyboardButtonRequestUsersLimit attribute*), 988
max_quantity (*telegram.KeyboardButtonRequestUsers attribute*), 453
MAX_QUERY_LENGTH (*telegram.constants.InlineQueryLimit attribute*), 980
MAX_QUERY_LENGTH (*telegram.InlineQuery attribute*), 693
MAX_QUESTION_LENGTH (*telegram.constants.PollLimit attribute*), 1011
MAX_QUESTION_LENGTH (*telegram.Poll attribute*), 564
max_reaction_count (*telegram.ChatFullInfo at-*

tribute), 308
MAX_RESULTS (*telegram.InlineQuery attribute*), 693
MAX_ROTATION_ANGLE (*telegram.constants.BackgroundFillLimit attribute*), 962
MAX_ROTATION_ANGLE (*telegram.constants.StoryAreaPositionLimit attribute*), 1026
MAX_SEARCH_KEYWORDS (*telegram.constants.StickerLimit attribute*), 1024
MAX_SECRET_TOKEN_LENGTH (*telegram.constants.WebhookLimit attribute*), 1036
MAX_SEND_DATE (*telegram.constants.SuggestedPost attribute*), 1029
MAX_SHORT_DESCRIPTION_LENGTH (*telegram.constants.BotDescriptionLimit attribute*), 966
MAX_STAR_COUNT (*telegram.constants.BusinessLimit attribute*), 967
MAX_STAR_COUNT (*telegram.constants.InvoiceLimit attribute*), 986
MAX_START_PARAMETER_LENGTH (*telegram.constants.InlineQueryResultsButtonLimit attribute*), 982
MAX_START_PARAMETER_LENGTH (*telegram.InlineQueryResultsButton attribute*), 731
MAX_STATIC_STICKERS (*telegram.constants.StickerSetLimit attribute*), 1025
MAX_STATIC_THUMBNAIL_SIZE (*telegram.constants.StickerSetLimit attribute*), 1025
MAX_STICKER_EMOJI (*telegram.constants.StickerLimit attribute*), 1024
MAX_SUGGESTED_REACTION_AREAS (*telegram.constants.StoryAreaTypeLimit attribute*), 1027
MAX_TAG_LENGTH (*telegram.constants.TagLimit attribute*), 1030
MAX_TASK_NUMBER (*telegram.constants.InputChecklistLimit attribute*), 982
MAX_TEXT_LENGTH (*telegram.constants.GiftLimit attribute*), 978
MAX_TEXT_LENGTH (*telegram.constants.InputChecklistLimit attribute*), 982
MAX_TEXT_LENGTH (*telegram.constants.MessageLimit attribute*), 999
MAX_TEXT_LENGTH (*telegram.constants.PremiumSubscription attribute*), 1013
MAX_TEXT_LENGTH (*telegram.constants.VerifyLimit attribute*), 1036
max_tip_amount (*telegram.InputInvoiceMessageContent attribute*), 750
MAX_TIP_AMOUNTS (*telegram.constants.InvoiceLimit attribute*), 986
MAX_TIP_AMOUNTS (*telegram.Invoice attribute*), 755
MAX_TITLE_LENGTH (*telegram.constants.InputChecklistLimit attribute*), 982
MAX_TITLE_LENGTH (*telegram.constants.InvoiceLimit attribute*), 986
MAX_TITLE_LENGTH (*telegram.Invoice attribute*), 755
MAX_UNIQUE_GIFT_AREAS (*telegram.constants.StoryAreaTypeLimit attribute*), 1027
MAX_USERNAME_LENGTH (*telegram.constants.BusinessLimit attribute*), 967
MAX_VALUE_BASKETBALL (*telegram.constants.DiceLimit attribute*), 975
MAX_VALUE_BASKETBALL (*telegram.Dice attribute*), 374
MAX_VALUE_BOWLING (*telegram.constants.DiceLimit attribute*), 975
MAX_VALUE_BOWLING (*telegram.Dice attribute*), 375
MAX_VALUE_DARTS (*telegram.constants.DiceLimit attribute*), 975
MAX_VALUE_DARTS (*telegram.Dice attribute*), 375
MAX_VALUE_DICE (*telegram.constants.DiceLimit attribute*), 975
MAX_VALUE_DICE (*telegram.Dice attribute*), 375
MAX_VALUE_FOOTBALL (*telegram.constants.DiceLimit attribute*), 975
MAX_VALUE_FOOTBALL (*telegram.Dice attribute*), 375
MAX_VALUE_SLOT_MACHINE (*telegram.constants.DiceLimit attribute*), 975
MAX_VALUE_SLOT_MACHINE (*telegram.Dice attribute*), 375
MAX_VIDEO_DURATION (*telegram.constants.InputStoryContentLimit attribute*), 984
MAX_WEATHER_AREAS (*telegram.constants.StoryAreaTypeLimit attribute*), 1027
MAX_WINNERS (*telegram.constants.GiveawayLimit attribute*), 978
maxsize (*telegram.ext.CallbackDataCache property*), 954
MaybeInaccessibleMessage (*class in telegram*), 461
media (*telegram.InputMedia attribute*), 416
media (*telegram.InputMediaAnimation attribute*), 418
media (*telegram.InputMediaAudio attribute*), 421
media (*telegram.InputMediaDocument attribute*), 423
media (*telegram.InputMediaLivePhoto attribute*), 425
media (*telegram.InputMediaPhoto attribute*), 428
media (*telegram.InputMediaSticker attribute*), 429
media (*telegram.InputMediaVideo attribute*), 433
media (*telegram.InputPaidMedia attribute*), 435
media (*telegram.InputPaidMediaLivePhoto attribute*),

436
media (*telegram.InputPaidMediaPhoto attribute*), 437
media (*telegram.InputPaidMediaVideo attribute*), 439
media (*telegram.InputPollOption attribute*), 443
media (*telegram.Poll attribute*), 563
media (*telegram.PollOption attribute*), 571
media_group_id (*telegram.Message attribute*), 477
media_write_timeout() (*telegram.ext.ApplicationBuilder method*), 829
MediaGroupLimit (*class in telegram.constants*), 991
MEMBER (*telegram.ChatMember attribute*), 350
MEMBER (*telegram.constants.ChatMemberStatus attribute*), 972
member_limit (*telegram.ChatInviteLink attribute*), 345
members_only (*telegram.Poll attribute*), 562
Mention (*class in telegram.ext.filters*), 904
MENTION (*telegram.constants.MessageEntityType attribute*), 997
MENTION (*telegram.MessageEntity attribute*), 534
mention_button() (*telegram.User method*), 644
mention_html() (*in module telegram.helpers*), 1041
mention_html() (*telegram.Chat method*), 262
mention_html() (*telegram.ChatFullInfo method*), 325
mention_html() (*telegram.User method*), 644
mention_markdown() (*in module telegram.helpers*), 1041
mention_markdown() (*telegram.Chat method*), 262
mention_markdown() (*telegram.ChatFullInfo method*), 325
mention_markdown() (*telegram.User method*), 644
mention_markdown_v2() (*telegram.Chat method*), 263
mention_markdown_v2() (*telegram.ChatFullInfo method*), 326
mention_markdown_v2() (*telegram.User method*), 644
MenuButton (*class in telegram*), 462
MenuButtonCommands (*class in telegram*), 463
MenuButtonDefault (*class in telegram*), 464
MenuButtonType (*class in telegram.constants*), 992
MenuButtonWebApp (*class in telegram*), 464
Message (*class in telegram*), 465
message (*telegram.BusinessIntro attribute*), 237
message (*telegram.CallbackQuery attribute*), 242
MESSAGE (*telegram.constants.UpdateType attribute*), 1034
MESSAGE (*telegram.ext.filters.UpdateType attribute*), 914
message (*telegram.PassportElementError attribute*), 788
message (*telegram.PassportElementErrorDataField attribute*), 789
message (*telegram.PassportElementErrorFile attribute*), 789
message (*telegram.PassportElementErrorFiles attribute*), 790
message (*telegram.PassportElementErrorFrontSide attribute*), 791
message (*telegram.PassportElementErrorReverseSide attribute*), 792
message (*telegram.PassportElementErrorSelfie attribute*), 792
message (*telegram.PassportElementErrorTranslationFile attribute*), 793
message (*telegram.PassportElementErrorTranslationFiles attribute*), 794
message (*telegram.PassportElementErrorUnspecified attribute*), 795
MESSAGE (*telegram.Update attribute*), 629
message (*telegram.Update attribute*), 624
message (*telegram.warnings.PTBDeprecationWarning attribute*), 1050
message_auto_delete_time (*telegram.ChatFullInfo attribute*), 311
message_auto_delete_time (*telegram.MessageAutoDeleteTimerChanged attribute*), 529
MESSAGE_AUTO_DELETE_TIMER_CHANGED (*telegram.constants.MessageType attribute*), 1004
MESSAGE_AUTO_DELETE_TIMER_CHANGED (*telegram.ext.filters.StatusUpdate attribute*), 909
message_auto_delete_timer_changed (*telegram.Message attribute*), 481
MESSAGE_ENTITIES (*telegram.constants.MessageLimit attribute*), 999
message_id (*telegram.ExternalReplyInfo attribute*), 380
message_id (*telegram.InaccessibleMessage attribute*), 400
message_id (*telegram.MaybeInaccessibleMessage attribute*), 462
message_id (*telegram.Message attribute*), 476
message_id (*telegram.MessageId attribute*), 537
message_id (*telegram.MessageOriginChannel attribute*), 539
message_id (*telegram.MessageReactionCountUpdated attribute*), 542
message_id (*telegram.MessageReactionUpdated attribute*), 543
message_id (*telegram.ReplyParameters attribute*), 589
message_ids (*telegram.BusinessMessagesDeleted attribute*), 240
MESSAGE_REACTION (*telegram.constants.UpdateType attribute*), 1034
MESSAGE_REACTION (*telegram.ext.MessageReactionHandler attribute*), 923
MESSAGE_REACTION (*telegram.Update attribute*), 629
message_reaction (*telegram.Update attribute*), 626
MESSAGE_REACTION_COUNT (*telegram.constants.UpdateType attribute*),

1034
MESSAGE_REACTION_COUNT (*telegram.Update attribute*), 629
message_reaction_count (*telegram.Update attribute*), 626
MESSAGE_REACTION_COUNT_UPDATED (*telegram.ext.MessageReactionHandler attribute*), 923
message_reaction_types (*telegram.ext.MessageReactionHandler attribute*), 923
MESSAGE_REACTION_UPDATED (*telegram.ext.MessageReactionHandler attribute*), 924
message_text (*telegram.InputTextMessageContent attribute*), 741
message_thread_id (*telegram.ForumTopic attribute*), 388
message_thread_id (*telegram.Message attribute*), 484
MessageAttachmentType (*class in telegram.constants*), 992
MessageAutoDeleteTimerChanged (*class in telegram*), 529
MessageEntity (*class in telegram*), 529
MessageEntityDateTimeFormats (*class in telegram.constants*), 994
MessageEntityType (*class in telegram.constants*), 996
MessageFilter (*class in telegram.ext.filters*), 904
MessageHandler (*class in telegram.ext*), 920
MessageId (*class in telegram*), 537
MessageLimit (*class in telegram.constants*), 998
MessageOrigin (*class in telegram*), 538
MessageOriginChannel (*class in telegram*), 539
MessageOriginChat (*class in telegram*), 540
MessageOriginHiddenUser (*class in telegram*), 540
MessageOriginType (*class in telegram.constants*), 999
MessageOriginUser (*class in telegram*), 541
MessageReactionCountUpdated (*class in telegram*), 542
MessageReactionHandler (*class in telegram.ext*), 922
MessageReactionUpdated (*class in telegram*), 543
MESSAGES (*telegram.ext.filters.UpdateType attribute*), 914
MESSAGES_PER_MINUTE_PER_GROUP (*telegram.constants.FloodLimit attribute*), 976
MESSAGES_PER_SECOND (*telegram.constants.FloodLimit attribute*), 976
MESSAGES_PER_SECOND_PER_CHAT (*telegram.constants.FloodLimit attribute*), 976
MessageType (*class in telegram.constants*), 1000
middle_name (*telegram.PersonalDetails attribute*), 797
middle_name_native (*telegram.PersonalDetails attribute*), 798
MIGRATE (*telegram.ext.filters.StatusUpdate attribute*), 909
migrate_chat_data() (*telegram.ext.Application method*), 811
migrate_from_chat_id (*telegram.Message attribute*), 482
MIGRATE_TO_CHAT_ID (*telegram.constants.MessageType attribute*), 1004
migrate_to_chat_id (*telegram.Message attribute*), 482
mime_type (*telegram.Animation attribute*), 221
mime_type (*telegram.Audio attribute*), 223
mime_type (*telegram.Document attribute*), 378
mime_type (*telegram.InlineQueryResultDocument attribute*), 717
mime_type (*telegram.InlineQueryResultVideo attribute*), 736
mime_type (*telegram.LivePhoto attribute*), 456
mime_type (*telegram.Video attribute*), 666
mime_type (*telegram.Voice attribute*), 673
mimetype (*telegram.InputFile attribute*), 415
MIN_ADDRESS (*telegram.ChatLocation attribute*), 349
MIN_AMOUNT (*telegram.constants.NanostarLimit attribute*), 1009
MIN_CALLBACK_DATA (*telegram.constants.InlineKeyboardButtonLimit attribute*), 979
MIN_CALLBACK_DATA (*telegram.InlineKeyboardButton attribute*), 406
MIN_CHAT_LOCATION_ADDRESS (*telegram.constants.LocationLimit attribute*), 990
MIN_CHAT_TITLE_LENGTH (*telegram.constants.ChatLimit attribute*), 971
MIN_COMMAND (*telegram.BotCommand attribute*), 226
MIN_COMMAND (*telegram.constants.BotCommandLimit attribute*), 965
MIN_CONNECTIONS_LIMIT (*telegram.constants.WebhookLimit attribute*), 1036
MIN_COPY_TEXT (*telegram.constants.InlineKeyboardButtonLimit attribute*), 979
MIN_DESCRIPTION (*telegram.BotCommand attribute*), 226
MIN_DESCRIPTION (*telegram.constants.BotCommandLimit attribute*), 965
MIN_DESCRIPTION_LENGTH (*telegram.constants.InvoiceLimit attribute*), 987
MIN_DESCRIPTION_LENGTH (*telegram.Invoice attribute*), 755
MIN_GIFT_RESULTS (*tele-*

gram.constants.BusinessLimit attribute), 967
MIN_HEADING (*telegram.constants.LocationLimit attribute*), 990
MIN_HEADING (*telegram.InlineQueryResultLocation attribute*), 725
MIN_HEADING (*telegram.InputLocationMessageContent attribute*), 744
MIN_HEADING (*telegram.Location attribute*), 458
MIN_ID_LENGTH (*telegram.constants.InlineQueryResultLimit attribute*), 980
MIN_ID_LENGTH (*telegram.InlineQueryResult attribute*), 694
MIN_INITIAL_STICKERS (*telegram.constants.StickerSetLimit attribute*), 1026
MIN_INPUT_FIELD_PLACEHOLDER (*telegram.constants.ReplyLimit attribute*), 1022
MIN_INPUT_FIELD_PLACEHOLDER (*telegram.ForceReply attribute*), 387
MIN_INPUT_FIELD_PLACEHOLDER (*telegram.ReplyKeyboardMarkup attribute*), 584
MIN_LENGTH (*telegram.PollOption attribute*), 572
MIN_LIMIT (*telegram.constants.BulkRequestLimit attribute*), 966
MIN_LIMIT (*telegram.constants.PersonalChatMessagesLimit attribute*), 1010
MIN_LIMIT (*telegram.constants.PollingLimit attribute*), 1013
MIN_LIMIT (*telegram.constants.StarTransactionsLimit attribute*), 1023
MIN_LIMIT (*telegram.constants.UserProfileAudiosLimit attribute*), 1035
MIN_LIMIT (*telegram.constants.UserProfilePhotosLimit attribute*), 1035
MIN_LIVE_PERIOD (*telegram.constants.LocationLimit attribute*), 990
MIN_LIVE_PERIOD (*telegram.InlineQueryResultLocation attribute*), 725
MIN_LIVE_PERIOD (*telegram.InputLocationMessageContent attribute*), 744
MIN_MEDIA_LENGTH (*telegram.constants.MediaGroupLimit attribute*), 992
MIN_MEMBER_LIMIT (*telegram.constants.ChatInviteLinkLimit attribute*), 971
MIN_MEMBERSHIP_HOURS (*telegram.constants.PollLimit attribute*), 1012
MIN_MEMBERSHIP_HOURS (*telegram.Poll attribute*), 564
MIN_NAME_AND_TITLE (*telegram.constants.StickerLimit attribute*), 1025
MIN_NAME_LENGTH (*telegram.constants.BusinessLimit attribute*), 968
MIN_NAME_LENGTH (*telegram.constants.ForumTopicLimit attribute*), 978
MIN_OPEN_PERIOD (*telegram.constants.PollLimit attribute*), 1012
MIN_OPEN_PERIOD (*telegram.Poll attribute*), 564
MIN_OPTION_LENGTH (*telegram.constants.PollLimit attribute*), 1012
MIN_OPTION_LENGTH (*telegram.Poll attribute*), 564
MIN_OPTION_NUMBER (*telegram.constants.PollLimit attribute*), 1012
MIN_OPTION_NUMBER (*telegram.Poll attribute*), 564
MIN_PAYLOAD_LENGTH (*telegram.constants.InvoiceLimit attribute*), 987
MIN_PAYLOAD_LENGTH (*telegram.Invoice attribute*), 755
MIN_PRICE (*telegram.constants.ChatSubscriptionLimit attribute*), 973
MIN_PRICE_NANOTONCOINS (*telegram.constants.SuggestedPost attribute*), 1029
MIN_PRICE_STARS (*telegram.constants.SuggestedPost attribute*), 1029
MIN_PROXIMITY_ALERT_RADIUS (*telegram.constants.LocationLimit attribute*), 990
MIN_PROXIMITY_ALERT_RADIUS (*telegram.InlineQueryResultLocation attribute*), 725
MIN_PROXIMITY_ALERT_RADIUS (*telegram.InputLocationMessageContent attribute*), 744
MIN_QUANTITY (*telegram.constants.KeyboardButtonRequestUsersLimit attribute*), 988
MIN_QUESTION_LENGTH (*telegram.constants.PollLimit attribute*), 1012
MIN_QUESTION_LENGTH (*telegram.Poll attribute*), 564
MIN_SECRET_TOKEN_LENGTH (*telegram.constants.WebhookLimit attribute*), 1036
MIN_SEND_DATE (*telegram.constants.SuggestedPost attribute*), 1029
MIN_STAR_COUNT (*telegram.constants.BusinessLimit attribute*), 968
MIN_STAR_COUNT (*telegram.constants.InvoiceLimit attribute*), 987
MIN_START_PARAMETER_LENGTH (*telegram.constants.InlineQueryResultsButtonLimit attribute*), 982
MIN_START_PARAMETER_LENGTH (*telegram.InlineQueryResultsButton attribute*), 731
MIN_STICKER_EMOJI (*telegram.constants.StickerLimit attribute*), 1025

MIN_TASK_NUMBER (*telegram.constants.InputChecklistLimit attribute*), 982
MIN_TEXT_LENGTH (*telegram.constants.InputChecklistLimit attribute*), 983
MIN_TEXT_LENGTH (*telegram.constants.MessageLimit attribute*), 999
MIN_TITLE_LENGTH (*telegram.constants.InputChecklistLimit attribute*), 983
MIN_TITLE_LENGTH (*telegram.constants.InvoiceLimit attribute*), 987
MIN_TITLE_LENGTH (*telegram.Invoice attribute*), 755
MIN_VALUE (*telegram.constants.DiceLimit attribute*), 975
MIN_VALUE (*telegram.Dice attribute*), 375
model (*telegram.UniqueGift attribute*), 614
model_custom_emoji_id (*telegram.UniqueGiftColors attribute*), 616
module
    telegram, 13
    telegram.constants, 960
    telegram.error, 1036
    telegram.ext, 803
    telegram.ext.filters, 888
    telegram.helpers, 1040
    telegram.warnings, 1049
month (*telegram.Birthdate attribute*), 224
MONTH_COUNT_SIX (*telegram.constants.PremiumSubscription attribute*), 1013
MONTH_COUNT_THREE (*telegram.constants.PremiumSubscription attribute*), 1013
MONTH_COUNT_TWELVE (*telegram.constants.PremiumSubscription attribute*), 1013
MOUTH (*telegram.constants.MaskPosition attribute*), 991
MOUTH (*telegram.MaskPosition attribute*), 683
MOYAI (*telegram.constants.ReactionEmoji attribute*), 1018
MP3 (*telegram.ext.filters.Document attribute*), 901
MP4 (*telegram.ext.filters.Document attribute*), 901
mpeg4_duration (*telegram.InlineQueryResultMpeg4Gif attribute*), 727
mpeg4_file_id (*telegram.InlineQueryResultCachedMpeg4Gif attribute*), 705
mpeg4_height (*telegram.InlineQueryResultMpeg4Gif attribute*), 726
mpeg4_url (*telegram.InlineQueryResultMpeg4Gif attribute*), 726
mpeg4_width (*telegram.InlineQueryResultMpeg4Gif attribute*), 726
MPEG4GIF (*telegram.constants.InlineQueryResultType attribute*), 981
multipart_data (*telegram.request.RequestData property*), 1046
MY_CHAT_MEMBER (*telegram.constants.UpdateType attribute*), 1034
MY_CHAT_MEMBER (*telegram.ext.ChatMemberHandler attribute*), 879
MY_CHAT_MEMBER (*telegram.Update attribute*), 629
my_chat_member (*telegram.Update attribute*), 626

N

NAIL_POLISH (*telegram.constants.ReactionEmoji attribute*), 1018
name (*telegram.Bot property*), 104
name (*telegram.BotName attribute*), 232
name (*telegram.ChatInviteLink attribute*), 345
name (*telegram.ext.ConversationHandler property*), 888
name (*telegram.ext.filters.BaseFilter property*), 893
name (*telegram.ext.filters.Chat property*), 896
name (*telegram.ext.filters.ForwardedFrom property*), 903
name (*telegram.ext.filters.SenderChat property*), 907
name (*telegram.ext.filters.User property*), 915
name (*telegram.ext.filters.ViaBot property*), 916
name (*telegram.ext.Job attribute*), 852
name (*telegram.ForumTopic attribute*), 388
name (*telegram.ForumTopicCreated attribute*), 389
name (*telegram.ForumTopicEdited attribute*), 390
name (*telegram.OrderInfo attribute*), 756
name (*telegram.SharedUser property*), 592
name (*telegram.StickerSet attribute*), 688
name (*telegram.StoryAreaTypeUniqueGift attribute*), 598
name (*telegram.UniqueGift attribute*), 614
name (*telegram.UniqueGiftBackdrop attribute*), 617
name (*telegram.UniqueGiftModel attribute*), 620
name (*telegram.UniqueGiftSymbol attribute*), 621
name (*telegram.User property*), 644
NAME_LENGTH (*telegram.constants.ChatInviteLinkLimit attribute*), 971
Nanostar (*class in telegram.constants*), 1008
nanostar_amount (*telegram.AffiliateInfo attribute*), 753
nanostar_amount (*telegram.StarAmount attribute*), 765
nanostar_amount (*telegram.StarTransaction attribute*), 766
NanostarLimit (*class in telegram.constants*), 1008
need_email (*telegram.InputInvoiceMessageContent attribute*), 751
need_name (*telegram.InputInvoiceMessageContent attribute*), 751
need_phone_number (*telegram.InputInvoiceMessageContent attribute*), 751
need_shipping_address (*telegram.InputInvoiceMessageContent attribute*),

751
needs_repainting (*telegram.Sticker attribute*), 687
NERD_FACE (*telegram.constants.ReactionEmoji attribute*), 1019
NetworkError, 1038
NEUTRAL_FACE (*telegram.constants.ReactionEmoji attribute*), 1019
new_chat_id (*telegram.error.ChatMigrated attribute*), 1037
new_chat_member (*telegram.ChatMemberUpdated attribute*), 364
NEW_CHAT_MEMBERS (*telegram.constants.MessageType attribute*), 1004
NEW_CHAT_MEMBERS (*telegram.ext.filters.StatusUpdate attribute*), 909
new_chat_members (*telegram.Message attribute*), 480
NEW_CHAT_PHOTO (*telegram.constants.MessageType attribute*), 1004
NEW_CHAT_PHOTO (*telegram.ext.filters.StatusUpdate attribute*), 909
new_chat_photo (*telegram.Message attribute*), 481
NEW_CHAT_TITLE (*telegram.constants.MessageType attribute*), 1004
NEW_CHAT_TITLE (*telegram.ext.filters.StatusUpdate attribute*), 909
new_chat_title (*telegram.Message attribute*), 481
NEW_MOON_WITH_FACE (*telegram.constants.ReactionEmoji attribute*), 1019
new_owner (*telegram.ChatOwnerChanged attribute*), 365
new_owner (*telegram.ChatOwnerLeft attribute*), 365
new_reaction (*telegram.MessageReactionUpdated attribute*), 544
next_level_rating (*telegram.UserRating attribute*), 662
next_offset (*telegram.OwnedGifts attribute*), 549
next_t (*telegram.ext.Job property*), 854
next_transfer_date (*telegram.OwnedGiftUnique attribute*), 550
next_transfer_date (*telegram.UniqueGiftInfo attribute*), 619
no_permissions() (*telegram.ChatPermissions class method*), 369
no_rights() (*telegram.ChatAdministratorRights class method*), 284
nonce (*telegram.Credentials attribute*), 780
number (*telegram.UniqueGift attribute*), 614

O

OFFER (*telegram.constants.UniqueGiftInfoOrigin attribute*), 1032
OFFER (*telegram.UniqueGiftInfo attribute*), 620
offset (*telegram.InlineQuery attribute*), 692
offset (*telegram.MessageEntity attribute*), 532
OK_HAND_SIGN (*telegram.constants.ReactionEmoji attribute*), 1019
old_chat_member (*telegram.ChatMemberUpdated attribute*), 363
old_reaction (*telegram.MessageReactionUpdated attribute*), 543
on_flush (*telegram.ext.PicklePersistence attribute*), 950
one_time_keyboard (*telegram.ReplyKeyboardMarkup attribute*), 583
only_new_members (*telegram.Giveaway attribute*), 395
only_new_members (*telegram.GiveawayWinners attribute*), 399
open_period (*telegram.Poll attribute*), 562
opening_hours (*telegram.BusinessOpeningHours attribute*), 238
opening_minute (*telegram.BusinessOpeningHoursInterval attribute*), 239
opening_time (*telegram.BusinessOpeningHoursInterval property*), 240
option_ids (*telegram.PollAnswer attribute*), 568
option_persistent_id (*telegram.PollOptionAdded attribute*), 573
option_persistent_id (*telegram.PollOptionDeleted attribute*), 575
option_persistent_ids (*telegram.PollAnswer attribute*), 568
option_text (*telegram.PollOptionAdded attribute*), 573
option_text (*telegram.PollOptionDeleted attribute*), 575
option_text_entities (*telegram.PollOptionAdded attribute*), 573
option_text_entities (*telegram.PollOptionDeleted attribute*), 575
options (*telegram.Poll attribute*), 561
order_info (*telegram.PreCheckoutQuery attribute*), 758
order_info (*telegram.SuccessfulPayment attribute*), 769
OrderInfo (*class in telegram*), 756
origin (*telegram.ExternalReplyInfo attribute*), 380
origin (*telegram.UniqueGiftInfo attribute*), 619
OTHER (*telegram.constants.TransactionPartnerType attribute*), 1030
OTHER (*telegram.TransactionPartner attribute*), 770
others_can_add_tasks (*telegram.Checklist attribute*), 286
others_can_add_tasks (*telegram.InputChecklist attribute*), 411
others_can_mark_tasks_as_done (*telegram.Checklist attribute*), 286
others_can_mark_tasks_as_done (*telegram.InputChecklist attribute*), 411
owned_gift_id (*telegram.GiftInfo attribute*), 392
owned_gift_id (*telegram.OwnedGiftRegular at-*

tribute), 546
owned_gift_id (*telegram.OwnedGiftUnique attribute*), 550
owned_gift_id (*telegram.UniqueGiftInfo attribute*), 619
OwnedGift (*class in telegram*), 544
OwnedGiftRegular (*class in telegram*), 545
OwnedGifts (*class in telegram*), 548
OwnedGiftType (*class in telegram.constants*), 1009
OwnedGiftUnique (*class in telegram*), 549
OWNER (*telegram.ChatMember attribute*), 351
OWNER (*telegram.constants.ChatMemberStatus attribute*), 972

P

PAID (*telegram.constants.ReactionType attribute*), 1022
PAID (*telegram.ReactionType attribute*), 578
PAID_MEDIA (*in module telegram.ext.filters*), 890
PAID_MEDIA (*telegram.constants.MessageAttachmentType attribute*), 993
PAID_MEDIA (*telegram.constants.MessageType attribute*), 1004
paid_media (*telegram.ExternalReplyInfo attribute*), 382
paid_media (*telegram.Message attribute*), 488
paid_media (*telegram.PaidMediaInfo attribute*), 552
paid_media (*telegram.TransactionPartnerUser attribute*), 776
paid_media_payload (*telegram.PaidMediaPurchased attribute*), 555
paid_media_payload (*telegram.TransactionPartnerUser attribute*), 776
PAID_MEDIA_PAYMENT (*telegram.constants.TransactionPartnerUser attribute*), 1031
PAID_MESSAGE_PRICE_CHANGED (*telegram.constants.MessageType attribute*), 1005
PAID_MESSAGE_PRICE_CHANGED (*telegram.ext.filters.StatusUpdate attribute*), 910
paid_message_price_changed (*telegram.Message attribute*), 486
paid_message_star_count (*telegram.ChatFullInfo attribute*), 313
paid_message_star_count (*telegram.PaidMessagePriceChanged attribute*), 557
PAID_MESSAGES_PER_SECOND (*telegram.constants.FloodLimit attribute*), 977
paid_star_count (*telegram.Message attribute*), 482
PaidMedia (*class in telegram*), 551
PaidMediaInfo (*class in telegram*), 552
PaidMediaLivePhoto (*class in telegram*), 552
PaidMediaPhoto (*class in telegram*), 553
PaidMediaPreview (*class in telegram*), 554
PaidMediaPurchased (*class in telegram*), 555
PaidMediaPurchasedHandler (*class in telegram.ext*), 924
PaidMediaType (*class in telegram.constants*), 1009
PaidMediaVideo (*class in telegram*), 556
PaidMessagePriceChanged (*class in telegram*), 557
parameters (*telegram.request.RequestData property*), 1046
parametrized_url() (*telegram.request.RequestData method*), 1046
parent_chat (*telegram.ChatFullInfo attribute*), 313
parse_caption_entities() (*telegram.Message method*), 503
parse_caption_entity() (*telegram.Message method*), 503
parse_description_entities() (*telegram.Poll method*), 564
parse_description_entity() (*telegram.Poll method*), 565
parse_entities() (*telegram.Checklist method*), 286
parse_entities() (*telegram.ChecklistTask method*), 288
parse_entities() (*telegram.GiftInfo method*), 393
parse_entities() (*telegram.Message method*), 504
parse_entities() (*telegram.OwnedGiftRegular method*), 547
parse_entities() (*telegram.PollOption method*), 572
parse_entity() (*telegram.Checklist method*), 286
parse_entity() (*telegram.ChecklistTask method*), 288
parse_entity() (*telegram.GiftInfo method*), 394
parse_entity() (*telegram.Message method*), 504
parse_entity() (*telegram.OwnedGiftRegular method*), 548
parse_entity() (*telegram.PollOption method*), 572
parse_explanation_entities() (*telegram.Poll method*), 565
parse_explanation_entity() (*telegram.Poll method*), 566
parse_json_payload() (*telegram.request.BaseRequest static method*), 1044
parse_mode (*telegram.ext.Defaults property*), 848
parse_mode (*telegram.InlineQueryResultAudio attribute*), 698
parse_mode (*telegram.InlineQueryResultCachedAudio attribute*), 700
parse_mode (*telegram.InlineQueryResultCachedDocument attribute*), 702
parse_mode (*telegram.InlineQueryResultCachedGif attribute*), 703
parse_mode (*telegram.InlineQueryResultCachedMpeg4Gif attribute*), 706
parse_mode (*telegram.InlineQueryResultCachedPhoto attribute*), 708
parse_mode (*telegram.InlineQueryResultCachedVideo*

attribute), 711
parse_mode (*telegram.InlineQueryResultCachedVoice attribute*), 713
parse_mode (*telegram.InlineQueryResultDocument attribute*), 717
parse_mode (*telegram.InlineQueryResultGif attribute*), 721
parse_mode (*telegram.InlineQueryResultMpeg4Gif attribute*), 727
parse_mode (*telegram.InlineQueryResultPhoto attribute*), 730
parse_mode (*telegram.InlineQueryResultVideo attribute*), 736
parse_mode (*telegram.InlineQueryResultVoice attribute*), 739
parse_mode (*telegram.InputChecklist attribute*), 411
parse_mode (*telegram.InputChecklistTask attribute*), 412
parse_mode (*telegram.InputMedia attribute*), 416
parse_mode (*telegram.InputMediaAnimation attribute*), 418
parse_mode (*telegram.InputMediaAudio attribute*), 421
parse_mode (*telegram.InputMediaDocument attribute*), 423
parse_mode (*telegram.InputMediaLivePhoto attribute*), 425
parse_mode (*telegram.InputMediaPhoto attribute*), 428
parse_mode (*telegram.InputMediaVideo attribute*), 433
parse_mode (*telegram.InputTextMessageContent attribute*), 741
parse_option_text_entities() (*telegram.PollOptionAdded method*), 573
parse_option_text_entities() (*telegram.PollOptionDeleted method*), 575
parse_option_text_entity() (*telegram.PollOptionAdded method*), 574
parse_option_text_entity() (*telegram.PollOptionDeleted method*), 576
parse_question_entities() (*telegram.Poll method*), 566
parse_question_entity() (*telegram.Poll method*), 567
parse_text_entities() (*telegram.Game method*), 778
parse_text_entity() (*telegram.Game method*), 779
ParseMode (*class in telegram.constants*), 1010
PARTY_POPPER (*telegram.constants.ReactionEmoji attribute*), 1019
passport (*telegram.SecureData attribute*), 800
PASSPORT_DATA (*in module telegram.ext.filters*), 890
PASSPORT_DATA (*telegram.constants.MessageAttachmentType attribute*), 993
PASSPORT_DATA (*telegram.constants.MessageType attribute*), 1005
passport_data (*telegram.Message attribute*), 482
passport_registration (*telegram.SecureData attribute*), 801
PassportData (*class in telegram*), 786
PassportDecryptionError, 1038
PassportElementError (*class in telegram*), 787
PassportElementErrorDataField (*class in telegram*), 788
PassportElementErrorFile (*class in telegram*), 789
PassportElementErrorFiles (*class in telegram*), 789
PassportElementErrorFrontSide (*class in telegram*), 790
PassportElementErrorReverseSide (*class in telegram*), 791
PassportElementErrorSelfie (*class in telegram*), 792
PassportElementErrorTranslationFile (*class in telegram*), 793
PassportElementErrorTranslationFiles (*class in telegram*), 793
PassportElementErrorUnspecified (*class in telegram*), 794
PassportFile (*class in telegram*), 795
PATTERN (*telegram.BackgroundType attribute*), 291
PATTERN (*telegram.constants.BackgroundTypeType attribute*), 963
pattern (*telegram.ext.CallbackQueryHandler attribute*), 874
pattern (*telegram.ext.ChosenInlineResultHandler attribute*), 881
pattern (*telegram.ext.InlineQueryHandler attribute*), 918
pattern (*telegram.ext.PreCheckoutQueryHandler attribute*), 929
pattern (*telegram.ext.StringRegexHandler attribute*), 935
pay (*telegram.InlineKeyboardButton attribute*), 405
payload (*telegram.InputInvoiceMessageContent attribute*), 750
PAYMENT_REFUNDED (*telegram.constants.SuggestedPostRefunded attribute*), 1030
PDF (*telegram.ext.filters.Document attribute*), 901
PENDING (*telegram.constants.RevenueWithdrawalStateType attribute*), 1023
PENDING (*telegram.constants.SuggestedPostInfoState attribute*), 1029
PENDING (*telegram.RevenueWithdrawalState attribute*), 760
PENDING (*telegram.SuggestedPostInfo attribute*), 602
pending_join_request_count (*telegram.ChatInviteLink attribute*), 346
pending_update_count (*telegram.WebhookInfo attribute*), 676
per_chat (*telegram.ext.ConversationHandler property*), 888
per_message (*telegram.ext.ConversationHandler*

property), 888
per_user (*telegram.ext.ConversationHandler property*), 888
performer (*telegram.Audio attribute*), 223
performer (*telegram.InlineQueryResultAudio attribute*), 698
performer (*telegram.InputMediaAudio attribute*), 422
permissions (*telegram.ChatFullInfo attribute*), 311
persistence (*telegram.ext.Application attribute*), 805
persistence() (*telegram.ext.ApplicationBuilder method*), 830
persistence_data (*telegram.ext.CallbackDataCache property*), 954
PersistenceInput (*class in telegram.ext*), 947
persistent (*telegram.ext.ConversationHandler property*), 888
persistent_id (*telegram.PollOption attribute*), 571
PERSON_WITH_FOLDED_HANDS (*telegram.constants.ReactionEmoji attribute*), 1019
personal_chat (*telegram.ChatFullInfo attribute*), 309
personal_details (*telegram.SecureData attribute*), 800
personal_remaining_count (*telegram.Gift attribute*), 680
personal_total_count (*telegram.Gift attribute*), 680
PersonalChatMessagesLimit (*class in telegram.constants*), 1010
PersonalDetails (*class in telegram*), 797
PHONE_NUMBER (*telegram.constants.MessageEntityType attribute*), 997
phone_number (*telegram.Contact attribute*), 373
phone_number (*telegram.EncryptedPassportElement attribute*), 784
phone_number (*telegram.InlineQueryResultContact attribute*), 714
phone_number (*telegram.InputContactMessageContent attribute*), 747
PHONE_NUMBER (*telegram.MessageEntity attribute*), 534
phone_number (*telegram.OrderInfo attribute*), 757
PHOTO (*in module telegram.ext.filters*), 890
photo (*telegram.ChatFullInfo attribute*), 308
photo (*telegram.ChatShared attribute*), 372
PHOTO (*telegram.constants.BaseInputMediaType attribute*), 964
PHOTO (*telegram.constants.InlineQueryResultType attribute*), 981
PHOTO (*telegram.constants.InputMediaType attribute*), 983
PHOTO (*telegram.constants.InputPaidMediaType attribute*), 984
PHOTO (*telegram.constants.InputStoryContentType attribute*), 985
PHOTO (*telegram.constants.MessageAttachmentType attribute*), 993
PHOTO (*telegram.constants.MessageType attribute*), 1005
PHOTO (*telegram.constants.PaidMediaType attribute*), 1009
photo (*telegram.ExternalReplyInfo attribute*), 380
photo (*telegram.Game attribute*), 778
photo (*telegram.InputMediaLivePhoto attribute*), 425
PHOTO (*telegram.InputPaidMedia attribute*), 436
photo (*telegram.InputPaidMediaLivePhoto attribute*), 437
photo (*telegram.InputProfilePhotoStatic attribute*), 442
PHOTO (*telegram.InputStoryContent attribute*), 443
photo (*telegram.InputStoryContentPhoto attribute*), 444
photo (*telegram.LivePhoto attribute*), 456
photo (*telegram.Message attribute*), 479
PHOTO (*telegram.PaidMedia attribute*), 551
photo (*telegram.PaidMediaPhoto attribute*), 554
photo (*telegram.PollMedia attribute*), 569
photo (*telegram.SharedUser attribute*), 592
photo_file_id (*telegram.InlineQueryResultCachedPhoto attribute*), 707
PHOTO_HEIGHT (*telegram.constants.InputStoryContentLimit attribute*), 985
photo_height (*telegram.InlineQueryResultPhoto attribute*), 729
photo_height (*telegram.InputInvoiceMessageContent attribute*), 751
photo_size (*telegram.InputInvoiceMessageContent attribute*), 751
photo_url (*telegram.InlineQueryResultPhoto attribute*), 729
photo_url (*telegram.InputInvoiceMessageContent attribute*), 750
PHOTO_WIDTH (*telegram.constants.InputStoryContentLimit attribute*), 985
photo_width (*telegram.InlineQueryResultPhoto attribute*), 729
photo_width (*telegram.InputInvoiceMessageContent attribute*), 751
photos (*telegram.UserProfilePhotos attribute*), 661
PhotoSize (*class in telegram*), 557
PHOTOSIZE_UPLOAD (*telegram.constants.FileSizeLimit attribute*), 976
PHOTOSIZE_UPLOAD (*telegram.constants.InputStoryContentLimit attribute*), 984
PicklePersistence (*class in telegram.ext*), 948
PILE_OF_POO (*telegram.constants.ReactionEmoji attribute*), 1019
PILL (*telegram.constants.ReactionEmoji attribute*), 1019

pin() (*telegram.Message method*), 505
pin_chat_message() (*telegram.Bot method*), 104
pin_message() (*telegram.CallbackQuery method*), 247
pin_message() (*telegram.Chat method*), 263
pin_message() (*telegram.ChatFullInfo method*), 326
pin_message() (*telegram.User method*), 645
pinChatMessage() (*telegram.Bot method*), 104
PINK (*telegram.constants.ForumIconColor attribute*), 977
pinned_message (*telegram.ChatFullInfo attribute*), 311
PINNED_MESSAGE (*telegram.constants.MessageType attribute*), 1005
PINNED_MESSAGE (*telegram.ext.filters.StatusUpdate attribute*), 910
pinned_message (*telegram.Message attribute*), 482
point (*telegram.MaskPosition attribute*), 683
Poll (*class in telegram*), 559
POLL (*in module telegram.ext.filters*), 890
POLL (*telegram.constants.MessageAttachmentType attribute*), 993
POLL (*telegram.constants.MessageType attribute*), 1005
POLL (*telegram.constants.UpdateType attribute*), 1034
poll (*telegram.ExternalReplyInfo attribute*), 382
poll (*telegram.Message attribute*), 483
POLL (*telegram.Update attribute*), 629
poll (*telegram.Update attribute*), 625
POLL_ANSWER (*telegram.constants.UpdateType attribute*), 1034
POLL_ANSWER (*telegram.Update attribute*), 629
poll_answer (*telegram.Update attribute*), 625
poll_id (*telegram.PollAnswer attribute*), 568
poll_message (*telegram.PollOptionAdded attribute*), 573
poll_message (*telegram.PollOptionDeleted attribute*), 575
POLL_OPTION_ADDED (*telegram.constants.MessageType attribute*), 1005
POLL_OPTION_ADDED (*telegram.ext.filters.StatusUpdate attribute*), 910
poll_option_added (*telegram.Message attribute*), 489
POLL_OPTION_DELETED (*telegram.constants.MessageType attribute*), 1005
POLL_OPTION_DELETED (*telegram.ext.filters.StatusUpdate attribute*), 910
poll_option_deleted (*telegram.Message attribute*), 489
poll_option_id (*telegram.ReplyParameters attribute*), 590
PollAnswer (*class in telegram*), 567
PollAnswerHandler (*class in telegram.ext*), 925
PollHandler (*class in telegram.ext*), 926
PollingLimit (*class in telegram.constants*), 1012
PollLimit (*class in telegram.constants*), 1010
PollMedia (*class in telegram*), 568
PollOption (*class in telegram*), 570
PollOptionAdded (*class in telegram*), 573
PollOptionDeleted (*class in telegram*), 574
PollType (*class in telegram.constants*), 1012
pool_timeout() (*telegram.ext.ApplicationBuilder method*), 830
position (*telegram.GameHighScore attribute*), 780
position (*telegram.StoryArea attribute*), 594
position (*telegram.TextQuote attribute*), 612
post() (*telegram.request.BaseRequest method*), 1044
post_code (*telegram.ResidentialAddress attribute*), 799
post_code (*telegram.ShippingAddress attribute*), 763
POST_DELETED (*telegram.constants.SuggestedPostRefunded attribute*), 1030
post_init (*telegram.ext.Application attribute*), 806
post_init() (*telegram.ext.ApplicationBuilder method*), 831
post_shutdown (*telegram.ext.Application attribute*), 806
post_shutdown() (*telegram.ext.ApplicationBuilder method*), 832
post_stop (*telegram.ext.Application attribute*), 806
post_stop() (*telegram.ext.ApplicationBuilder method*), 832
post_story() (*telegram.Bot method*), 105
postStory() (*telegram.Bot method*), 105
POUTING_FACE (*telegram.constants.ReactionEmoji attribute*), 1019
PRE (*telegram.constants.MessageEntityType attribute*), 997
PRE (*telegram.MessageEntity attribute*), 534
PRE_CHECKOUT_QUERY (*telegram.constants.UpdateType attribute*), 1034
PRE_CHECKOUT_QUERY (*telegram.Update attribute*), 630
pre_checkout_query (*telegram.Update attribute*), 625
PreCheckoutQuery (*class in telegram*), 757
PreCheckoutQueryHandler (*class in telegram.ext*), 928
prefer_large_media (*telegram.LinkPreviewOptions attribute*), 454
prefer_small_media (*telegram.LinkPreviewOptions attribute*), 454
PrefixHandler (*class in telegram.ext*), 929
PREMIUM (*telegram.ChatBoostSource attribute*), 301
PREMIUM (*telegram.constants.ChatBoostSources attribute*), 969
PREMIUM (*telegram.ext.filters.Sticker attribute*), 911
premium_animation (*telegram.Sticker attribute*), 686
PREMIUM_PURCHASE (*telegram.constants.TransactionPartnerUser*

attribute), 1031
premium_subscription (*telegram.AcceptedGiftTypes attribute*), 219
premium_subscription_duration (*telegram.TransactionPartnerUser attribute*), 776
premium_subscription_month_count (*telegram.Giveaway attribute*), 396
premium_subscription_month_count (*telegram.GiveawayWinners attribute*), 399
PREMIUM_USER (*in module telegram.ext.filters*), 891
PremiumSubscription (*class in telegram.constants*), 1013
prepaid_upgrade_star_count (*telegram.GiftInfo attribute*), 393
prepaid_upgrade_star_count (*telegram.OwnedGiftRegular attribute*), 547
PreparedInlineMessage (*class in telegram*), 752
PreparedKeyboardButton (*class in telegram*), 576
PREVIEW (*telegram.constants.PaidMediaType attribute*), 1009
PREVIEW (*telegram.PaidMedia attribute*), 551
price (*telegram.SuggestedPostApprovalFailed attribute*), 600
price (*telegram.SuggestedPostApproved attribute*), 601
price (*telegram.SuggestedPostInfo attribute*), 602
price (*telegram.SuggestedPostParameters attribute*), 604
prices (*telegram.InputInvoiceMessageContent attribute*), 750
prices (*telegram.ShippingOption attribute*), 763
PRIMARY (*telegram.constants.KeyboardButtonStyle attribute*), 988
PRIVATE (*telegram.Chat attribute*), 251
PRIVATE (*telegram.ChatFullInfo attribute*), 314
PRIVATE (*telegram.constants.ChatType attribute*), 973
PRIVATE (*telegram.ext.filters.ChatType attribute*), 897
private_key (*telegram.Bot property*), 106
private_key() (*telegram.ext.ApplicationBuilder method*), 833
prize_description (*telegram.Giveaway attribute*), 396
prize_description (*telegram.GiveawayWinners attribute*), 399
prize_star_count (*telegram.ChatBoostSourceGiveaway attribute*), 302
prize_star_count (*telegram.Giveaway attribute*), 396
prize_star_count (*telegram.GiveawayCreated attribute*), 397
prize_star_count (*telegram.GiveawayWinners attribute*), 399
process_callback_query() (*telegram.ext.CallbackDataCache method*), 954
process_error() (*telegram.ext.Application method*), 812
process_keyboard() (*telegram.ext.CallbackDataCache method*), 955
process_message() (*telegram.ext.CallbackDataCache method*), 955
process_request() (*telegram.ext.AIORateLimiter method*), 959
process_request() (*telegram.ext.BaseRateLimiter method*), 956
process_update() (*telegram.ext.Application method*), 812
process_update() (*telegram.ext.BaseUpdateProcessor method*), 839
profile_accent_color_id (*telegram.ChatFullInfo attribute*), 310
profile_background_custom_emoji_id (*telegram.ChatFullInfo attribute*), 310
ProfileAccentColor (*class in telegram.constants*), 1014
promote_chat_member() (*telegram.Bot method*), 106
promote_member() (*telegram.Chat method*), 263
promote_member() (*telegram.ChatFullInfo method*), 326
promoteChatMember() (*telegram.Bot method*), 106
protect_content (*telegram.ext.Defaults property*), 848
provider_data (*telegram.InputInvoiceMessageContent attribute*), 750
provider_payment_charge_id (*telegram.RefundedPayment attribute*), 759
provider_payment_charge_id (*telegram.SuccessfulPayment attribute*), 769
provider_token (*telegram.InputInvoiceMessageContent attribute*), 750
proximity_alert_radius (*telegram.InlineQueryResultLocation attribute*), 724
proximity_alert_radius (*telegram.InputLocationMessageContent attribute*), 743
proximity_alert_radius (*telegram.Location attribute*), 458
PROXIMITY_ALERT_TRIGGERED (*telegram.constants.MessageType attribute*), 1005
PROXIMITY_ALERT_TRIGGERED (*telegram.ext.filters.StatusUpdate attribute*), 910
proximity_alert_triggered (*telegram.Message attribute*), 483
ProximityAlertTriggered (*class in telegram*), 576
proxy() (*telegram.ext.ApplicationBuilder method*), 834

PTBDeprecationWarning, 1049
PTBRuntimeWarning, 1050
PTBUserWarning, 1050
publisher_chat (*telegram.Gift attribute*), 680
publisher_chat (*telegram.UniqueGift attribute*), 614
PURCHASED_PAID_MEDIA (*telegram.constants.UpdateType attribute*), 1034
PURCHASED_PAID_MEDIA (*telegram.Update attribute*), 630
purchased_paid_media (*telegram.Update attribute*), 627
PURPLE (*telegram.constants.ForumIconColor attribute*), 977
PY (*telegram.ext.filters.Document attribute*), 901

Q

qualities (*telegram.Video attribute*), 667
query (*telegram.ChosenInlineResult attribute*), 690
query (*telegram.InlineQuery attribute*), 692
query (*telegram.SwitchInlineQueryChosenChat attribute*), 607
question (*telegram.Poll attribute*), 561
question_entities (*telegram.Poll attribute*), 562
question_parse_mode (*telegram.ext.Defaults property*), 848
QUIZ (*telegram.constants.PollType attribute*), 1012
QUIZ (*telegram.Poll attribute*), 564
quote (*telegram.Message attribute*), 487
quote (*telegram.ReplyParameters attribute*), 589
quote_entities (*telegram.ReplyParameters attribute*), 589
quote_parse_mode (*telegram.ext.Defaults property*), 848
quote_parse_mode (*telegram.ReplyParameters attribute*), 589
quote_position (*telegram.ReplyParameters attribute*), 589

R

RARE (*telegram.constants.UniqueGiftModelRarity attribute*), 1032
rarity (*telegram.UniqueGiftModel attribute*), 621
rarity_per_mille (*telegram.UniqueGiftBackdrop attribute*), 617
rarity_per_mille (*telegram.UniqueGiftModel attribute*), 620
rarity_per_mille (*telegram.UniqueGiftSymbol attribute*), 621
rate_limiter (*telegram.ext.ExtBot property*), 851
rate_limiter() (*telegram.ext.ApplicationBuilder method*), 834
rating (*telegram.ChatFullInfo attribute*), 313
rating (*telegram.UserRating attribute*), 662
reaction_type (*telegram.StoryAreaTypeSuggestedReaction attribute*), 598
ReactionCount (*class in telegram*), 577
ReactionEmoji (*class in telegram.constants*), 1015
reactions (*telegram.MessageReactionCountUpdated attribute*), 542
ReactionType (*class in telegram*), 578
ReactionType (*class in telegram.constants*), 1022
ReactionTypeCustomEmoji (*class in telegram*), 579
ReactionTypeEmoji (*class in telegram*), 579
ReactionTypePaid (*class in telegram*), 580
read_business_message() (*telegram.Bot method*), 109
read_business_message() (*telegram.Chat method*), 264
read_business_message() (*telegram.ChatFullInfo method*), 327
read_business_message() (*telegram.Message method*), 505
read_timeout (*telegram.request.BaseRequest property*), 1045
read_timeout (*telegram.request.HTTPXRequest property*), 1049
read_timeout() (*telegram.ext.ApplicationBuilder method*), 834
readBusinessMessage() (*telegram.Bot method*), 108
reason (*telegram.SuggestedPostRefunded attribute*), 606
receiver (*telegram.StarTransaction attribute*), 767
RECORD_VIDEO (*telegram.constants.ChatAction attribute*), 968
RECORD_VIDEO_NOTE (*telegram.constants.ChatAction attribute*), 968
RECORD_VOICE (*telegram.constants.ChatAction attribute*), 969
RED (*telegram.constants.ForumIconColor attribute*), 977
RED (*telegram.constants.KeyboardButtonStyle attribute*), 988
RED_HEART (*telegram.constants.ReactionEmoji attribute*), 1019
refresh_bot_data() (*telegram.ext.BasePersistence method*), 941
refresh_bot_data() (*telegram.ext.DictPersistence method*), 946
refresh_bot_data() (*telegram.ext.PicklePersistence method*), 951
refresh_chat_data() (*telegram.ext.BasePersistence method*), 941
refresh_chat_data() (*telegram.ext.DictPersistence method*), 946
refresh_chat_data() (*telegram.ext.PicklePersistence method*), 951
refresh_data() (*telegram.ext.CallbackContext method*), 843
refresh_user_data() (*telegram.ext.BasePersistence method*), 941
refresh_user_data() (*telegram.ext.DictPersistence method*), 946
refresh_user_data() (*telegram.ext.PicklePersistence method*), 951

refund_star_payment() (*telegram.Bot method*), 109
refund_star_payment() (*telegram.User method*), 645
REFUNDED_PAYMENT (*telegram.constants.MessageType attribute*), 1005
REFUNDED_PAYMENT (*telegram.ext.filters.StatusUpdate attribute*), 910
refunded_payment (*telegram.Message attribute*), 488
RefundedPayment (*class in telegram*), 759
refundStarPayment() (*telegram.Bot method*), 109
Regex (*class in telegram.ext.filters*), 905
REGULAR (*telegram.constants.OwnedGiftType attribute*), 1009
REGULAR (*telegram.constants.PollType attribute*), 1012
REGULAR (*telegram.constants.StickerType attribute*), 1026
REGULAR (*telegram.OwnedGift attribute*), 544
REGULAR (*telegram.Poll attribute*), 564
REGULAR (*telegram.Sticker attribute*), 687
RELATIVE (*telegram.constants.MessageEntityDateTimeFormat attribute*), 995
remaining_count (*telegram.Gift attribute*), 680
remove_bot_ids() (*telegram.ext.filters.ViaBot method*), 917
remove_business_account_profile_photo() (*telegram.Bot method*), 110
remove_chat_ids() (*telegram.ext.filters.Chat method*), 896
remove_chat_ids() (*telegram.ext.filters.ForwardedFrom method*), 903
remove_chat_ids() (*telegram.ext.filters.SenderChat method*), 907
remove_chat_verification() (*telegram.Bot method*), 111
remove_date (*telegram.ChatBoostRemoved attribute*), 300
remove_error_handler() (*telegram.ext.Application method*), 813
remove_handler() (*telegram.ext.Application method*), 813
remove_keyboard (*telegram.ReplyKeyboardRemove attribute*), 587
remove_my_profile_photo() (*telegram.Bot method*), 112
remove_user_ids() (*telegram.ext.filters.User method*), 915
remove_user_verification() (*telegram.Bot method*), 112
remove_usernames() (*telegram.ext.filters.Chat method*), 896
remove_usernames() (*telegram.ext.filters.ForwardedFrom method*), 903
remove_usernames() (*telegram.ext.filters.SenderChat method*), 907
remove_usernames() (*telegram.ext.filters.User method*), 915
remove_usernames() (*telegram.ext.filters.ViaBot method*), 916
remove_verification() (*telegram.Chat method*), 264
remove_verification() (*telegram.ChatFullInfo method*), 327
remove_verification() (*telegram.User method*), 645
removeBusinessAccountProfilePhoto() (*telegram.Bot method*), 110
removeChatVerification() (*telegram.Bot method*), 110
removed (*telegram.ext.Job property*), 854
REMOVED_CHAT_BOOST (*telegram.constants.UpdateType attribute*), 1035
REMOVED_CHAT_BOOST (*telegram.ext.ChatBoostHandler attribute*), 876
REMOVED_CHAT_BOOST (*telegram.Update attribute*), 630
removed_chat_boost (*telegram.Update attribute*), 626
removeMyProfilePhoto() (*telegram.Bot method*), 110
removeUserVerification() (*telegram.Bot method*), 110
rental_agreement (*telegram.SecureData attribute*), 801
reopen_forum_topic() (*telegram.Bot method*), 113
reopen_forum_topic() (*telegram.Chat method*), 265
reopen_forum_topic() (*telegram.ChatFullInfo method*), 328
reopen_forum_topic() (*telegram.Message method*), 505
reopen_general_forum_topic() (*telegram.Bot method*), 114
reopen_general_forum_topic() (*telegram.Chat method*), 265
reopen_general_forum_topic() (*telegram.ChatFullInfo method*), 328
reopenForumTopic() (*telegram.Bot method*), 113
reopenGeneralForumTopic() (*telegram.Bot method*), 113
replace_managed_bot_token() (*telegram.Bot method*), 115
replace_sticker_in_set() (*telegram.Bot method*), 115
replace_token() (*telegram.User method*), 645
replaceManagedBotToken() (*telegram.Bot method*), 114
replaceStickerInSet() (*telegram.Bot method*), 114
REPLY (*in module telegram.ext.filters*), 891
reply_animation() (*telegram.Message method*), 506
reply_audio() (*telegram.Message method*), 506
reply_chat_action() (*telegram.Message method*), 507
reply_checklist() (*telegram.Message method*), 507

reply_contact() (*telegram.Message method*), 508
reply_copy() (*telegram.Message method*), 509
reply_dice() (*telegram.Message method*), 509
reply_document() (*telegram.Message method*), 510
reply_game() (*telegram.Message method*), 511
reply_html() (*telegram.Message method*), 511
reply_invoice() (*telegram.Message method*), 512
reply_live_photo() (*telegram.Message method*), 513
reply_location() (*telegram.Message method*), 513
reply_markdown() (*telegram.Message method*), 514
reply_markdown_v2() (*telegram.Message method*), 515
reply_markup (*telegram.InlineQueryResultArticle attribute*), 696
reply_markup (*telegram.InlineQueryResultAudio attribute*), 698
reply_markup (*telegram.InlineQueryResultCachedAudio attribute*), 700
reply_markup (*telegram.InlineQueryResultCachedDocument attribute*), 702
reply_markup (*telegram.InlineQueryResultCachedGif attribute*), 704
reply_markup (*telegram.InlineQueryResultCachedMpeg4Gif attribute*), 706
reply_markup (*telegram.InlineQueryResultCachedPhoto attribute*), 708
reply_markup (*telegram.InlineQueryResultCachedSticker attribute*), 709
reply_markup (*telegram.InlineQueryResultCachedVideo attribute*), 711
reply_markup (*telegram.InlineQueryResultCachedVoice attribute*), 713
reply_markup (*telegram.InlineQueryResultContact attribute*), 715
reply_markup (*telegram.InlineQueryResultDocument attribute*), 717
reply_markup (*telegram.InlineQueryResultGame attribute*), 719
reply_markup (*telegram.InlineQueryResultGif attribute*), 721
reply_markup (*telegram.InlineQueryResultLocation attribute*), 724
reply_markup (*telegram.InlineQueryResultMpeg4Gif attribute*), 727
reply_markup (*telegram.InlineQueryResultPhoto attribute*), 730
reply_markup (*telegram.InlineQueryResultVenue attribute*), 734
reply_markup (*telegram.InlineQueryResultVideo attribute*), 737
reply_markup (*telegram.InlineQueryResultVoice attribute*), 739
reply_markup (*telegram.Message attribute*), 484
reply_media_group() (*telegram.Message method*), 516
reply_paid_media() (*telegram.Message method*), 516
reply_photo() (*telegram.Message method*), 517
reply_poll() (*telegram.Message method*), 518
reply_sticker() (*telegram.Message method*), 518
reply_text() (*telegram.Message method*), 519
reply_text_draft() (*telegram.Message method*), 520
reply_to_checklist_task_id (*telegram.Message attribute*), 489
reply_to_message (*telegram.Message attribute*), 477
reply_to_poll_option_id (*telegram.Message attribute*), 489
REPLY_TO_STORY (*in module telegram.ext.filters*), 891
REPLY_TO_STORY (*telegram.constants.MessageType attribute*), 1005
reply_to_story (*telegram.Message attribute*), 487
reply_venue() (*telegram.Message method*), 520
reply_video() (*telegram.Message method*), 521
reply_video_note() (*telegram.Message method*), 522
reply_voice() (*telegram.Message method*), 522
ReplyKeyboardMarkup (*class in telegram*), 581
ReplyKeyboardRemove (*class in telegram*), 586
ReplyLimit (*class in telegram.constants*), 1022
ReplyParameters (*class in telegram*), 587
repost() (*telegram.Story method*), 593
repost_story() (*telegram.Bot method*), 116
repost_story() (*telegram.Chat method*), 265
repost_story() (*telegram.ChatFullInfo method*), 328
repost_story() (*telegram.User method*), 646
repostStory() (*telegram.Bot method*), 116
request (*telegram.Bot property*), 117
request() (*telegram.ext.ApplicationBuilder method*), 835
request_chat (*telegram.KeyboardButton attribute*), 448
request_contact (*telegram.KeyboardButton attribute*), 447
request_count (*telegram.TransactionPartnerTelegramApi attribute*), 774
request_id (*telegram.ChatShared attribute*), 371
request_id (*telegram.KeyboardButtonRequestChat attribute*), 450
request_id (*telegram.KeyboardButtonRequestManagedBot attribute*), 452
request_id (*telegram.KeyboardButtonRequestUsers attribute*), 453
request_id (*telegram.UsersShared attribute*), 663
request_location (*telegram.KeyboardButton attribute*), 447

request_managed_bot (*telegram.KeyboardButton attribute*), 448
request_name (*telegram.KeyboardButtonRequestUsers attribute*), 453
request_photo (*telegram.KeyboardButtonRequestChat attribute*), 451
request_photo (*telegram.KeyboardButtonRequestUsers attribute*), 453
request_poll (*telegram.KeyboardButton attribute*), 447
request_title (*telegram.KeyboardButtonRequestChat attribute*), 451
request_username (*telegram.KeyboardButtonRequestChat attribute*), 451
request_username (*telegram.KeyboardButtonRequestUsers attribute*), 453
request_users (*telegram.KeyboardButton attribute*), 448
request_write_access (*telegram.LoginUrl attribute*), 460
RequestData (*class in telegram.request*), 1046
RESALE (*telegram.constants.UniqueGiftInfoOrigin attribute*), 1032
RESALE (*telegram.UniqueGiftInfo attribute*), 620
residence_country_code (*telegram.PersonalDetails attribute*), 798
ResidentialAddress (*class in telegram*), 798
resize_keyboard (*telegram.ReplyKeyboardMarkup attribute*), 583
restrict_chat_member() (*telegram.Bot method*), 117
restrict_member() (*telegram.Chat method*), 265
restrict_member() (*telegram.ChatFullInfo method*), 328
restrictChatMember() (*telegram.Bot method*), 117
RESTRICTED (*telegram.ChatMember attribute*), 351
RESTRICTED (*telegram.constants.ChatMemberStatus attribute*), 972
result_id (*telegram.ChosenInlineResult attribute*), 689
RESULTS (*telegram.constants.InlineQueryLimit attribute*), 980
retrieve() (*telegram.request.BaseRequest method*), 1045
retry_after (*telegram.error.RetryAfter attribute*), 1039
RetryAfter, 1038
RevenueWithdrawalState (*class in telegram*), 760
RevenueWithdrawalStateFailed (*class in telegram*), 760
RevenueWithdrawalStatePending (*class in telegram*), 761
RevenueWithdrawalStateSucceeded (*class in telegram*), 761
RevenueWithdrawalStateType (*class in telegram.constants*), 1023
reverse_side (*telegram.EncryptedPassportElement attribute*), 784
reverse_side (*telegram.SecureValue attribute*), 802
REVERSED_HAND_WITH_MIDDLE_FINGER_EXTENDED (*telegram.constants.ReactionEmoji attribute*), 1019
revoke_chat_invite_link() (*telegram.Bot method*), 118
revoke_invite_link() (*telegram.Chat method*), 266
revoke_invite_link() (*telegram.ChatFullInfo method*), 329
revokeChatInviteLink() (*telegram.Bot method*), 118
rights (*telegram.BusinessConnection attribute*), 236
ROLLING_ON_THE_FLOOR_LAUGHING (*telegram.constants.ReactionEmoji attribute*), 1020
rotation_angle (*telegram.BackgroundFillGradient attribute*), 297
rotation_angle (*telegram.StoryAreaPosition attribute*), 595
run() (*telegram.ext.Job method*), 854
run_custom() (*telegram.ext.JobQueue method*), 857
run_daily() (*telegram.ext.JobQueue method*), 857
run_monthly() (*telegram.ext.JobQueue method*), 858
run_once() (*telegram.ext.JobQueue method*), 859
run_polling() (*telegram.ext.Application method*), 813
run_repeating() (*telegram.ext.JobQueue method*), 860
run_webhook() (*telegram.ext.Application method*), 815
running (*telegram.ext.Application property*), 817

S

SALUTING_FACE (*telegram.constants.ReactionEmoji attribute*), 1020
save_prepared_inline_message() (*telegram.Bot method*), 119
save_prepared_keyboard_button() (*telegram.Bot method*), 120
savePreparedInlineMessage() (*telegram.Bot method*), 119
savePreparedKeyboardButton() (*telegram.Bot method*), 119
scale (*telegram.MaskPosition attribute*), 683
schedule_removal() (*telegram.ext.Job method*), 854
scheduler (*telegram.ext.JobQueue attribute*), 855
scheduler_configuration (*telegram.ext.JobQueue property*), 861
score (*telegram.GameHighScore attribute*), 780
secret (*telegram.DataCredentials attribute*), 781
secret (*telegram.EncryptedCredentials attribute*), 782
secret (*telegram.FileCredentials attribute*), 785

secure_data (*telegram.Credentials attribute*), 780
SecureData (*class in telegram*), 799
SecureValue (*class in telegram*), 801
SEE_NO_EVIL_MONKEY (*telegram.constants.ReactionEmoji attribute*), 1020
selective (*telegram.ForceReply attribute*), 387
selective (*telegram.ReplyKeyboardMarkup attribute*), 583
selective (*telegram.ReplyKeyboardRemove attribute*), 587
selfie (*telegram.EncryptedPassportElement attribute*), 784
selfie (*telegram.SecureValue attribute*), 802
send_action() (*telegram.Chat method*), 266
send_action() (*telegram.ChatFullInfo method*), 329
send_action() (*telegram.User method*), 646
send_animation() (*telegram.Bot method*), 125
send_animation() (*telegram.Chat method*), 266
send_animation() (*telegram.ChatFullInfo method*), 329
send_animation() (*telegram.User method*), 646
send_audio() (*telegram.Bot method*), 128
send_audio() (*telegram.Chat method*), 267
send_audio() (*telegram.ChatFullInfo method*), 330
send_audio() (*telegram.User method*), 647
send_chat_action() (*telegram.Bot method*), 131
send_chat_action() (*telegram.Chat method*), 267
send_chat_action() (*telegram.ChatFullInfo method*), 330
send_chat_action() (*telegram.User method*), 647
send_checklist() (*telegram.Bot method*), 132
send_checklist() (*telegram.Chat method*), 267
send_checklist() (*telegram.ChatFullInfo method*), 330
send_contact() (*telegram.Bot method*), 133
send_contact() (*telegram.Chat method*), 267
send_contact() (*telegram.ChatFullInfo method*), 330
send_contact() (*telegram.User method*), 648
send_copies() (*telegram.Chat method*), 268
send_copies() (*telegram.ChatFullInfo method*), 331
send_copies() (*telegram.User method*), 648
send_copy() (*telegram.Chat method*), 268
send_copy() (*telegram.ChatFullInfo method*), 331
send_copy() (*telegram.User method*), 648
send_date (*telegram.OwnedGiftRegular attribute*), 546
send_date (*telegram.OwnedGiftUnique attribute*), 550
send_date (*telegram.SuggestedPostApproved attribute*), 601
send_date (*telegram.SuggestedPostInfo attribute*), 602
send_date (*telegram.SuggestedPostParameters attribute*), 604
send_dice() (*telegram.Bot method*), 135
send_dice() (*telegram.Chat method*), 269
send_dice() (*telegram.ChatFullInfo method*), 332
send_dice() (*telegram.User method*), 649
send_document() (*telegram.Bot method*), 137
send_document() (*telegram.Chat method*), 269
send_document() (*telegram.ChatFullInfo method*), 332
send_document() (*telegram.User method*), 649
send_email_to_provider (*telegram.InputInvoiceMessageContent attribute*), 751
send_game() (*telegram.Bot method*), 140
send_game() (*telegram.Chat method*), 269
send_game() (*telegram.ChatFullInfo method*), 332
send_game() (*telegram.User method*), 650
send_gift() (*telegram.Bot method*), 142
send_gift() (*telegram.Chat method*), 270
send_gift() (*telegram.ChatFullInfo method*), 333
send_gift() (*telegram.User method*), 650
send_invoice() (*telegram.Bot method*), 143
send_invoice() (*telegram.Chat method*), 270
send_invoice() (*telegram.ChatFullInfo method*), 333
send_invoice() (*telegram.User method*), 651
send_live_photo() (*telegram.Bot method*), 146
send_live_photo() (*telegram.Chat method*), 271
send_live_photo() (*telegram.ChatFullInfo method*), 334
send_live_photo() (*telegram.User method*), 651
send_location() (*telegram.Bot method*), 148
send_location() (*telegram.Chat method*), 271
send_location() (*telegram.ChatFullInfo method*), 334
send_location() (*telegram.User method*), 652
send_media_group() (*telegram.Bot method*), 151
send_media_group() (*telegram.Chat method*), 271
send_media_group() (*telegram.ChatFullInfo method*), 334
send_media_group() (*telegram.User method*), 652
send_message() (*telegram.Bot method*), 153
send_message() (*telegram.Chat method*), 272
send_message() (*telegram.ChatFullInfo method*), 335
send_message() (*telegram.User method*), 653
send_message_draft() (*telegram.Bot method*), 156
send_message_draft() (*telegram.Chat method*), 272
send_message_draft() (*telegram.ChatFullInfo method*), 335
send_message_draft() (*telegram.User method*), 653
send_paid_media() (*telegram.Bot method*), 157
send_paid_media() (*telegram.Chat method*), 272
send_paid_media() (*telegram.ChatFullInfo method*), 335
send_phone_number_to_provider (*telegram.InputInvoiceMessageContent attribute*), 751
send_photo() (*telegram.Bot method*), 159
send_photo() (*telegram.Chat method*), 273
send_photo() (*telegram.ChatFullInfo method*), 336
send_photo() (*telegram.User method*), 654
send_poll() (*telegram.Bot method*), 161
send_poll() (*telegram.Chat method*), 273
send_poll() (*telegram.ChatFullInfo method*), 336
send_poll() (*telegram.User method*), 654

send_sticker() (*telegram.Bot method*), 165
send_sticker() (*telegram.Chat method*), 273
send_sticker() (*telegram.ChatFullInfo method*), 336
send_sticker() (*telegram.User method*), 655
send_venue() (*telegram.Bot method*), 168
send_venue() (*telegram.Chat method*), 274
send_venue() (*telegram.ChatFullInfo method*), 337
send_venue() (*telegram.User method*), 655
send_video() (*telegram.Bot method*), 170
send_video() (*telegram.Chat method*), 274
send_video() (*telegram.ChatFullInfo method*), 337
send_video() (*telegram.User method*), 656
send_video_note() (*telegram.Bot method*), 173
send_video_note() (*telegram.Chat method*), 274
send_video_note() (*telegram.ChatFullInfo method*), 337
send_video_note() (*telegram.User method*), 656
send_voice() (*telegram.Bot method*), 176
send_voice() (*telegram.Chat method*), 275
send_voice() (*telegram.ChatFullInfo method*), 338
send_voice() (*telegram.User method*), 657
sendAnimation() (*telegram.Bot method*), 120
sendAudio() (*telegram.Bot method*), 121
sendChatAction() (*telegram.Bot method*), 121
sendChecklist() (*telegram.Bot method*), 121
sendContact() (*telegram.Bot method*), 121
sendDice() (*telegram.Bot method*), 121
sendDocument() (*telegram.Bot method*), 121
SENDER (*telegram.Chat attribute*), 251
SENDER (*telegram.ChatFullInfo attribute*), 314
SENDER (*telegram.constants.ChatType attribute*), 973
SENDER_BOOST_COUNT (*in module telegram.ext.filters*), 891
SENDER_BOOST_COUNT (*telegram.constants.MessageType attribute*), 1006
sender_boost_count (*telegram.Message attribute*), 487
SENDER_BUSINESS_BOT (*telegram.constants.MessageType attribute*), 1006
sender_business_bot (*telegram.Message attribute*), 488
sender_chat (*telegram.Message attribute*), 476
sender_chat (*telegram.MessageOriginChat attribute*), 540
sender_tag (*telegram.Message attribute*), 489
sender_user (*telegram.MessageOriginUser attribute*), 542
sender_user (*telegram.OwnedGiftRegular attribute*), 546
sender_user (*telegram.OwnedGiftUnique attribute*), 550
sender_user_name (*telegram.MessageOriginHiddenUser attribute*), 541
SenderChat (*class in telegram.ext.filters*), 905
sendGame() (*telegram.Bot method*), 122
sendGift() (*telegram.Bot method*), 122
sendInvoice() (*telegram.Bot method*), 122
sendLivePhoto() (*telegram.Bot method*), 122
sendLocation() (*telegram.Bot method*), 122
sendMediaGroup() (*telegram.Bot method*), 123
sendMessage() (*telegram.Bot method*), 123
sendMessageDraft() (*telegram.Bot method*), 123
sendPaidMedia() (*telegram.Bot method*), 123
sendPhoto() (*telegram.Bot method*), 123
sendPoll() (*telegram.Bot method*), 124
sendSticker() (*telegram.Bot method*), 124
sendVenue() (*telegram.Bot method*), 124
sendVideo() (*telegram.Bot method*), 124
sendVideoNote() (*telegram.Bot method*), 124
sendVoice() (*telegram.Bot method*), 125
SentGuestMessage (*class in telegram*), 590
SentWebAppMessage (*class in telegram*), 590
SERIOUS_FACE_WITH_SYMBOLS_COVERING_MOUTH (*telegram.constants.ReactionEmoji attribute*), 1020
SERVICE_CHAT (*telegram.constants.ChatID attribute*), 970
set_administrator_custom_title() (*telegram.Chat method*), 275
set_administrator_custom_title() (*telegram.ChatFullInfo method*), 338
set_application() (*telegram.ext.JobQueue method*), 862
set_bot() (*telegram.ext.BasePersistence method*), 942
set_bot() (*telegram.TelegramObject method*), 611
set_business_account_bio() (*telegram.Bot method*), 181
set_business_account_gift_settings() (*telegram.Bot method*), 182
set_business_account_name() (*telegram.Bot method*), 183
set_business_account_profile_photo() (*telegram.Bot method*), 183
set_business_account_username() (*telegram.Bot method*), 184
set_chat_administrator_custom_title() (*telegram.Bot method*), 185
set_chat_description() (*telegram.Bot method*), 185
set_chat_member_tag() (*telegram.Bot method*), 186
set_chat_member_tag() (*telegram.Chat method*), 275
set_chat_member_tag() (*telegram.ChatFullInfo method*), 338
set_chat_member_tag() (*telegram.User method*), 657
set_chat_menu_button() (*telegram.Bot method*), 187
set_chat_permissions() (*telegram.Bot method*), 188
set_chat_photo() (*telegram.Bot method*), 189
set_chat_sticker_set() (*telegram.Bot method*),

189
set_chat_title() (*telegram.Bot method*), 190
set_credentials() (*telegram.File method*), 386
set_custom_emoji_sticker_set_thumbnail() (*telegram.Bot method*), 191
set_description() (*telegram.Chat method*), 276
set_description() (*telegram.ChatFullInfo method*), 339
set_game_score() (*telegram.Bot method*), 191
set_game_score() (*telegram.CallbackQuery method*), 247
set_game_score() (*telegram.Message method*), 523
set_managed_bot_access_settings() (*telegram.Bot method*), 192
set_managed_bot_access_settings() (*telegram.User method*), 657
set_menu_button() (*telegram.Chat method*), 276
set_menu_button() (*telegram.ChatFullInfo method*), 339
set_menu_button() (*telegram.User method*), 658
set_message_reaction() (*telegram.Bot method*), 193
set_message_reaction() (*telegram.Chat method*), 276
set_message_reaction() (*telegram.ChatFullInfo method*), 339
set_my_commands() (*telegram.Bot method*), 194
set_my_default_administrator_rights() (*telegram.Bot method*), 195
set_my_description() (*telegram.Bot method*), 196
set_my_name() (*telegram.Bot method*), 197
set_my_profile_photo() (*telegram.Bot method*), 197
set_my_short_description() (*telegram.Bot method*), 198
set_name (*telegram.Sticker attribute*), 686
set_passport_data_errors() (*telegram.Bot method*), 198
set_permissions() (*telegram.Chat method*), 277
set_permissions() (*telegram.ChatFullInfo method*), 340
set_photo() (*telegram.Chat method*), 277
set_photo() (*telegram.ChatFullInfo method*), 340
set_reaction() (*telegram.Message method*), 523
set_sticker_emoji_list() (*telegram.Bot method*), 199
set_sticker_keywords() (*telegram.Bot method*), 200
set_sticker_mask_position() (*telegram.Bot method*), 200
set_sticker_position_in_set() (*telegram.Bot method*), 201
set_sticker_set_thumbnail() (*telegram.Bot method*), 201
set_sticker_set_title() (*telegram.Bot method*), 202
set_title() (*telegram.Chat method*), 277
set_title() (*telegram.ChatFullInfo method*), 340
set_user_emoji_status() (*telegram.Bot method*), 203
set_webhook() (*telegram.Bot method*), 204
setBusinessAccountBio() (*telegram.Bot method*), 179
setBusinessAccountGiftSettings() (*telegram.Bot method*), 179
setBusinessAccountName() (*telegram.Bot method*), 179
setBusinessAccountProfilePhoto() (*telegram.Bot method*), 179
setBusinessAccountUsername() (*telegram.Bot method*), 179
setChatAdministratorCustomTitle() (*telegram.Bot method*), 179
setChatDescription() (*telegram.Bot method*), 179
setChatMemberTag() (*telegram.Bot method*), 179
setChatMenuButton() (*telegram.Bot method*), 180
setChatPermissions() (*telegram.Bot method*), 180
setChatPhoto() (*telegram.Bot method*), 180
setChatStickerSet() (*telegram.Bot method*), 180
setChatTitle() (*telegram.Bot method*), 180
setCustomEmojiStickerSetThumbnail() (*telegram.Bot method*), 180
setGameScore() (*telegram.Bot method*), 180
setManagedBotAccessSettings() (*telegram.Bot method*), 180
setMessageReaction() (*telegram.Bot method*), 180
setMyCommands() (*telegram.Bot method*), 180
setMyDefaultAdministratorRights() (*telegram.Bot method*), 180
setMyDescription() (*telegram.Bot method*), 180
setMyName() (*telegram.Bot method*), 181
setMyProfilePhoto() (*telegram.Bot method*), 181
setMyShortDescription() (*telegram.Bot method*), 181
setPassportDataErrors() (*telegram.Bot method*), 181
setStickerEmojiList() (*telegram.Bot method*), 181
setStickerKeywords() (*telegram.Bot method*), 181
setStickerMaskPosition() (*telegram.Bot method*), 181
setStickerPositionInSet() (*telegram.Bot method*), 181
setStickerSetThumbnail() (*telegram.Bot method*), 181
setStickerSetTitle() (*telegram.Bot method*), 181
setUserEmojiStatus() (*telegram.Bot method*), 181
setWebhook() (*telegram.Bot method*), 181
SharedUser (*class in telegram*), 591
shift_entities() (*telegram.MessageEntity static method*), 536
shipping_address (*telegram.OrderInfo attribute*), 757
shipping_address (*telegram.ShippingQuery attribute*), 764
shipping_option_id (*telegram.PreCheckoutQuery attribute*), 758

shipping_option_id (*telegram.SuccessfulPayment attribute*), 769
SHIPPING_QUERY (*telegram.constants.UpdateType attribute*), 1035
SHIPPING_QUERY (*telegram.Update attribute*), 630
shipping_query (*telegram.Update attribute*), 625
ShippingAddress (*class in telegram*), 762
ShippingOption (*class in telegram*), 763
ShippingQuery (*class in telegram*), 764
ShippingQueryHandler (*class in telegram.ext*), 931
SHOCKED_FACE_WITH_EXPLODING_HEAD (*telegram.constants.ReactionEmoji attribute*), 1020
SHORT_DATE (*telegram.constants.MessageEntityDateTimeFormat attribute*), 996
SHORT_DATE_LONG_TIME (*telegram.constants.MessageEntityDateTimeFormat attribute*), 996
SHORT_DATE_SHORT_TIME (*telegram.constants.MessageEntityDateTimeFormat attribute*), 996
short_description (*telegram.BotShortDescription attribute*), 232
SHORT_TIME (*telegram.constants.MessageEntityDateTimeFormat attribute*), 996
show_above_text (*telegram.LinkPreviewOptions attribute*), 455
show_caption_above_media (*telegram.InlineQueryResultCachedGif attribute*), 704
show_caption_above_media (*telegram.InlineQueryResultCachedMpeg4Gif attribute*), 706
show_caption_above_media (*telegram.InlineQueryResultCachedPhoto attribute*), 708
show_caption_above_media (*telegram.InlineQueryResultCachedVideo attribute*), 711
show_caption_above_media (*telegram.InlineQueryResultGif attribute*), 722
show_caption_above_media (*telegram.InlineQueryResultMpeg4Gif attribute*), 728
show_caption_above_media (*telegram.InlineQueryResultPhoto attribute*), 730
show_caption_above_media (*telegram.InlineQueryResultVideo attribute*), 737
show_caption_above_media (*telegram.InputMediaAnimation attribute*), 419
show_caption_above_media (*telegram.InputMediaLivePhoto attribute*), 426
show_caption_above_media (*telegram.InputMediaPhoto attribute*), 429
show_caption_above_media (*telegram.InputMediaVideo attribute*), 434
show_caption_above_media (*telegram.Message attribute*), 478
SHRUG (*telegram.constants.ReactionEmoji attribute*), 1020
shutdown() (*telegram.Bot method*), 205
shutdown() (*telegram.ext.AIORateLimiter method*), 960
shutdown() (*telegram.ext.Application method*), 818
shutdown() (*telegram.ext.BaseRateLimiter method*), 958
shutdown() (*telegram.ext.BaseUpdateProcessor method*), 839
shutdown() (*telegram.ext.ExtBot method*), 851
shutdown() (*telegram.ext.SimpleUpdateProcessor method*), 862
shutdown() (*telegram.ext.Updater method*), 864
shutdown() (*telegram.request.BaseRequest method*), 1045
shutdown() (*telegram.request.HTTPXRequest method*), 1049
SimpleUpdateProcessor (*class in telegram.ext*), 862
single_file (*telegram.ext.PicklePersistence attribute*), 950
SIZE_BIG (*telegram.ChatPhoto attribute*), 370
SIZE_SMALL (*telegram.ChatPhoto attribute*), 370
SLEEPING_FACE (*telegram.constants.ReactionEmoji attribute*), 1020
SLOT_MACHINE (*telegram.constants.DiceEmoji attribute*), 975
SLOT_MACHINE (*telegram.Dice attribute*), 375
SLOT_MACHINE (*telegram.ext.filters.Dice attribute*), 899
slow_mode_delay (*telegram.ChatFullInfo attribute*), 311
SMALL (*telegram.constants.ChatPhotoSize attribute*), 972
small_file_id (*telegram.ChatPhoto attribute*), 370
small_file_unique_id (*telegram.ChatPhoto attribute*), 370
SMILING_FACE_WITH_HALO (*telegram.constants.ReactionEmoji attribute*), 1020
SMILING_FACE_WITH_HEART_SHAPED_EYES (*telegram.constants.ReactionEmoji attribute*), 1020
SMILING_FACE_WITH_HEARTS (*telegram.constants.ReactionEmoji attribute*), 1020
SMILING_FACE_WITH_HORNS (*telegram.constants.ReactionEmoji attribute*), 1020
SMILING_FACE_WITH_SUNGLASSES (*telegram.constants.ReactionEmoji attribute*), 1021
SNOWMAN (*telegram.constants.ReactionEmoji attribute*),

1021
socket_options() (*telegram.ext.ApplicationBuilder method*), 835
SOLID (*telegram.BackgroundFill attribute*), 296
SOLID (*telegram.constants.BackgroundFillType attribute*), 962
source (*telegram.ChatBoost attribute*), 299
source (*telegram.ChatBoostRemoved attribute*), 300
source (*telegram.ChatBoostSource attribute*), 300
source (*telegram.ChatBoostSourceGiftCode attribute*), 301
source (*telegram.ChatBoostSourceGiveaway attribute*), 302
source (*telegram.ChatBoostSourcePremium attribute*), 303
source (*telegram.PassportElementError attribute*), 788
source (*telegram.StarTransaction attribute*), 766
SPEAK_NO_EVIL_MONKEY (*telegram.constants.ReactionEmoji attribute*), 1021
SPOILER (*telegram.constants.MessageEntityType attribute*), 997
SPOILER (*telegram.MessageEntity attribute*), 534
sponsor_user (*telegram.TransactionPartnerAffiliateProgram attribute*), 771
SPOUTING_WHALE (*telegram.constants.ReactionEmoji attribute*), 1021
SQUARED_COOL (*telegram.constants.ReactionEmoji attribute*), 1021
star_amount (*telegram.SuggestedPostPaid attribute*), 603
star_count (*telegram.Gift attribute*), 679
star_count (*telegram.PaidMediaInfo attribute*), 552
StarAmount (*class in telegram*), 765
STARS_SIX_MONTHS (*telegram.constants.PremiumSubscription attribute*), 1013
STARS_THREE_MONTHS (*telegram.constants.PremiumSubscription attribute*), 1013
STARS_TWELVE_MONTHS (*telegram.constants.PremiumSubscription attribute*), 1014
start() (*telegram.ext.Application method*), 818
start() (*telegram.ext.JobQueue method*), 862
start_date (*telegram.VideoChatScheduled attribute*), 669
start_parameter (*telegram.InlineQueryResultsButton attribute*), 731
start_parameter (*telegram.Invoice attribute*), 754
start_polling() (*telegram.ext.Updater method*), 864
start_timestamp (*telegram.InputMediaVideo attribute*), 435
start_timestamp (*telegram.InputPaidMediaVideo attribute*), 439
start_timestamp (*telegram.Video attribute*), 667
start_webhook() (*telegram.ext.Updater method*), 865
StarTransaction (*class in telegram*), 765
StarTransactions (*class in telegram*), 767
StarTransactionsLimit (*class in telegram.constants*), 1023
state (*telegram.ext.ApplicationHandlerStop attribute*), 837
state (*telegram.LocationAddress attribute*), 459
state (*telegram.ResidentialAddress attribute*), 799
state (*telegram.ShippingAddress attribute*), 762
state (*telegram.SuggestedPostInfo attribute*), 602
states (*telegram.ext.ConversationHandler property*), 888
STATIC (*telegram.constants.InputProfilePhotoType attribute*), 984
STATIC (*telegram.constants.StickerFormat attribute*), 1024
STATIC (*telegram.ext.filters.Sticker attribute*), 911
STATIC (*telegram.InputProfilePhoto attribute*), 440
STATIC_THUMB_DIMENSIONS (*telegram.constants.StickerSetLimit attribute*), 1026
status (*telegram.ChatMember attribute*), 350
status (*telegram.ChatMemberAdministrator attribute*), 353
status (*telegram.ChatMemberBanned attribute*), 356
status (*telegram.ChatMemberLeft attribute*), 356
status (*telegram.ChatMemberMember attribute*), 357
status (*telegram.ChatMemberOwner attribute*), 358
status (*telegram.ChatMemberRestricted attribute*), 360
StatusUpdate (*class in telegram.ext.filters*), 907
Sticker (*class in telegram*), 684
Sticker (*class in telegram.ext.filters*), 911
sticker (*telegram.BusinessIntro attribute*), 237
STICKER (*telegram.constants.BaseInputMediaType attribute*), 964
STICKER (*telegram.constants.InlineQueryResultType attribute*), 981
STICKER (*telegram.constants.MessageAttachmentType attribute*), 993
STICKER (*telegram.constants.MessageType attribute*), 1006
sticker (*telegram.ExternalReplyInfo attribute*), 380
sticker (*telegram.Gift attribute*), 679
sticker (*telegram.InputSticker attribute*), 682
sticker (*telegram.Message attribute*), 479
sticker (*telegram.PollMedia attribute*), 570
sticker (*telegram.UniqueGiftModel attribute*), 620
sticker (*telegram.UniqueGiftSymbol attribute*), 621
sticker_file_id (*telegram.InlineQueryResultCachedSticker attribute*), 709
sticker_set_name (*telegram.ChatFullInfo attribute*), 312

sticker_type (*telegram.StickerSet attribute*), 688
StickerFormat (*class in telegram.constants*), 1023
StickerLimit (*class in telegram.constants*), 1024
stickers (*telegram.StickerSet attribute*), 688
StickerSet (*class in telegram*), 687
StickerSetLimit (*class in telegram.constants*), 1025
StickerType (*class in telegram.constants*), 1026
stop() (*telegram.ext.Application method*), 818
stop() (*telegram.ext.JobQueue method*), 862
stop() (*telegram.ext.Updater method*), 867
stop_live_location() (*telegram.Message method*), 524
stop_message_live_location() (*telegram.Bot method*), 206
stop_message_live_location() (*telegram.CallbackQuery method*), 247
stop_poll() (*telegram.Bot method*), 207
stop_poll() (*telegram.Message method*), 524
stop_running() (*telegram.ext.Application method*), 819
stopMessageLiveLocation() (*telegram.Bot method*), 205
stopPoll() (*telegram.Bot method*), 205
store_data (*telegram.ext.BasePersistence attribute*), 939
store_data (*telegram.ext.DictPersistence attribute*), 944
store_data (*telegram.ext.PicklePersistence attribute*), 949
Story (*class in telegram*), 592
STORY (*in module telegram.ext.filters*), 891
STORY (*telegram.constants.MessageAttachmentType attribute*), 994
STORY (*telegram.constants.MessageType attribute*), 1006
story (*telegram.ExternalReplyInfo attribute*), 381
story (*telegram.Message attribute*), 479
StoryArea (*class in telegram*), 594
StoryAreaPosition (*class in telegram*), 594
StoryAreaPositionLimit (*class in telegram.constants*), 1026
StoryAreaType (*class in telegram*), 595
StoryAreaTypeLimit (*class in telegram.constants*), 1026
StoryAreaTypeLink (*class in telegram*), 596
StoryAreaTypeLocation (*class in telegram*), 597
StoryAreaTypeSuggestedReaction (*class in telegram*), 597
StoryAreaTypeType (*class in telegram.constants*), 1027
StoryAreaTypeUniqueGift (*class in telegram*), 598
StoryAreaTypeWeather (*class in telegram*), 599
StoryLimit (*class in telegram.constants*), 1028
STRAWBERRY (*telegram.constants.ReactionEmoji attribute*), 1021
street (*telegram.LocationAddress attribute*), 459
street_line1 (*telegram.ResidentialAddress attribute*), 799
street_line1 (*telegram.ShippingAddress attribute*), 762
street_line2 (*telegram.ResidentialAddress attribute*), 799
street_line2 (*telegram.ShippingAddress attribute*), 763
strict (*telegram.ext.TypeHandler attribute*), 937
STRIKETHROUGH (*telegram.constants.MessageEntityType attribute*), 998
STRIKETHROUGH (*telegram.MessageEntity attribute*), 534
StringCommandHandler (*class in telegram.ext*), 933
StringRegexHandler (*class in telegram.ext*), 934
style (*telegram.InlineKeyboardButton attribute*), 405
style (*telegram.KeyboardButton attribute*), 448
subscription_expiration_date (*telegram.SuccessfulPayment attribute*), 768
SUBSCRIPTION_MAX_PRICE (*telegram.constants.InvoiceLimit attribute*), 987
subscription_period (*telegram.ChatInviteLink attribute*), 346
SUBSCRIPTION_PERIOD (*telegram.constants.ChatSubscriptionLimit attribute*), 973
SUBSCRIPTION_PERIOD (*telegram.constants.InvoiceLimit attribute*), 987
subscription_period (*telegram.TransactionPartnerUser attribute*), 775
subscription_price (*telegram.ChatInviteLink attribute*), 346
SUCCEEDED (*telegram.constants.RevenueWithdrawalStateType attribute*), 1023
SUCCEEDED (*telegram.RevenueWithdrawalState attribute*), 760
SUCCESS (*telegram.constants.KeyboardButtonStyle attribute*), 988
SUCCESSFUL_PAYMENT (*in module telegram.ext.filters*), 891
SUCCESSFUL_PAYMENT (*telegram.constants.MessageAttachmentType attribute*), 994
SUCCESSFUL_PAYMENT (*telegram.constants.MessageType attribute*), 1006
successful_payment (*telegram.Message attribute*), 482
SuccessfulPayment (*class in telegram*), 767
SuccessfulPayment (*class in telegram.ext.filters*), 911
suggested_name (*telegram.KeyboardButtonRequestManagedBot attribute*), 452
SUGGESTED_POST_APPROVAL_FAILED (*telegram.constants.MessageType attribute*),

1006
SUGGESTED_POST_APPROVAL_FAILED (*telegram.ext.filters.StatusUpdate attribute*), 910
suggested_post_approval_failed (*telegram.Message attribute*), 486
SUGGESTED_POST_APPROVED (*telegram.constants.MessageType attribute*), 1006
SUGGESTED_POST_APPROVED (*telegram.ext.filters.StatusUpdate attribute*), 910
suggested_post_approved (*telegram.Message attribute*), 486
SUGGESTED_POST_DECLINED (*telegram.constants.MessageType attribute*), 1006
SUGGESTED_POST_DECLINED (*telegram.ext.filters.StatusUpdate attribute*), 910
suggested_post_declined (*telegram.Message attribute*), 486
SUGGESTED_POST_INFO (*in module telegram.ext.filters*), 891
SUGGESTED_POST_INFO (*telegram.constants.MessageType attribute*), 1006
suggested_post_info (*telegram.Message attribute*), 478
suggested_post_message (*telegram.SuggestedPostApprovalFailed attribute*), 600
suggested_post_message (*telegram.SuggestedPostApproved attribute*), 600
suggested_post_message (*telegram.SuggestedPostDeclined attribute*), 601
suggested_post_message (*telegram.SuggestedPostPaid attribute*), 603
suggested_post_message (*telegram.SuggestedPostRefunded attribute*), 606
SUGGESTED_POST_PAID (*telegram.constants.MessageType attribute*), 1007
SUGGESTED_POST_PAID (*telegram.ext.filters.StatusUpdate attribute*), 910
suggested_post_paid (*telegram.Message attribute*), 487
SUGGESTED_POST_REFUNDED (*telegram.constants.MessageType attribute*), 1007
SUGGESTED_POST_REFUNDED (*telegram.ext.filters.StatusUpdate attribute*), 910
suggested_post_refunded (*telegram.Message attribute*), 487
SUGGESTED_REACTION (*telegram.constants.StoryAreaTypeType attribute*), 1027
SUGGESTED_REACTION (*telegram.StoryAreaType attribute*), 596
suggested_tip_amounts (*telegram.InputInvoiceMessageContent attribute*), 750
suggested_username (*telegram.KeyboardButtonRequestManagedBot attribute*), 452
SuggestedPost (*class in telegram.constants*), 1028
SuggestedPostApprovalFailed (*class in telegram*), 599
SuggestedPostApproved (*class in telegram*), 600
SuggestedPostDeclined (*class in telegram*), 601
SuggestedPostInfo (*class in telegram*), 602
SuggestedPostInfoState (*class in telegram.constants*), 1029
SuggestedPostPaid (*class in telegram*), 603
SuggestedPostParameters (*class in telegram*), 604
SuggestedPostPrice (*class in telegram*), 605
SuggestedPostRefunded (*class in telegram*), 606
SuggestedPostRefunded (*class in telegram.constants*), 1029
SUPER_GROUP (*telegram.ext.filters.SenderChat attribute*), 907
SUPERGROUP (*telegram.Chat attribute*), 251
SUPERGROUP (*telegram.ChatFullInfo attribute*), 314
SUPERGROUP (*telegram.constants.ChatType attribute*), 973
SUPERGROUP (*telegram.ext.filters.ChatType attribute*), 897
SUPERGROUP_CHAT_CREATED (*telegram.constants.MessageType attribute*), 1007
supergroup_chat_created (*telegram.Message attribute*), 481
SUPPORTED_WEBHOOK_PORTS (*in module telegram.constants*), 1023
supports_guest_queries (*telegram.User attribute*), 636
supports_inline_queries (*telegram.Bot property*), 207
supports_inline_queries (*telegram.User attribute*), 635
supports_streaming (*telegram.InputMediaVideo attribute*), 434
supports_streaming (*telegram.InputPaidMediaVideo attribute*), 440
SVG (*telegram.ext.filters.Document attribute*), 901
switch_inline_query (*telegram.InlineKeyboardButton attribute*), 404
switch_inline_query_chosen_chat (*telegram.InlineKeyboardButton attribute*),

405
switch_inline_query_current_chat (*telegram.InlineKeyboardButton attribute*), 405
SwitchInlineQueryChosenChat (*class in telegram*), 606
symbol (*telegram.UniqueGift attribute*), 614
symbol_color (*telegram.UniqueGiftBackdropColors attribute*), 617
symbol_custom_emoji_id (*telegram.UniqueGiftColors attribute*), 616

T

tag (*telegram.ChatMemberMember attribute*), 357
tag (*telegram.ChatMemberRestricted attribute*), 362
TagLimit (*class in telegram.constants*), 1030
TARGZ (*telegram.ext.filters.Document attribute*), 901
tasks (*telegram.Checklist attribute*), 286
tasks (*telegram.ChecklistTasksAdded attribute*), 289
tasks (*telegram.InputChecklist attribute*), 411
telegram
    module, 13
telegram.constants
    module, 960
telegram.error
    module, 1036
telegram.ext
    module, 803
telegram.ext.filters
    module, 888
telegram.helpers
    module, 1040
telegram.warnings
    module, 1049
TELEGRAM_ADS (*telegram.constants.TransactionPartnerType attribute*), 1030
TELEGRAM_ADS (*telegram.TransactionPartner attribute*), 770
TELEGRAM_API (*telegram.constants.TransactionPartnerType attribute*), 1031
TELEGRAM_API (*telegram.TransactionPartner attribute*), 770
telegram_payment_charge_id (*telegram.RefundedPayment attribute*), 759
telegram_payment_charge_id (*telegram.SuccessfulPayment attribute*), 769
TelegramError, 1039
TelegramObject (*class in telegram*), 608
temperature (*telegram.StoryAreaTypeWeather attribute*), 599
temporary_registration (*telegram.SecureData attribute*), 801
Text (*class in telegram.ext.filters*), 912
TEXT (*in module telegram.ext.filters*), 891
text (*telegram.ChecklistTask attribute*), 287
TEXT (*telegram.constants.MessageType attribute*), 1007
text (*telegram.CopyTextButton attribute*), 291
TEXT (*telegram.ext.filters.Document attribute*), 900
text (*telegram.Game attribute*), 778
text (*telegram.GiftInfo attribute*), 393
text (*telegram.InlineKeyboardButton attribute*), 404
text (*telegram.InlineQueryResultsButton attribute*), 731
text (*telegram.InputChecklistTask attribute*), 412
text (*telegram.InputPollOption attribute*), 442
text (*telegram.KeyboardButton attribute*), 447
text (*telegram.MenuButtonWebApp attribute*), 465
text (*telegram.Message attribute*), 477
text (*telegram.OwnedGiftRegular attribute*), 546
text (*telegram.PollOption attribute*), 571
text (*telegram.TextQuote attribute*), 612
text_color (*telegram.GiftBackground attribute*), 391
text_color (*telegram.UniqueGiftBackdropColors attribute*), 618
text_entities (*telegram.ChecklistTask attribute*), 288
text_entities (*telegram.Game attribute*), 778
text_entities (*telegram.InputChecklistTask attribute*), 412
text_entities (*telegram.InputPollOption attribute*), 443
text_entities (*telegram.PollOption attribute*), 571
text_html (*telegram.Message property*), 525
text_html_urled (*telegram.Message property*), 525
TEXT_LINK (*telegram.constants.MessageEntityType attribute*), 998
TEXT_LINK (*telegram.MessageEntity attribute*), 534
text_markdown (*telegram.Message property*), 525
text_markdown_urled (*telegram.Message property*), 526
text_markdown_v2 (*telegram.Message property*), 527
text_markdown_v2_urled (*telegram.Message property*), 527
TEXT_MENTION (*telegram.constants.MessageEntityType attribute*), 998
TEXT_MENTION (*telegram.MessageEntity attribute*), 534
text_parse_mode (*telegram.ext.Defaults property*), 848
text_parse_mode (*telegram.InputPollOption attribute*), 442
TextQuote (*class in telegram*), 612
theme_name (*telegram.BackgroundTypeChatTheme attribute*), 295
THINKING_FACE (*telegram.constants.ReactionEmoji attribute*), 1021
thumbnail (*telegram.Animation attribute*), 221
thumbnail (*telegram.Audio attribute*), 223
thumbnail (*telegram.Document attribute*), 378
thumbnail (*telegram.InputMediaAnimation attribute*), 419
thumbnail (*telegram.InputMediaAudio attribute*), 422
thumbnail (*telegram.InputMediaDocument attribute*), 424

thumbnail (*telegram.InputMediaVideo attribute*), 434
thumbnail (*telegram.InputPaidMediaVideo attribute*), 439
thumbnail (*telegram.Sticker attribute*), 687
thumbnail (*telegram.StickerSet attribute*), 689
thumbnail (*telegram.Video attribute*), 666
thumbnail (*telegram.VideoNote attribute*), 670
thumbnail_height (*telegram.InlineQueryResultArticle attribute*), 696
thumbnail_height (*telegram.InlineQueryResultContact attribute*), 715
thumbnail_height (*telegram.InlineQueryResultDocument attribute*), 718
thumbnail_height (*telegram.InlineQueryResultLocation attribute*), 724
thumbnail_height (*telegram.InlineQueryResultVenue attribute*), 734
thumbnail_mime_type (*telegram.InlineQueryResultGif attribute*), 721
thumbnail_mime_type (*telegram.InlineQueryResultMpeg4Gif attribute*), 727
thumbnail_url (*telegram.InlineQueryResultArticle attribute*), 696
thumbnail_url (*telegram.InlineQueryResultContact attribute*), 715
thumbnail_url (*telegram.InlineQueryResultDocument attribute*), 718
thumbnail_url (*telegram.InlineQueryResultGif attribute*), 721
thumbnail_url (*telegram.InlineQueryResultLocation attribute*), 724
thumbnail_url (*telegram.InlineQueryResultMpeg4Gif attribute*), 727
thumbnail_url (*telegram.InlineQueryResultPhoto attribute*), 729
thumbnail_url (*telegram.InlineQueryResultVenue attribute*), 734
thumbnail_url (*telegram.InlineQueryResultVideo attribute*), 736
thumbnail_width (*telegram.InlineQueryResultArticle attribute*), 696
thumbnail_width (*telegram.InlineQueryResultContact attribute*), 715
thumbnail_width (*telegram.InlineQueryResultDocument attribute*), 718
thumbnail_width (*telegram.InlineQueryResultLocation attribute*), 724
thumbnail_width (*telegram.InlineQueryResultVenue attribute*), 734
THUMBS_DOWN (*telegram.constants.ReactionEmoji attribute*), 1021
THUMBS_UP (*telegram.constants.ReactionEmoji attribute*), 1021
time_zone_name (*telegram.BusinessOpeningHours attribute*), 238
TimedOut, 1040
TIMEOUT (*telegram.ext.ConversationHandler attribute*), 886
title (*telegram.Audio attribute*), 223
title (*telegram.BusinessIntro attribute*), 236
title (*telegram.Chat attribute*), 250
title (*telegram.ChatFullInfo attribute*), 308
title (*telegram.ChatShared attribute*), 372
title (*telegram.Checklist attribute*), 285
title (*telegram.Game attribute*), 778
title (*telegram.InlineQueryResultArticle attribute*), 695
title (*telegram.InlineQueryResultAudio attribute*), 697
title (*telegram.InlineQueryResultCachedDocument attribute*), 701
title (*telegram.InlineQueryResultCachedGif attribute*), 703
title (*telegram.InlineQueryResultCachedMpeg4Gif attribute*), 705
title (*telegram.InlineQueryResultCachedPhoto attribute*), 707
title (*telegram.InlineQueryResultCachedVideo attribute*), 711
title (*telegram.InlineQueryResultCachedVoice attribute*), 713
title (*telegram.InlineQueryResultDocument attribute*), 717
title (*telegram.InlineQueryResultGif attribute*), 721
title (*telegram.InlineQueryResultLocation attribute*), 723
title (*telegram.InlineQueryResultMpeg4Gif attribute*), 727
title (*telegram.InlineQueryResultPhoto attribute*), 729
title (*telegram.InlineQueryResultVenue attribute*), 733
title (*telegram.InlineQueryResultVideo attribute*), 736
title (*telegram.InlineQueryResultVoice attribute*), 738
title (*telegram.InputChecklist attribute*), 411
title (*telegram.InputInvoiceMessageContent attribute*), 749
title (*telegram.InputMediaAudio attribute*), 422
title (*telegram.InputMediaVenue attribute*), 430
title (*telegram.InputVenueMessageContent attribute*), 745

title (*telegram.Invoice attribute*), 754
title (*telegram.ShippingOption attribute*), 763
title (*telegram.StickerSet attribute*), 688
title (*telegram.Venue attribute*), 664
title_entities (*telegram.Checklist attribute*), 286
title_entities (*telegram.InputChecklist attribute*), 411
to_date() (*telegram.Birthdate method*), 224
to_dict() (*telegram.Bot method*), 207
to_dict() (*telegram.TelegramObject method*), 611
to_json() (*telegram.TelegramObject method*), 611
token (*telegram.Bot property*), 207
token() (*telegram.ext.ApplicationBuilder method*), 835
top_color (*telegram.BackgroundFillGradient attribute*), 297
topic_id (*telegram.DirectMessagesTopic attribute*), 376
total_amount (*telegram.Invoice attribute*), 754
total_amount (*telegram.PreCheckoutQuery attribute*), 758
total_amount (*telegram.RefundedPayment attribute*), 759
total_amount (*telegram.SuccessfulPayment attribute*), 768
TOTAL_BUTTON_NUMBER (*telegram.constants.InlineKeyboardMarkupLimit attribute*), 979
total_count (*telegram.Gift attribute*), 680
total_count (*telegram.OwnedGifts attribute*), 549
total_count (*telegram.ReactionCount attribute*), 577
total_count (*telegram.UserProfileAudios attribute*), 660
total_count (*telegram.UserProfilePhotos attribute*), 661
total_voter_count (*telegram.Poll attribute*), 561
transaction_type (*telegram.TransactionPartnerUser attribute*), 775
TransactionPartner (*class in telegram*), 769
TransactionPartnerAffiliateProgram (*class in telegram*), 771
TransactionPartnerChat (*class in telegram*), 771
TransactionPartnerFragment (*class in telegram*), 772
TransactionPartnerOther (*class in telegram*), 773
TransactionPartnerTelegramAds (*class in telegram*), 773
TransactionPartnerTelegramApi (*class in telegram*), 773
TransactionPartnerType (*class in telegram.constants*), 1030
TransactionPartnerUser (*class in telegram*), 774
TransactionPartnerUser (*class in telegram.constants*), 1031
transactions (*telegram.StarTransactions attribute*), 767
TRANSFER (*telegram.constants.UniqueGiftInfoOrigin attribute*), 1032
TRANSFER (*telegram.UniqueGiftInfo attribute*), 620
transfer_business_account_stars() (*telegram.Bot method*), 208
transfer_gift() (*telegram.Bot method*), 208
transfer_gift() (*telegram.Chat method*), 277
transfer_gift() (*telegram.ChatFullInfo method*), 340
transfer_star_count (*telegram.OwnedGiftUnique attribute*), 550
transfer_star_count (*telegram.UniqueGiftInfo attribute*), 619
transferBusinessAccountStars() (*telegram.Bot method*), 208
transferGift() (*telegram.Bot method*), 208
translation (*telegram.EncryptedPassportElement attribute*), 784
translation (*telegram.SecureValue attribute*), 802
traveler (*telegram.ProximityAlertTriggered attribute*), 577
TROPHY (*telegram.constants.ReactionEmoji attribute*), 1021
TXT (*telegram.ext.filters.Document attribute*), 901
type (*telegram.BackgroundFill attribute*), 295
type (*telegram.BackgroundFillFreeformGradient attribute*), 298
type (*telegram.BackgroundFillGradient attribute*), 297
type (*telegram.BackgroundFillSolid attribute*), 296
type (*telegram.BackgroundType attribute*), 291
type (*telegram.BackgroundTypeChatTheme attribute*), 295
type (*telegram.BackgroundTypeFill attribute*), 292
type (*telegram.BackgroundTypePattern attribute*), 294
type (*telegram.BackgroundTypeWallpaper attribute*), 293
type (*telegram.BotCommandScope attribute*), 227
type (*telegram.BotCommandScopeAllChatAdministrators attribute*), 228
type (*telegram.BotCommandScopeAllGroupChats attribute*), 228
type (*telegram.BotCommandScopeAllPrivateChats attribute*), 229
type (*telegram.BotCommandScopeChat attribute*), 229
type (*telegram.BotCommandScopeChatAdministrators attribute*), 230
type (*telegram.BotCommandScopeChatMember attribute*), 230
type (*telegram.BotCommandScopeDefault attribute*), 231
type (*telegram.Chat attribute*), 250
type (*telegram.ChatBackground attribute*), 285
type (*telegram.ChatFullInfo attribute*), 308
type (*telegram.EncryptedPassportElement attribute*), 783
type (*telegram.ext.TypeHandler attribute*), 937
type (*telegram.InlineQueryResult attribute*), 694
type (*telegram.InlineQueryResultArticle attribute*), 695

type (*telegram.InlineQueryResultAudio attribute*), 697
type (*telegram.InlineQueryResultCachedAudio attribute*), 699
type (*telegram.InlineQueryResultCachedDocument attribute*), 701
type (*telegram.InlineQueryResultCachedGif attribute*), 703
type (*telegram.InlineQueryResultCachedMpeg4Gif attribute*), 705
type (*telegram.InlineQueryResultCachedPhoto attribute*), 707
type (*telegram.InlineQueryResultCachedSticker attribute*), 709
type (*telegram.InlineQueryResultCachedVideo attribute*), 710
type (*telegram.InlineQueryResultCachedVoice attribute*), 712
type (*telegram.InlineQueryResultContact attribute*), 714
type (*telegram.InlineQueryResultDocument attribute*), 716
type (*telegram.InlineQueryResultGame attribute*), 718
type (*telegram.InlineQueryResultGif attribute*), 720
type (*telegram.InlineQueryResultLocation attribute*), 723
type (*telegram.InlineQueryResultMpeg4Gif attribute*), 726
type (*telegram.InlineQueryResultPhoto attribute*), 729
type (*telegram.InlineQueryResultVenue attribute*), 733
type (*telegram.InlineQueryResultVideo attribute*), 735
type (*telegram.InlineQueryResultVoice attribute*), 738
type (*telegram.InputMedia attribute*), 416
type (*telegram.InputMediaAnimation attribute*), 418
type (*telegram.InputMediaAudio attribute*), 421
type (*telegram.InputMediaDocument attribute*), 423
type (*telegram.InputMediaLivePhoto attribute*), 425
type (*telegram.InputMediaLocation attribute*), 426
type (*telegram.InputMediaPhoto attribute*), 428
type (*telegram.InputMediaSticker attribute*), 429
type (*telegram.InputMediaVenue attribute*), 430
type (*telegram.InputMediaVideo attribute*), 433
type (*telegram.InputPaidMedia attribute*), 435
type (*telegram.InputPaidMediaLivePhoto attribute*), 436
type (*telegram.InputPaidMediaPhoto attribute*), 437
type (*telegram.InputPaidMediaVideo attribute*), 439
type (*telegram.InputProfilePhoto attribute*), 440
type (*telegram.InputProfilePhotoAnimated attribute*), 441
type (*telegram.InputProfilePhotoStatic attribute*), 442
type (*telegram.InputStoryContent attribute*), 443
type (*telegram.InputStoryContentPhoto attribute*), 444
type (*telegram.InputStoryContentVideo attribute*), 445
type (*telegram.KeyboardButtonPollType attribute*), 449
type (*telegram.MenuButton attribute*), 463
type (*telegram.MenuButtonCommands attribute*), 464
type (*telegram.MenuButtonDefault attribute*), 464
type (*telegram.MenuButtonWebApp attribute*), 465
type (*telegram.MessageEntity attribute*), 532
type (*telegram.MessageOrigin attribute*), 538
type (*telegram.MessageOriginChannel attribute*), 539
type (*telegram.MessageOriginChat attribute*), 540
type (*telegram.MessageOriginHiddenUser attribute*), 541
type (*telegram.MessageOriginUser attribute*), 541
type (*telegram.OwnedGift attribute*), 544
type (*telegram.OwnedGiftRegular attribute*), 546
type (*telegram.OwnedGiftUnique attribute*), 550
type (*telegram.PaidMedia attribute*), 551
type (*telegram.PaidMediaLivePhoto attribute*), 553
type (*telegram.PaidMediaPhoto attribute*), 553
type (*telegram.PaidMediaPreview attribute*), 554
type (*telegram.PaidMediaVideo attribute*), 556
type (*telegram.PassportElementError attribute*), 788
type (*telegram.PassportElementErrorDataField attribute*), 788
type (*telegram.PassportElementErrorFile attribute*), 789
type (*telegram.PassportElementErrorFiles attribute*), 790
type (*telegram.PassportElementErrorFrontSide attribute*), 791
type (*telegram.PassportElementErrorReverseSide attribute*), 791
type (*telegram.PassportElementErrorSelfie attribute*), 792
type (*telegram.PassportElementErrorTranslationFile attribute*), 793
type (*telegram.PassportElementErrorTranslationFiles attribute*), 794
type (*telegram.PassportElementErrorUnspecified attribute*), 795
type (*telegram.Poll attribute*), 561
type (*telegram.ReactionCount attribute*), 577
type (*telegram.ReactionType attribute*), 578
type (*telegram.ReactionTypeCustomEmoji attribute*), 579
type (*telegram.ReactionTypeEmoji attribute*), 580
type (*telegram.ReactionTypePaid attribute*), 580
type (*telegram.RevenueWithdrawalState attribute*), 760
type (*telegram.RevenueWithdrawalStateFailed attribute*), 761
type (*telegram.RevenueWithdrawalStatePending attribute*), 761
type (*telegram.RevenueWithdrawalStateSucceeded attribute*), 761
type (*telegram.Sticker attribute*), 686
type (*telegram.StoryArea attribute*), 594
type (*telegram.StoryAreaType attribute*), 596
type (*telegram.StoryAreaTypeLink attribute*), 596
type (*telegram.StoryAreaTypeLocation attribute*), 597
type (*telegram.StoryAreaTypeSuggestedReaction attribute*), 598
type (*telegram.StoryAreaTypeUniqueGift attribute*),

598
type (*telegram.StoryAreaTypeWeather attribute*), 599
type (*telegram.TransactionPartner attribute*), 770
type (*telegram.TransactionPartnerAffiliateProgram attribute*), 771
type (*telegram.TransactionPartnerChat attribute*), 772
type (*telegram.TransactionPartnerFragment attribute*), 772
type (*telegram.TransactionPartnerOther attribute*), 773
type (*telegram.TransactionPartnerTelegramAds attribute*), 773
type (*telegram.TransactionPartnerTelegramApi attribute*), 774
type (*telegram.TransactionPartnerUser attribute*), 775
TypeHandler (*class in telegram.ext*), 936
TYPING (*telegram.constants.ChatAction attribute*), 969
tzinfo (*telegram.ext.Defaults property*), 849

U

unban_chat() (*telegram.Chat method*), 278
unban_chat() (*telegram.ChatFullInfo method*), 341
unban_chat_member() (*telegram.Bot method*), 209
unban_chat_sender_chat() (*telegram.Bot method*), 210
unban_member() (*telegram.Chat method*), 278
unban_member() (*telegram.ChatFullInfo method*), 341
unban_sender_chat() (*telegram.Chat method*), 278
unban_sender_chat() (*telegram.ChatFullInfo method*), 341
unbanChatMember() (*telegram.Bot method*), 209
unbanChatSenderChat() (*telegram.Bot method*), 209
unclaimed_prize_count (*telegram.GiveawayCompleted attribute*), 397
unclaimed_prize_count (*telegram.GiveawayWinners attribute*), 399
UNCOMMON (*telegram.constants.UniqueGiftModelRarity attribute*), 1032
UNDERLINE (*telegram.constants.MessageEntityType attribute*), 998
UNDERLINE (*telegram.MessageEntity attribute*), 534
unhide_general_forum_topic() (*telegram.Bot method*), 211
unhide_general_forum_topic() (*telegram.Chat method*), 279
unhide_general_forum_topic() (*telegram.ChatFullInfo method*), 342
unhideGeneralForumTopic() (*telegram.Bot method*), 211
UNICORN_FACE (*telegram.constants.ReactionEmoji attribute*), 1021
UNIQUE (*telegram.constants.OwnedGiftType attribute*), 1009
UNIQUE (*telegram.OwnedGift attribute*), 544
UNIQUE_GIFT (*telegram.constants.MessageType attribute*), 1007
UNIQUE_GIFT (*telegram.constants.StoryAreaTypeType attribute*), 1027
UNIQUE_GIFT (*telegram.ext.filters.StatusUpdate attribute*), 910
unique_gift (*telegram.Message attribute*), 485
UNIQUE_GIFT (*telegram.StoryAreaType attribute*), 596
unique_gift_colors (*telegram.ChatFullInfo attribute*), 313
unique_gift_number (*telegram.GiftInfo attribute*), 393
unique_gift_number (*telegram.OwnedGiftRegular attribute*), 547
unique_gift_variant_count (*telegram.Gift attribute*), 681
unique_gifts (*telegram.AcceptedGiftTypes attribute*), 219
UniqueGift (*class in telegram*), 613
UniqueGiftBackdrop (*class in telegram*), 616
UniqueGiftBackdropColors (*class in telegram*), 617
UniqueGiftColors (*class in telegram*), 615
UniqueGiftInfo (*class in telegram*), 618
UniqueGiftInfoOrigin (*class in telegram.constants*), 1031
UniqueGiftModel (*class in telegram*), 620
UniqueGiftModelRarity (*class in telegram.constants*), 1032
UniqueGiftSymbol (*class in telegram*), 621
unix_time (*telegram.MessageEntity attribute*), 533
unlimited_gifts (*telegram.AcceptedGiftTypes attribute*), 219
unpin() (*telegram.Message method*), 528
unpin_all_chat_messages() (*telegram.Bot method*), 212
unpin_all_forum_topic_messages() (*telegram.Bot method*), 213
unpin_all_forum_topic_messages() (*telegram.Chat method*), 279
unpin_all_forum_topic_messages() (*telegram.ChatFullInfo method*), 342
unpin_all_forum_topic_messages() (*telegram.Message method*), 528
unpin_all_general_forum_topic_messages() (*telegram.Bot method*), 213
unpin_all_general_forum_topic_messages() (*telegram.Chat method*), 279
unpin_all_general_forum_topic_messages() (*telegram.ChatFullInfo method*), 342
unpin_all_messages() (*telegram.Chat method*), 279
unpin_all_messages() (*telegram.ChatFullInfo method*), 342
unpin_all_messages() (*telegram.User method*), 658
unpin_chat_message() (*telegram.Bot method*), 214
unpin_message() (*telegram.CallbackQuery method*), 248
unpin_message() (*telegram.Chat method*), 280
unpin_message() (*telegram.ChatFullInfo method*), 343
unpin_message() (*telegram.User method*), 659
unpinAllChatMessages() (*telegram.Bot method*), 212

unpinAllForumTopicMessages() (*telegram.Bot method*), 212
unpinAllGeneralForumTopicMessages() (*telegram.Bot method*), 212
unpinChatMessage() (*telegram.Bot method*), 212
unrestrict_boost_count (*telegram.ChatFullInfo attribute*), 311
until_date (*telegram.ChatMemberBanned attribute*), 356
until_date (*telegram.ChatMemberMember attribute*), 357
until_date (*telegram.ChatMemberRestricted attribute*), 361
Update (*class in telegram*), 622
update() (*telegram.ext.CallbackContext method*), 843
update_bot_data() (*telegram.ext.BasePersistence method*), 942
update_bot_data() (*telegram.ext.DictPersistence method*), 946
update_bot_data() (*telegram.ext.PicklePersistence method*), 951
update_callback_data() (*telegram.ext.BasePersistence method*), 942
update_callback_data() (*telegram.ext.DictPersistence method*), 947
update_callback_data() (*telegram.ext.PicklePersistence method*), 952
update_callback_data() (*telegram.InlineKeyboardButton method*), 406
update_chat_data() (*telegram.ext.BasePersistence method*), 942
update_chat_data() (*telegram.ext.DictPersistence method*), 947
update_chat_data() (*telegram.ext.PicklePersistence method*), 952
update_conversation() (*telegram.ext.BasePersistence method*), 942
update_conversation() (*telegram.ext.DictPersistence method*), 947
update_conversation() (*telegram.ext.PicklePersistence method*), 952
update_id (*telegram.Update attribute*), 624
update_interval (*telegram.ext.BasePersistence property*), 942
update_persistence() (*telegram.ext.Application method*), 819
update_processor (*telegram.ext.Application property*), 820
update_queue (*telegram.ext.Application attribute*), 804
update_queue (*telegram.ext.CallbackContext property*), 844
update_queue (*telegram.ext.Updater attribute*), 863
update_queue() (*telegram.ext.ApplicationBuilder method*), 836
update_user_data() (*telegram.ext.BasePersistence method*), 943
update_user_data() (*telegram.ext.DictPersistence method*), 947
update_user_data() (*telegram.ext.PicklePersistence method*), 952
UpdateFilter (*class in telegram.ext.filters*), 912
Updater (*class in telegram.ext*), 862
updater (*telegram.ext.Application attribute*), 805
updater() (*telegram.ext.ApplicationBuilder method*), 836
UpdateType (*class in telegram.constants*), 1032
UpdateType (*class in telegram.ext.filters*), 913
UPGRADE (*telegram.constants.UniqueGiftInfoOrigin attribute*), 1032
UPGRADE (*telegram.UniqueGiftInfo attribute*), 620
upgrade_gift() (*telegram.Bot method*), 215
upgrade_star_count (*telegram.Gift attribute*), 680
upgradeGift() (*telegram.Bot method*), 215
UPLOAD_DOCUMENT (*telegram.constants.ChatAction attribute*), 969
UPLOAD_PHOTO (*telegram.constants.ChatAction attribute*), 969
upload_sticker_file() (*telegram.Bot method*), 216
UPLOAD_VIDEO (*telegram.constants.ChatAction attribute*), 969
UPLOAD_VIDEO_NOTE (*telegram.constants.ChatAction attribute*), 969
UPLOAD_VOICE (*telegram.constants.ChatAction attribute*), 969
uploadStickerFile() (*telegram.Bot method*), 216
URL (*telegram.constants.MessageEntityType attribute*), 998
url (*telegram.InlineKeyboardButton attribute*), 404
url (*telegram.InlineQueryResultArticle attribute*), 696
url (*telegram.LinkPreviewOptions attribute*), 454
url (*telegram.LoginUrl attribute*), 460
URL (*telegram.MessageEntity attribute*), 534
url (*telegram.MessageEntity attribute*), 532
url (*telegram.RevenueWithdrawalStateSucceeded attribute*), 762
url (*telegram.StoryAreaTypeLink attribute*), 596
url (*telegram.WebAppInfo attribute*), 675
url (*telegram.WebhookInfo attribute*), 676
url_encoded_parameters() (*telegram.request.RequestData method*), 1047
User (*class in telegram*), 632
User (*class in telegram.ext.filters*), 914
USER (*in module telegram.ext.filters*), 891
user (*telegram.BusinessConnection attribute*), 236
user (*telegram.ChatBoostSourceGiftCode attribute*), 301
user (*telegram.ChatBoostSourceGiveaway attribute*), 302
user (*telegram.ChatBoostSourcePremium attribute*), 303
user (*telegram.ChatMember attribute*), 350
user (*telegram.ChatMemberAdministrator attribute*), 353
user (*telegram.ChatMemberBanned attribute*), 356
user (*telegram.ChatMemberLeft attribute*), 356

user (*telegram.ChatMemberMember attribute*), 357
user (*telegram.ChatMemberOwner attribute*), 358
user (*telegram.ChatMemberRestricted attribute*), 360
USER (*telegram.constants.MessageOriginType attribute*), 1000
USER (*telegram.constants.TransactionPartnerType attribute*), 1031
user (*telegram.DirectMessagesTopic attribute*), 376
user (*telegram.GameHighScore attribute*), 780
user (*telegram.ManagedBotUpdated attribute*), 461
user (*telegram.MessageEntity attribute*), 532
USER (*telegram.MessageOrigin attribute*), 538
user (*telegram.MessageReactionUpdated attribute*), 544
user (*telegram.PollAnswer attribute*), 568
USER (*telegram.TransactionPartner attribute*), 770
user (*telegram.TransactionPartnerUser attribute*), 775
user_administrator_rights (*telegram.KeyboardButtonRequestChat attribute*), 450
USER_AGENT (*telegram.request.BaseRequest attribute*), 1043
USER_ATTACHMENT (*in module telegram.ext.filters*), 891
user_chat_id (*telegram.BusinessConnection attribute*), 236
user_chat_id (*telegram.ChatJoinRequest attribute*), 347
user_data (*telegram.ext.Application attribute*), 805
user_data (*telegram.ext.CallbackContext property*), 844
user_data (*telegram.ext.ContextTypes property*), 845
user_data (*telegram.ext.DictPersistence property*), 947
user_data (*telegram.ext.PersistenceInput attribute*), 948
user_data_json (*telegram.ext.DictPersistence property*), 947
user_id (*telegram.BotCommandScopeChatMember attribute*), 231
user_id (*telegram.Contact attribute*), 373
user_id (*telegram.ext.Job attribute*), 853
user_id (*telegram.SharedUser attribute*), 591
user_ids (*telegram.ext.filters.User property*), 915
user_is_bot (*telegram.KeyboardButtonRequestUsers attribute*), 453
user_is_premium (*telegram.KeyboardButtonRequestUsers attribute*), 453
UserChatBoosts (*class in telegram*), 659
username (*telegram.Bot property*), 217
username (*telegram.Chat attribute*), 250
username (*telegram.ChatFullInfo attribute*), 308
username (*telegram.ChatShared attribute*), 372
username (*telegram.SharedUser attribute*), 592
username (*telegram.User attribute*), 635
usernames (*telegram.ext.filters.Chat property*), 896
usernames (*telegram.ext.filters.ForwardedFrom property*), 903
usernames (*telegram.ext.filters.SenderChat property*), 907
usernames (*telegram.ext.filters.User property*), 915
usernames (*telegram.ext.filters.ViaBot property*), 916
UserProfileAudios (*class in telegram*), 660
UserProfileAudiosLimit (*class in telegram.constants*), 1035
UserProfilePhotos (*class in telegram*), 661
UserProfilePhotosLimit (*class in telegram.constants*), 1035
UserRating (*class in telegram*), 661
users (*telegram.UsersShared attribute*), 663
users (*telegram.VideoChatParticipantsInvited attribute*), 668
USERS_SHARED (*telegram.constants.MessageType attribute*), 1007
USERS_SHARED (*telegram.ext.filters.StatusUpdate attribute*), 910
users_shared (*telegram.Message attribute*), 485
UsersShared (*class in telegram*), 662
utility_bill (*telegram.SecureData attribute*), 801

V

VALUE (*telegram.constants.Nanostar attribute*), 1008
value (*telegram.Dice attribute*), 374
VCARD (*telegram.constants.ContactLimit attribute*), 974
vcard (*telegram.Contact attribute*), 373
vcard (*telegram.InlineQueryResultContact attribute*), 715
vcard (*telegram.InputContactMessageContent attribute*), 747
Venue (*class in telegram*), 663
VENUE (*in module telegram.ext.filters*), 891
VENUE (*telegram.constants.BaseInputMediaType attribute*), 964
VENUE (*telegram.constants.InlineQueryResultType attribute*), 981
VENUE (*telegram.constants.MessageAttachmentType attribute*), 994
VENUE (*telegram.constants.MessageType attribute*), 1007
venue (*telegram.ExternalReplyInfo attribute*), 382
venue (*telegram.Message attribute*), 481
venue (*telegram.PollMedia attribute*), 570
verify() (*telegram.Chat method*), 280
verify() (*telegram.ChatFullInfo method*), 343
verify() (*telegram.User method*), 659
verify_chat() (*telegram.Bot method*), 217
verify_user() (*telegram.Bot method*), 218
verifyChat() (*telegram.Bot method*), 217
VerifyLimit (*class in telegram.constants*), 1035
verifyUser() (*telegram.Bot method*), 217
version (*telegram.warnings.PTBDeprecationWarning attribute*), 1050
VIA_BOT (*in module telegram.ext.filters*), 891
via_bot (*telegram.Message attribute*), 483
via_chat_folder_invite_link (*telegram.ChatMemberUpdated attribute*),

364
via_join_request (*telegram.ChatMemberUpdated attribute*), 364
ViaBot (*class in telegram.ext.filters*), 916
Video (*class in telegram*), 664
VIDEO (*in module telegram.ext.filters*), 891
VIDEO (*telegram.constants.BaseInputMediaType attribute*), 964
VIDEO (*telegram.constants.InlineQueryResultType attribute*), 981
VIDEO (*telegram.constants.InputMediaType attribute*), 983
VIDEO (*telegram.constants.InputPaidMediaType attribute*), 984
VIDEO (*telegram.constants.InputStoryContentType attribute*), 985
VIDEO (*telegram.constants.MessageAttachmentType attribute*), 994
VIDEO (*telegram.constants.MessageType attribute*), 1007
VIDEO (*telegram.constants.PaidMediaType attribute*), 1010
VIDEO (*telegram.constants.StickerFormat attribute*), 1024
VIDEO (*telegram.ext.filters.Document attribute*), 900
VIDEO (*telegram.ext.filters.Sticker attribute*), 911
video (*telegram.ExternalReplyInfo attribute*), 381
VIDEO (*telegram.InputPaidMedia attribute*), 436
VIDEO (*telegram.InputStoryContent attribute*), 443
video (*telegram.InputStoryContentVideo attribute*), 445
video (*telegram.Message attribute*), 479
VIDEO (*telegram.PaidMedia attribute*), 551
video (*telegram.PaidMediaVideo attribute*), 556
video (*telegram.PollMedia attribute*), 570
VIDEO_CHAT_ENDED (*telegram.constants.MessageType attribute*), 1007
VIDEO_CHAT_ENDED (*telegram.ext.filters.StatusUpdate attribute*), 910
video_chat_ended (*telegram.Message attribute*), 483
VIDEO_CHAT_PARTICIPANTS_INVITED (*telegram.constants.MessageType attribute*), 1008
VIDEO_CHAT_PARTICIPANTS_INVITED (*telegram.ext.filters.StatusUpdate attribute*), 911
video_chat_participants_invited (*telegram.Message attribute*), 483
VIDEO_CHAT_SCHEDULED (*telegram.constants.MessageType attribute*), 1008
VIDEO_CHAT_SCHEDULED (*telegram.ext.filters.StatusUpdate attribute*), 910
video_chat_scheduled (*telegram.Message attribute*), 483
VIDEO_CHAT_STARTED (*telegram.constants.MessageType attribute*), 1008
VIDEO_CHAT_STARTED (*telegram.ext.filters.StatusUpdate attribute*), 910
video_chat_started (*telegram.Message attribute*), 483
video_duration (*telegram.InlineQueryResultVideo attribute*), 737
video_file_id (*telegram.InlineQueryResultCachedVideo attribute*), 711
VIDEO_HEIGHT (*telegram.constants.InputStoryContentLimit attribute*), 985
video_height (*telegram.InlineQueryResultVideo attribute*), 737
VIDEO_NOTE (*in module telegram.ext.filters*), 891
VIDEO_NOTE (*telegram.constants.MessageAttachmentType attribute*), 994
VIDEO_NOTE (*telegram.constants.MessageType attribute*), 1008
video_note (*telegram.ExternalReplyInfo attribute*), 381
video_note (*telegram.Message attribute*), 480
video_url (*telegram.InlineQueryResultVideo attribute*), 736
VIDEO_WIDTH (*telegram.constants.InputStoryContentLimit attribute*), 985
video_width (*telegram.InlineQueryResultVideo attribute*), 736
VideoChatEnded (*class in telegram*), 667
VideoChatParticipantsInvited (*class in telegram*), 668
VideoChatScheduled (*class in telegram*), 668
VideoChatStarted (*class in telegram*), 669
VideoNote (*class in telegram*), 669
VideoQuality (*class in telegram*), 671
VIDEOSIZE_UPLOAD (*telegram.constants.InputStoryContentLimit attribute*), 985
Voice (*class in telegram*), 672
VOICE (*in module telegram.ext.filters*), 892
VOICE (*telegram.constants.InlineQueryResultType attribute*), 982
VOICE (*telegram.constants.MessageAttachmentType attribute*), 994
VOICE (*telegram.constants.MessageType attribute*), 1008
voice (*telegram.ExternalReplyInfo attribute*), 381
voice (*telegram.Message attribute*), 480
voice_duration (*telegram.InlineQueryResultVoice attribute*), 739
voice_file_id (*telegram.InlineQueryResultCachedVoice attribute*), 713
VOICE_NOTE_FILE_SIZE (*telegram.constants.FileSizeLimit attribute*), 976

voice_url (*telegram.InlineQueryResultVoice attribute*), 738
voter_chat (*telegram.PollAnswer attribute*), 568
voter_count (*telegram.PollOption attribute*), 571

W

WAITING (*telegram.ext.ConversationHandler attribute*), 886
WALLPAPER (*telegram.BackgroundType attribute*), 291
WALLPAPER (*telegram.constants.BackgroundTypeType attribute*), 963
was_refunded (*telegram.GiveawayWinners attribute*), 399
was_refunded (*telegram.OwnedGiftRegular attribute*), 547
watcher (*telegram.ProximityAlertTriggered attribute*), 577
WAV (*telegram.ext.filters.Document attribute*), 901
WEATHER (*telegram.constants.StoryAreaTypeType attribute*), 1027
WEATHER (*telegram.StoryAreaType attribute*), 596
WEB_APP (*telegram.constants.MenuButtonType attribute*), 992
web_app (*telegram.InlineKeyboardButton attribute*), 404
web_app (*telegram.InlineQueryResultsButton attribute*), 731
web_app (*telegram.KeyboardButton attribute*), 447
WEB_APP (*telegram.MenuButton attribute*), 463
web_app (*telegram.MenuButtonWebApp attribute*), 465
WEB_APP_DATA (*telegram.constants.MessageType attribute*), 1008
WEB_APP_DATA (*telegram.ext.filters.StatusUpdate attribute*), 911
web_app_data (*telegram.Message attribute*), 483
web_app_name (*telegram.WriteAccessAllowed attribute*), 678
WebAppData (*class in telegram*), 674
WebAppInfo (*class in telegram*), 674
WebhookInfo (*class in telegram*), 675
WebhookLimit (*class in telegram.constants*), 1036
width (*telegram.Animation attribute*), 221
width (*telegram.InputMediaAnimation attribute*), 419
width (*telegram.InputMediaVideo attribute*), 434
width (*telegram.InputPaidMediaVideo attribute*), 439
width (*telegram.LivePhoto attribute*), 456
width (*telegram.PaidMediaPreview attribute*), 554
width (*telegram.PhotoSize attribute*), 558
width (*telegram.Sticker attribute*), 686
width (*telegram.Video attribute*), 666
width (*telegram.VideoQuality attribute*), 672
width_percentage (*telegram.StoryAreaPosition attribute*), 595
winner_count (*telegram.Giveaway attribute*), 395
winner_count (*telegram.GiveawayCompleted attribute*), 397
winner_count (*telegram.GiveawayWinners attribute*), 399
winners (*telegram.GiveawayWinners attribute*), 399
winners_selection_date (*telegram.Giveaway attribute*), 395
winners_selection_date (*telegram.GiveawayWinners attribute*), 399
withdrawal_state (*telegram.TransactionPartnerFragment attribute*), 772
WOMAN_SHRUGGING (*telegram.constants.ReactionEmoji attribute*), 1022
WRITE_ACCESS_ALLOWED (*telegram.constants.MessageType attribute*), 1008
WRITE_ACCESS_ALLOWED (*telegram.ext.filters.StatusUpdate attribute*), 911
write_access_allowed (*telegram.Message attribute*), 485
write_timeout() (*telegram.ext.ApplicationBuilder method*), 836
WriteAccessAllowed (*class in telegram*), 677
WRITING_HAND (*telegram.constants.ReactionEmoji attribute*), 1022

X

x_percentage (*telegram.StoryAreaPosition attribute*), 595
x_shift (*telegram.MaskPosition attribute*), 683
XML (*telegram.ext.filters.Document attribute*), 901

Y

y_percentage (*telegram.StoryAreaPosition attribute*), 595
y_shift (*telegram.MaskPosition attribute*), 683
YAWNING_FACE (*telegram.constants.ReactionEmoji attribute*), 1022
year (*telegram.Birthdate attribute*), 224
YELLOW (*telegram.constants.ForumIconColor attribute*), 977

Z

ZERO_DATE (*in module telegram.constants*), 1036
ZIP (*telegram.ext.filters.Document attribute*), 901