



python-telegram-bot Documentation

Release 20.8

Leandro Toledo

Feb 08, 2024

REFERENCE

1	Note	3
2	Telegram API support	5
3	Installing	7
3.1	Verifying Releases	7
3.2	Dependencies & Their Versions	7
3.2.1	Optional Dependencies	8
4	Quick Start	9
5	Resources	11
6	Getting help	13
7	Concurrency	15
8	Contributing	17
9	Donating	19
10	License	21
10.1	telegram package	21
10.1.1	Version Constants	21
10.1.2	Classes in this package	22
10.2	telegram.ext package	516
10.2.1	Application	517
10.2.2	ApplicationBuilder	531
10.2.3	ApplicationHandlerStop	546
10.2.4	BaseUpdateProcessor	546
10.2.5	CallbackContext	548
10.2.6	ContextTypes	552
10.2.7	Defaults	553
10.2.8	ExtBot	557
10.2.9	Job	559
10.2.10	JobQueue	562
10.2.11	SimpleUpdateProcessor	569
10.2.12	Updater	569
10.2.13	Handlers	574
10.2.14	Persistence	630
10.2.15	Arbitrary Callback Data	644
10.2.16	Rate Limiting	647
10.3	Auxiliary modules	651
10.3.1	telegram.constants Module	651
10.3.2	telegram.error Module	945

10.3.3	telegram.helpers Module	948
10.3.4	telegram.request Module	950
10.3.5	telegram.warnings Module	957
10.4	Examples	957
10.4.1	echobot.py	957
10.4.2	timerbot.py	958
10.4.3	conversationbot.py	958
10.4.4	conversationbot2.py	958
10.4.5	nestedconversationbot.py	958
10.4.6	persistentconversationbot.py	958
10.4.7	inlinekeyboard.py	958
10.4.8	inlinekeyboard2.py	958
10.4.9	deeplinking.py	958
10.4.10	inlinebot.py	959
10.4.11	pollbot.py	959
10.4.12	passportbot.py	959
10.4.13	paymentbot.py	959
10.4.14	errorhandlerbot.py	959
10.4.15	chatmemberbot.py	959
10.4.16	webappbot.py	959
10.4.17	contexttypesbot.py	959
10.4.18	customwebhookbot.py	959
10.4.19	arbitrarycallbackdatabot.py	960
10.4.20	Pure API	960
10.5	Stability Policy	1028
10.5.1	What does this policy cover?	1028
10.5.2	What doesn't this policy cover?	1028
10.5.3	Versioning	1029
10.6	Changelog	1030
10.6.1	Version 20.8	1030
10.6.2	Version 20.7	1032
10.6.3	Version 20.6	1033
10.6.4	Version 20.5	1034
10.6.5	Version 20.4	1035
10.6.6	Version 20.3	1037
10.6.7	Version 20.2	1038
10.6.8	Version 20.1	1038
10.6.9	Version 20.0	1040
10.6.10	Version 20.0b0	1040
10.6.11	Version 20.0a6	1041
10.6.12	Version 20.0a5	1041
10.6.13	Version 20.0a4	1043
10.6.14	Version 20.0a3	1043
10.6.15	Version 20.0a2	1044
10.6.16	Version 20.0a1	1045
10.6.17	Version 20.0a0	1047
10.6.18	Version 13.11	1049
10.6.19	Version 13.10	1049
10.6.20	Version 13.9	1049
10.6.21	Version 13.8.1	1050
10.6.22	Version 13.8	1050
10.6.23	Version 13.7	1050
10.6.24	Version 13.6	1051
10.6.25	Version 13.5	1051
10.6.26	Version 13.4.1	1052
10.6.27	Version 13.4	1052
10.6.28	Version 13.3	1052
10.6.29	Version 13.2	1052

10.6.30	Version 13.1	1053
10.6.31	Version 13.0	1054
10.6.32	Version 12.8	1055
10.6.33	Version 12.7	1055
10.6.34	Version 12.6.1	1056
10.6.35	Version 12.6	1056
10.6.36	Version 12.5.1	1056
10.6.37	Version 12.5	1056
10.6.38	Version 12.4.2	1057
10.6.39	Version 12.4.1	1057
10.6.40	Version 12.4.0	1057
10.6.41	Version 12.3.0	1058
10.6.42	Version 12.2.0	1058
10.6.43	Version 12.1.1	1059
10.6.44	Version 12.1.0	1059
10.6.45	Version 12.0.0	1059
10.6.46	Version 11.1.0	1062
10.6.47	Version 11.0.0	1062
10.6.48	Version 10.1.0	1063
10.6.49	Version 10.0.2	1063
10.6.50	Version 10.0.1	1064
10.6.51	Version 10.0.0	1064
10.6.52	Version 9.0.0	1065
10.6.53	Version 8.1.1	1065
10.6.54	Version 8.1.0	1065
10.6.55	Version 8.0.0	1066
10.6.56	Version 7.0.1	1066
10.6.57	Version 7.0.0	1066
10.6.58	Pre-version 7.0	1067
10.7	Contributor Covenant Code of Conduct	1075
10.7.1	Our Pledge	1075
10.7.2	Our Standards	1075
10.7.3	Our Responsibilities	1075
10.7.4	Scope	1075
10.7.5	Enforcement	1076
10.7.6	Attribution	1076
10.8	How To Contribute	1076
10.8.1	Setting things up	1076
10.8.2	Finding something to do	1076
10.8.3	Instructions for making a code change	1077
10.8.4	Documenting	1079
10.8.5	Style commandments	1080
10.9	Testing in PTB	1081
10.9.1	Running tests	1081
10.9.2	Writing tests	1081
10.9.3	Bots used in tests	1082
Python Module Index		1083
Index		1085



python-telegram-bot

We have made you a wrapper you can't refuse

We have a vibrant community of developers helping each other in our [Telegram group](#). Join us!

Stay tuned for library updates and new releases on our [Telegram Channel](#).

This library provides a pure Python, asynchronous interface for the [Telegram Bot API](#). It's compatible with Python versions **3.8+**.

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

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

NOTE

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

TELEGRAM API SUPPORT

All types and methods of the Telegram Bot API **7.0** are supported.

INSTALLING

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

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

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

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

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

3.1 Verifying Releases

We sign all the releases 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` or `<first_version>-current.gpg` if the key is currently being used for new releases.

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

This allows you to verify that a release file that you downloaded was indeed provided by the `python-telegram-bot` team.

3.2 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 ~0.26.0` 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.

3.2.1 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~=1.1.0`. 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~=5.3.2` library. Use this, if you want to use `arbitrary callback_data`.
- `pip install "python-telegram-bot[job-queue]"` installs the `APScheduler~=3.10.4` library and enforces `pytz>=2018.6`, where `pytz` is a dependency of `APScheduler`. 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]`.

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.

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.

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](#).

CONCURRENCY

Since v20.0, `python-telegram-bot` is built on top of Python's `asyncio` module. Because `asyncio` is in general single-threaded, `python-telegram-bot` does currently not aim to be thread-safe. Noteworthy parts of `python-telegram-bot`'s 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

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](#).

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.

LICENSE

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

10.1 telegram package

10.1.1 Version Constants

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

```
telegram.__bot_api_version__ = '7.0'
```

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__ = (7, 0)
```

Shortcut for `telegram.constants.BOT_API_VERSION_INFO`.

New in version 20.0.

Type

`typing.NamedTuple`

```
telegram.__version__ = '20.8'
```

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__ = (20, 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.

New in version 20.0.

Type

`typing.NamedTuple`

10.1.2 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`, `typing.AsyncContextManager`

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()
```

Use In

`telegram.ext.ApplicationBuilder.bot()`

Available In

- `telegram.ext.Application.bot`
 - `telegram.ext.BasePersistence.bot`
 - `telegram.ext.CallbackContext.bot`
 - `telegram.ext.Updater.bot`
-

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

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

Changed in version 20.0:

- Removed the deprecated methods `kick_chat_member`, `kickChatMember`, `get_chat_members_count` and `getChatMembersCount`.
- Removed the deprecated property `commands`.
- Removed the deprecated `defaults` parameter. If you want to use `telegram.ext.Defaults`, please use the subclass `telegram.ext.ExtBot` instead.
- Attempting to pickle a bot instance will now raise `pickle.PicklingError`.
- 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`, optional) – Telegram Bot API service URL.
- **`base_file_url`** (`str`, optional) – Telegram Bot API file URL.
- **`request`** (`telegram.request.BaseRequest`, optional) – Pre initialized `telegram.request.BaseRequest` instances. Will be used for all bot methods *except* for `get_updates()`. If not passed, an instance of `telegram.request.HTTPXRequest` will be used.
- **`get_updates_request`** (`telegram.request.BaseRequest`, optional) – Pre initialized `telegram.request.BaseRequest` instances. Will be used exclusively for `get_updates()`. If not passed, an instance of `telegram.request.HTTPXRequest` will be used.
- **`private_key`** (`bytes`, optional) – Private key for decryption of telegram passport data.
- **`private_key_password`** (`bytes`, optional) – Password for above private key.
- **`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`.

New in version 20.0..

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

<code>answer_callback_query()</code>	Used for answering the callback query
<code>answer_inline_query()</code>	Used for answering the inline query
<code>answer_pre_checkout_query()</code>	Used for answering a pre checkout query
<code>answer_shipping_query()</code>	Used for answering a shipping query
<code>answer_web_app_query()</code>	Used for answering a web app query
<code>delete_message()</code>	Used for deleting messages.
<code>delete_messages()</code>	Used for deleting multiple messages as once.
<code>edit_message_caption()</code>	Used for editing captions
<code>edit_message_media()</code>	Used for editing the media on messages
<code>edit_message_location()</code>	Used for editing the location in live location messages
<code>edit_message_reply_markup()</code>	Used for editing the reply markup on messages
<code>edit_message_text()</code>	Used for editing text messages
<code>stop_poll()</code>	Used for stopping the running poll
<code>set_message_reaction()</code>	Used for setting reactions on messages

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

<code>set_my_commands</code>	Used for setting the list of commands
<code>delete_my_comma</code>	Used for deleting the list of commands
<code>get_my_commands</code>	Used for obtaining the list of commands
<code>get_my_default_</code>	Used for obtaining the default administrator rights for the bot
<code>set_my_default_</code>	Used for setting the default administrator rights for the bot
<code>get_chat_menu_b</code>	Used for obtaining the menu button of a private chat or the default menu button
<code>set_chat_menu_b</code>	Used for setting the menu button of a private chat or the default menu button
<code>set_my_descript</code>	Used for setting the description of the bot
<code>get_my_descript</code>	Used for obtaining the description of the bot
<code>set_my_short_de</code>	Used for setting the short description of the bot
<code>get_my_short_de</code>	Used for obtaining the short description of the bot
<code>set_my_name()</code>	Used for setting the name of the bot
<code>get_my_name()</code>	Used for obtaining the name of the bot

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

<code>get_game_high_s</code>	Used for getting the game high scores
<code>set_game_score(</code>	Used for setting the game score

<code>get_updates()</code>	Used for getting updates using long polling
<code>get_webhook_inf</code>	Used for getting current webhook status
<code>set_webhook()</code>	Used for setting a webhook to receive updates
<code>delete_webhook(</code>	Used for removing webhook integration

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

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

<code>base_file_url</code>	Telegram Bot API file URL
<code>base_url</code>	Telegram Bot API service URL
<code>bot</code>	The user instance of the bot as returned by <code>get_me()</code>
<code>can_join_groups</code>	Whether the bot can join groups
<code>can_read_all_gr</code>	Whether the bot can read all incoming group messages
<code>id</code>	The user id of the bot
<code>name</code>	The username of the bot, with leading @
<code>first_name</code>	The first name of the bot
<code>last_name</code>	The last name of the bot
<code>local_mode</code>	Whether the bot is running in local mode
<code>username</code>	The username of the bot, without leading @
<code>link</code>	The t.me link of the bot
<code>private_key</code>	Deserialized private key for decryption of telegram passport data
<code>supports_inline</code>	Whether the bot supports inline queries
<code>token</code>	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.

New 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.

New 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, *, 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. Animated and video sticker sets can have up to **50** stickers. Static 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.

New 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.

Deprecated since version 20.7: In future versions, the default value will be 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 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.

Parameters

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

Keyword Arguments

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

Returns

`bool` On success, `True` is returned.

Raises

`telegram.error.TelegramError` –

Shortcuts

`telegram.CallbackQuery.answer()`

```
async def 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.

Shortcuts

`telegram.InlineQuery.answer()`

See also:

[Working with Files and Media](#)

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`, optional) – The maximum amount of time in seconds that the result of the inline query may be cached on the server. Defaults to **300**.
- **is_personal** (`bool`, optional) – Pass `True`, if results may be cached on the server side only for the user that sent the query. By default, results may be returned to any user who sends the same query.
- **next_offset** (`str`, optional) – Pass the offset that a client should send in the next query with the same text to receive more results. Pass an empty string if there are no more results or if you don't support pagination. Offset length can't exceed **64** bytes.
- **button** (`telegram.InlineQueryResultsButton`, optional) – A button to be shown above the inline query results.

New 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.

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` –

Shortcuts

`telegram.PreCheckoutQuery.answer()`

```
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.

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` –

Shortcuts

`telegram.ShippingQuery.answer()`

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

Use this method to set the result of an interaction with a Web App and send a corresponding message on behalf of the user to the chat from which the query originated.

New in version 20.0.

Parameters

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

Keyword Arguments

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

Returns

On success, a sent `telegram.SentWebAppMessage` is returned.

Return type

`telegram.SentWebAppMessage`

Raises

`telegram.error.TelegramError` –

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

Alias for `approve_chat_join_request()`

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

Use this method to approve a chat join request.

The bot must be an administrator in the chat for this to work and must have the `telegram.ChatPermissions.can_invite_users` administrator right.

Shortcuts

- `telegram.Chat.approve_join_request()`
- `telegram.ChatJoinRequest.approve()`
- `telegram.User.approve_join_request()`

New in version 13.8.

Parameters

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

Keyword Arguments

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

New in version 13.7.

Parameters

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

- **`until_date`** (`int` | `datetime.datetime`, optional) – Date when the user will be unbanned, unix time. If user is banned for more than 366 days or less than 30 seconds from the current time they are considered to be banned forever. Applied for supergroups and channels only. For timezone naive `datetime.datetime` objects, the default timezone of the bot will be used, which is UTC unless `telegram.ext.Defaults.tzinfo` is used.
- **`revoke_messages`** (`bool`, optional) – Pass `True` to delete all messages from the chat for the user that is being removed. If `False`, the user will be able to see messages in the group that were sent before the user was removed. Always `True` for supergroups and channels.

New in version 13.4.

Keyword Arguments

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

New in version 13.9.

Parameters

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

Keyword Arguments

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

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

New in version 20.0.

Type

`str`

property `base_url`

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

New 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

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.Message.close_forum_topic()`
-

New 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()`

New 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 copyMessage(chat_id, from_chat_id, message_id, caption=None, parse_mode=None,
                  caption_entities=None, disable_notification=None, reply_to_message_id=None,
                  allow_sending_without_reply=None, reply_markup=None, protect_content=None,
                  message_thread_id=None, reply_parameters=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, *,
                   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_to_message_id=None,
                  allow_sending_without_reply=None, reply_markup=None,
                  protect_content=None, message_thread_id=None, reply_parameters=None, *,
                  read_timeout=None, write_timeout=None, connect_timeout=None,
                  pool_timeout=None, api_kwargs=None)
```

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

Parameters

- **chat_id** (`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **from_chat_id** (`int` | `str`) – Unique identifier for the chat where the original message was sent (or channel username in the format `@channelusername`).
- **message_id** (`int`) – Message identifier in the chat specified in `from_chat_id`.
- **caption** (`str`, optional) – New caption for media, 0-1024 characters after entities parsing. If not specified, the original caption is kept.
- **parse_mode** (`str`, optional) – Mode for parsing entities in the new caption. See the constants in `telegram.constants.ParseMode` for the available modes.
- **caption_entities** (`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.

New in version 13.10.

- **message_thread_id** (`int`, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.

New in version 20.0.

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

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- `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`.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- `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.

New in version 20.8.

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

Return type

`telegram.MessageId`

Raises

`telegram.error.TelegramError` –

Shortcuts

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

```
async copy_messages(chat_id, from_chat_id, message_ids, disable_notification=None,
                    protect_content=None, message_thread_id=None, remove_caption=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, 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 `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.User.copy_messages()`
 - `telegram.User.send_copies()`
-

New 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]`) – Identifiers of `1 - 100` 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 the forum; for forum supergroups only.
- **`remove_caption`** (`bool`, optional) – Pass `True` to copy the messages without their captions.

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 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, provider_token, currency, prices,
                        max_tip_amount=None, suggested_tip_amounts=None,
                        provider_data=None, photo_url=None, photo_size=None,
                        photo_width=None, photo_height=None, need_name=None,
                        need_phone_number=None, need_email=None,
                        need_shipping_address=None, send_phone_number_to_provider=None,
                        send_email_to_provider=None, is_flexible=None, *, read_timeout=None,
                        write_timeout=None, connect_timeout=None, pool_timeout=None,
                        api_kwargs=None)
```

Alias for `create_invoice_link()`

```
async createNewStickerSet(user_id, name, title, stickers, sticker_format, 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 undocumented and may be subject to change. See [this GitHub thread](#) for some discussion.

Shortcuts

`telegram.Chat.create_invite_link()`

New in version 13.4.

Parameters

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

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

New in version 13.8.

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

New in version 13.8.

Keyword Arguments

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

New 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, provider_token, currency, prices, max_tip_amount=None, suggested_tip_amounts=None, provider_data=None, photo_url=None, photo_size=None, photo_width=None, photo_height=None, need_name=None, need_phone_number=None, need_email=None, need_shipping_address=None, send_phone_number_to_provider=None, send_email_to_provider=None, is_flexible=None, *, read_timeout=None, write_timeout=None, connect_timeout=None, pool_timeout=None, api_kwargs=None)

Use this method to create a link for an invoice.

New in version 20.0.

Parameters

- **title** (str) – Product name. 1- 32 characters.
- **description** (str) – Product description. 1- 255 characters.
- **payload** (str) – Bot-defined invoice payload. 1- 128 bytes. This will not be displayed to the user, use for your internal processes.
- **provider_token** (str) – Payments provider token, obtained via `@BotFather`.
- **currency** (str) – Three-letter ISO 4217 currency code, see [more on currencies](#).
- **prices** (Sequence[`telegram.LabeledPrice`]) – Price breakdown, a sequence of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.).

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

Keyword Arguments

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

Returns

On success, the created invoice link is returned.

Return type

str

```
async create_new_sticker_set(user_id, name, title, stickers, sticker_format, 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.

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.

New in version 20.2.

- **sticker_format** (`str`) – Format of stickers in the set, must be one of `STATIC`, `ANIMATED` or `VIDEO`.

New 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

New 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.

New 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.

Deprecated since version 20.7: In future versions, the default value will be 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 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.ChatJoinRequest.decline()`
 - `telegram.User.decline_join_request()`
-

New in version 13.8.

Parameters

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

Keyword Arguments

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

Returns

On success, `True` is returned.

Return type

`bool`

Raises

`telegram.error.TelegramError` –

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

Alias for `delete_chat_photo()`

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

Alias for `delete_chat_sticker_set()`

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

Alias for `delete_webhook()`

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

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

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` –

Shortcuts

`telegram.Chat.delete_photo()`

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

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

Parameters

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

Keyword Arguments

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

Returns

On success, `True` is returned.

Return type

`bool`

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. The bot must be an administrator in the chat for this to work and must have `can_delete_messages` administrator rights.

Shortcuts

- `telegram.Chat.delete_forum_topic()`
 - `telegram.Message.delete_forum_topic()`
-

New 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.

Shortcuts

- `telegram.Chat.delete_message()`
 - `telegram.Message.delete()`
 - `telegram.User.delete_message()`
-

See also:

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

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_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.User.delete_messages()`
-

New 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]) – Identifiers of 1- 100 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.

New 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](#)) – File identifier of the sticker.

Keyword Arguments

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

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

New 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_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.
-

New 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.

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**.

Raises

`telegram.error.TelegramError` –

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

Alias for `edit_chat_invite_link()`

```
async 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, *, read_timeout=None, write_timeout=None,
                          connect_timeout=None, pool_timeout=None, api_kwargs=None)
```

Alias for `edit_message_caption()`

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

Alias for `edit_message_live_location()`

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

Alias for `edit_message_media()`

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

Alias for `edit_message_reply_markup()`

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

Alias for `edit_message_text()`

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

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

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

Shortcuts

`telegram.Chat.edit_invite_link()`

New in version 13.4.

Parameters

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

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

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

New in version 13.8.

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

New in version 13.8.

Keyword Arguments

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

New 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 `can_manage_topics` administrator rights.

Shortcuts

`telegram.Chat.edit_general_forum_topic()`

New 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, *, 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.

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.

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` –

Shortcuts

- `telegram.CallbackQuery.edit_message_caption()`
 - `telegram.Message.edit_caption()`
-

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

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

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

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.

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

Shortcuts

- *telegram.CallbackQuery.edit_message_live_location()*
- *telegram.Message.edit_live_location()*

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

Use this method to edit animation, audio, document, photo, or video messages. If a message is part of a message album, then it can be edited only to an audio for audio albums, only to a document for document albums and to a photo or a video otherwise. When an inline message is edited, a new file can't be uploaded; use a previously uploaded file via its *file_id* or specify a URL.

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

Shortcuts

- *telegram.CallbackQuery.edit_message_media()*
 - *telegram.Message.edit_media()*
-

See also:

Working with Files and Media

Parameters

- **media** (*telegram.InputMedia*) – An object for a new media content of the message.
- **chat_id** (int | str, optional) – Required if *inline_message_id* is not specified. Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **message_id** (int, optional) – Required if *inline_message_id* is not specified. Identifier of the message to edit.

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

Keyword Arguments

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

Parameters

- **`chat_id`** (`int` | `str`, optional) – Required if `inline_message_id` is not specified. Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **`message_id`** (`int`, optional) – Required if `inline_message_id` is not specified. Identifier of the message to edit.
- **`inline_message_id`** (`str`, optional) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.
- **`reply_markup`** (`telegram.InlineKeyboardMarkup`, optional) – An object for an inline keyboard.

Keyword Arguments

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

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

Shortcuts

- `telegram.CallbackQuery.edit_message_reply_markup()`
- `telegram.Message.edit_reply_markup()`

async `edit_message_text`(`text`, `chat_id=None`, `message_id=None`, `inline_message_id=None`, `parse_mode=None`, `disable_web_page_preview=None`, `reply_markup=None`, `entities=None`, `link_preview_options=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..

Shortcuts

- `telegram.CallbackQuery.edit_message_text()`
 - `telegram.Message.edit_text()`
-

See also:

`telegram.Game.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](#).

New in version 20.8.

- **disable_web_page_preview** (bool, optional) – Disables link previews for links in this message. 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.

Deprecated since version 20.8: In future versions, this argument will become keyword only.

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

Keyword Arguments

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

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` –

Shortcuts

`telegram.Chat.export_invite_link()`

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

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

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

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

Parameters

- **`chat_id`** (`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **`from_chat_id`** (`int` | `str`) – Unique identifier for the chat where the original message was sent (or channel username in the format @channelusername).
- **`message_id`** (`int`) – Message identifier in the chat specified in `from_chat_id`.
- **`disable_notification`** (`bool`, optional) – Sends the message silently. Users will receive a notification with no sound.
- **`protect_content`** (`bool`, optional) – Protects the contents of the sent message from forwarding and saving.

New in version 13.10.

- **`message_thread_id`** (`int`, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.

New in version 20.0.

Keyword Arguments

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

Returns

On success, the sent Message is returned.

Return type

`telegram.Message`

Raises

`telegram.error.TelegramError` –

Shortcuts

- `telegram.Chat.forward_from()`
- `telegram.Chat.forward_to()`
- `telegram.Message.forward()`
- `telegram.User.forward_from()`
- `telegram.User.forward_to()`

```
async forward_messages(chat_id, from_chat_id, message_ids, disable_notification=None,
                       protect_content=None, message_thread_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.User.forward_messages_from()`
 - `telegram.User.forward_messages_to()`
-

New 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]`) – Identifiers of 1- 100 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 the forum; for forum supergroups only.

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** **getChat**(*chat_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*,
pool_timeout=None, *api_kwargs=None*)Alias for `get_chat()`**async** **getChatAdministrators**(*chat_id*, *, *read_timeout=None*, *write_timeout=None*,
connect_timeout=None, *pool_timeout=None*, *api_kwargs=None*)Alias for `get_chat_administrators()`**async** **getChatMember**(*chat_id*, *user_id*, *, *read_timeout=None*, *write_timeout=None*,
connect_timeout=None, *pool_timeout=None*, *api_kwargs=None*)Alias for `get_chat_member()`**async** **getChatMemberCount**(*chat_id*, *, *read_timeout=None*, *write_timeout=None*,
connect_timeout=None, *pool_timeout=None*, *api_kwargs=None*)Alias for `get_chat_member_count()`**async** **getChatMenuButton**(*chat_id=None*, *, *read_timeout=None*, *write_timeout=None*,
connect_timeout=None, *pool_timeout=None*, *api_kwargs=None*)Alias for `get_chat_menu_button()`**async** **getCustomEmojiStickers**(*custom_emoji_ids*, *, *read_timeout=None*, *write_timeout=None*,
connect_timeout=None, *pool_timeout=None*, *api_kwargs=None*)Alias for `get_custom_emoji_stickers()`**async** **getFile**(*file_id*, *, *read_timeout=None*, *write_timeout=None*, *connect_timeout=None*,
pool_timeout=None, *api_kwargs=None*)Alias for `get_file()`**async** **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** **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 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 getUserProfilePhotos(user_id, offset=None, limit=None, *, read_timeout=None,
                           write_timeout=None, connect_timeout=None, pool_timeout=None,
                           api_kwargs=None)
```

Alias for `get_user_profile_photos()`

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

Alias for `get_webhook_info()`

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

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

Parameters

chat_id (`int` | `str`) – Unique identifier for the target chat or username of the target 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.Chat`

Raises

`telegram.error.TelegramError` –

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

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

Shortcuts

`telegram.Chat.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).

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_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.

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` –

Shortcuts

`telegram.Chat.get_member()`

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()`

New 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.

Shortcuts

- `telegram.Chat.get_menu_button()`
 - `telegram.User.get_menu_button()`
-

See also:

`set_chat_menu_button()`, `telegram.Chat.set_menu_button()`, `telegram.User.set_menu_button()`

New in version 20.0.

Parameters

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

Keyword Arguments

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

Returns

On success, the current menu button is returned.

Return type

`telegram.MenuButton`

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

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

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.

ReturnsTuple[[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.

Shortcuts

- [telegram.ChatPhoto.get_big_file\(\)](#)
 - [telegram.ChatPhoto.get_small_file\(\)](#)
-

See also:[Working with Files and Media](#)**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.

New in version 20.0.

Keyword Arguments

- **`read_timeout`** (float | None, optional) – Value to pass to `telegram.request.BaseRequest.post.read_timeout`. Defaults to `DEFAULT_NONE`.
- **`write_timeout`** (float | None, optional) – Value to pass to `telegram.request.BaseRequest.post.write_timeout`. Defaults to `DEFAULT_NONE`.
- **`connect_timeout`** (float | None, optional) – Value to pass to `telegram.request.BaseRequest.post.connect_timeout`. Defaults to `DEFAULT_NONE`.
- **`pool_timeout`** (float | None, optional) – Value to pass to `telegram.request.BaseRequest.post.pool_timeout`. Defaults to `DEFAULT_NONE`.
- **`api_kwargs`** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API. 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_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`.
New in version 13.7.
- **language_code** (str, optional) – A two-letter ISO 639-1 language code or an empty string.
New in version 13.7.

Keyword Arguments

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

New in version 20.0.

Parameters

`for_channels` (bool, optional) – Pass `True` to get default administrator rights of the bot in channels. Otherwise, default administrator rights of the bot for groups and supergroups will be returned.

Keyword Arguments

- **`read_timeout`** (float | None, optional) – Value to pass to `telegram.request.BaseRequest.post.read_timeout`. Defaults to `DEFAULT_NONE`.
- **`write_timeout`** (float | None, optional) – Value to pass to `telegram.request.BaseRequest.post.write_timeout`. Defaults to `DEFAULT_NONE`.
- **`connect_timeout`** (float | None, optional) – Value to pass to `telegram.request.BaseRequest.post.connect_timeout`. Defaults to `DEFAULT_NONE`.
- **`pool_timeout`** (float | None, optional) – Value to pass to `telegram.request.BaseRequest.post.pool_timeout`. Defaults to `DEFAULT_NONE`.
- **`api_kwargs`** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API. 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_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](#), optional) – Timeout in seconds for long polling. Defaults to 0, i.e. usual short polling. Should be positive, short polling should be used for testing purposes only.
- **allowed_updates** ([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](#) (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.User.get_chat_boosts()`
-

New 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_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.

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` –

Shortcuts

`telegram.User.get_profile_photos()`

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`

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\(\)](#)

New 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\(\)](#)

New in version 20.0.

property last_name

Optional. Bot’s last name. Shortcut for the corresponding attribute of [bot](#).

Type

`str`


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

Alias for `leave_chat()`

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

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

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` –

Shortcuts

`telegram.Chat.leave()`

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.

New 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

Return type

`True`

Raises

`telegram.error.TelegramError` –

property name

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

Type

`str`

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

Alias for `pin_chat_message()`

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

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

Parameters

- **`chat_id`** (`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **`message_id`** (`int`) – Identifier of a message to pin.
- **`disable_notification`** (`bool`, optional) – Pass `True`, if it is not necessary to send a notification to all chat members about the new pinned message. Notifications are always disabled in channels and private chats.

Keyword Arguments

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

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

Returns

On success, `True` is returned.

Return type

`bool`

Raises

`telegram.error.TelegramError` –

Shortcuts

- `telegram.Chat.pin_message()`
 - `telegram.Message.pin()`
 - `telegram.User.pin_message()`
-

property private_key

Deserialized private key for decryption of telegram passport data.

New 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, *, *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, *, *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()`

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

Parameters

- **chat_id** (`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **user_id** (`int`) – Unique identifier of the target user.
- **is_anonymous** (`bool`, optional) – Pass `True`, if the administrator's presence in the chat is hidden.
- **can_manage_chat** (`bool`, optional) – Pass `True`, if the administrator can access the chat event log, chat statistics, boost list in channels, see channel members, report spam messages, see anonymous administrators in supergroups and ignore slow mode. Implied by any other administrator privilege.

New in version 13.4.

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

New in version 20.0.

- **can_change_info** (`bool`, optional) – Pass `True`, if the administrator can change chat title, photo and other settings.
- **can_post_messages** (`bool`, optional) – Pass `True`, if the administrator can post messages in the channel, or access channel statistics; channels only.
- **can_edit_messages** (`bool`, optional) – Pass `True`, if the administrator can edit messages of other users and can pin messages, channels only.
- **can_delete_messages** (`bool`, optional) – Pass `True`, if the administrator can delete messages of other users.
- **can_invite_users** (`bool`, optional) – Pass `True`, if the administrator can invite new users to the chat.
- **can_restrict_members** (`bool`, optional) – Pass `True`, if the administrator can restrict, ban or unban chat members, or access supergroup statistics.
- **can_pin_messages** (`bool`, optional) – Pass `True`, if the administrator can pin messages, supergroups only.
- **can_promote_members** (`bool`, optional) – Pass `True`, if the administrator can add new administrators with a subset of 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; supergroups only.

New in version 20.0.

- **can_post_stories** (`bool`, optional) – Pass `True`, if the administrator can post stories in the channel; channels only.

New in version 20.6.

- **can_edit_stories** (`bool`, optional) – Pass `True`, if the administrator can edit stories posted by other users; channels only.

New in version 20.6.

- **can_delete_stories** (`bool`, optional) – Pass `True`, if the administrator can delete stories posted by other users; channels only.

New in version 20.6.

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, `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.Message.reopen_forum_topic()`
-

New 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()`

New 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 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.

Shortcuts

`telegram.Chat.restrict_member()`

See also:

`telegram.ChatPermissions.all_permissions()`

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()`

New in version 13.4.

Parameters

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

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

Keyword Arguments

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

Returns

`telegram.ChatInviteLink`

Raises

`telegram.error.TelegramError` –

async sendAnimation(*chat_id*, *animation*, *duration=None*, *width=None*, *height=None*, *caption=None*, *parse_mode=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *thumbnail=None*, *reply_parameters=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_to_message_id=None, reply_markup=None,
                 parse_mode=None, allow_sending_without_reply=None, caption_entities=None,
                 protect_content=None, message_thread_id=None, thumbnail=None,
                 reply_parameters=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, *, read_timeout=None,
                      write_timeout=None, connect_timeout=None, pool_timeout=None,
                      api_kwargs=None)
```

Alias for `send_chat_action()`

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

Alias for `send_contact()`

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

Alias for `send_dice()`

```
async sendDocument(chat_id, document, caption=None, disable_notification=None,
                    reply_to_message_id=None, reply_markup=None, parse_mode=None,
                    disable_content_type_detection=None, allow_sending_without_reply=None,
                    caption_entities=None, protect_content=None, message_thread_id=None,
                    thumbnail=None, reply_parameters=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_to_message_id=None,
                reply_markup=None, allow_sending_without_reply=None, protect_content=None,
                message_thread_id=None, reply_parameters=None, *, read_timeout=None,
                write_timeout=None, connect_timeout=None, pool_timeout=None,
                api_kwargs=None)
```

Alias for `send_game()`

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

Alias for `send_invoice()`


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

Alias for [send_location\(\)](#)

```
async sendMediaGroup(chat_id, media, disable_notification=None, reply_to_message_id=None,
                      allow_sending_without_reply=None, protect_content=None,
                      message_thread_id=None, reply_parameters=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_web_page_preview=None, disable_notification=None,
                   protect_content=None, reply_to_message_id=None,
                   allow_sending_without_reply=None, reply_markup=None,
                   message_thread_id=None, link_preview_options=None, reply_parameters=None,
                   *, read_timeout=None, write_timeout=None, connect_timeout=None,
                   pool_timeout=None, api_kwargs=None)
```

Alias for [send_message\(\)](#)

```
async sendPhoto(chat_id, photo, caption=None, disable_notification=None,
                 reply_to_message_id=None, reply_markup=None, parse_mode=None,
                 allow_sending_without_reply=None, caption_entities=None, protect_content=None,
                 message_thread_id=None, has_spoiler=None, reply_parameters=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_to_message_id=None, reply_markup=None,
                explanation=None, explanation_parse_mode=None, open_period=None,
                close_date=None, allow_sending_without_reply=None, explanation_entities=None,
                protect_content=None, message_thread_id=None, reply_parameters=None, *,
                read_timeout=None, write_timeout=None, connect_timeout=None,
                pool_timeout=None, api_kwargs=None)
```

Alias for [send_poll\(\)](#)

```
async sendSticker(chat_id, sticker, disable_notification=None, reply_to_message_id=None,
                  reply_markup=None, allow_sending_without_reply=None, protect_content=None,
                  message_thread_id=None, emoji=None, reply_parameters=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_to_message_id=None,
                 reply_markup=None, foursquare_type=None, google_place_id=None,
                 google_place_type=None, allow_sending_without_reply=None,
                 protect_content=None, message_thread_id=None, reply_parameters=None, *,
                 venue=None, read_timeout=None, write_timeout=None, connect_timeout=None,
                 pool_timeout=None, api_kwargs=None)
```

Alias for [send_venue\(\)](#)

```
async sendVideo(chat_id, video, duration=None, caption=None, disable_notification=None,
                  reply_to_message_id=None, reply_markup=None, width=None, height=None,
                  parse_mode=None, supports_streaming=None, allow_sending_without_reply=None,
                  caption_entities=None, protect_content=None, message_thread_id=None,
                  has_spoiler=None, thumbnail=None, reply_parameters=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_to_message_id=None, reply_markup=None,
                     allow_sending_without_reply=None, protect_content=None,
                     message_thread_id=None, thumbnail=None, reply_parameters=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_to_message_id=None, reply_markup=None, parse_mode=None,
                  allow_sending_without_reply=None, caption_entities=None, protect_content=None,
                  message_thread_id=None, reply_parameters=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_to_message_id=None, reply_markup=None,
                      allow_sending_without_reply=None, caption_entities=None,
                      protect_content=None, message_thread_id=None, has_spoiler=None,
                      thumbnail=None, reply_parameters=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.

Shortcuts

- `telegram.Chat.send_animation()`
 - `telegram.Message.reply_animation()`
 - `telegram.User.send_animation()`
-

See also:

[Working with Files and Media](#)

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` | `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`, optional) – Duration of sent animation in seconds.
- **`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.

New in version 13.10.

- **`message_thread_id`** (`int`, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.

New in version 20.0.

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

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`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`.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`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.

New 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.

New in version 20.2.

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

New in version 20.8.

Keyword Arguments

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

New in version 13.1.

- **read_timeout** (`float` | `None`, optional) – Value to pass to `telegram.request.BaseRequest.post.read_timeout`. Defaults to `DEFAULT_NONE`.
- **write_timeout** (`float` | `None`, optional) – Value to pass to `telegram.request.BaseRequest.post.write_timeout`. By default, 20 seconds are used as write timeout.

Deprecated since version 20.7: In future versions, the default value will be 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_to_message_id=None, reply_markup=None,
                 parse_mode=None, allow_sending_without_reply=None, caption_entities=None,
                 protect_content=None, message_thread_id=None, thumbnail=None,
                 reply_parameters=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.

Shortcuts

- `telegram.Chat.send_audio()`
 - `telegram.Message.reply_audio()`
 - `telegram.User.send_audio()`
-

See also:

Working with Files and Media

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 | `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`, optional) – Duration of sent audio in seconds.
- **performer** (`str`, optional) – Performer.
- **title** (`str`, optional) – Track name.
- **disable_notification** (`bool`, optional) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (`bool`, optional) – Protects the contents of the sent message from forwarding and saving.

New in version 13.10.

- **message_thread_id** (`int`, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.

New in version 20.0.

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

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`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`**.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`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.

New in version 20.2.

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

New in version 20.8.

Keyword Arguments

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

New in version 13.1.

- **`read_timeout`** (`float` | `None`, optional) – Value to pass to `telegram.request.BaseRequest.post.read_timeout`. Defaults to `DEFAULT_NONE`.
- **`write_timeout`** (`float` | `None`, optional) – Value to pass to `telegram.request.BaseRequest.post.write_timeout`. By default, 20 seconds are used as write timeout.

Deprecated since version 20.7: In future versions, the default value will be 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, *, *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.

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 the forum; for forum supergroups only.

New in version 20.0.

Keyword Arguments

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

Returns

On success, `True` is returned.

Return type

`bool`

Raises

`telegram.error.TelegramError` –

Shortcuts

- `telegram.Chat.send_action()`
- `telegram.Chat.send_chat_action()`
- `telegram.Message.reply_chat_action()`

- `telegram.User.send_action()`
 - `telegram.User.send_chat_action()`
-

async send_contact(*chat_id*, *phone_number=None*, *first_name=None*, *last_name=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *vcard=None*, *allow_sending_without_reply=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=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`.

Parameters

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

New in version 13.10.

- **`message_thread_id`** (`int`, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.

New in version 20.0.

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

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`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`.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **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.

New in version 20.8.

Keyword Arguments

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

Returns

On success, the sent Message is returned.

Return type

telegram.Message

Raises

telegram.error.TelegramError –

Shortcuts

- *telegram.Chat.send_contact()*
 - *telegram.Message.reply_contact()*
 - *telegram.User.send_contact()*
-

```
async send_dice(chat_id, disable_notification=None, reply_to_message_id=None,
                reply_markup=None, emoji=None, allow_sending_without_reply=None,
                protect_content=None, message_thread_id=None, reply_parameters=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.

Parameters

- **chat_id** (*int* | *str*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **disable_notification** (*bool*, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (*int*, optional) – If the message is a reply, ID of the original message. Mutually exclusive with *reply_parameters*.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`reply_markup`** (`InlineKeyboardMarkup` | `ReplyKeyboardMarkup` | `ReplyKeyboardRemove` | `ForceReply`, optional) – Additional interface options. An object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user
- **`emoji`** (`str`, optional) – Emoji on which the dice throw animation is based. Currently, must be one of `telegram.constants.DiceEmoji`. Dice can have values 1-6 for " ", " and ", values 1-5 for " and ", and values 1- 64 for ". Defaults to ".

Changed in version 13.4: Added the " emoji.

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

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

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

New in version 13.10.

- **`message_thread_id`** (`int`, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.

New in version 20.0.

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

New in version 20.8.

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, the sent Message is returned.

Return type

`telegram.Message`

Raises

`telegram.error.TelegramError` –

Shortcuts

- `telegram.Chat.send_dice()`
 - `telegram.Message.reply_dice()`
 - `telegram.User.send_dice()`
-

async send_document(*chat_id*, *document*, *caption=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *parse_mode=None*, *disable_content_type_detection=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *thumbnail=None*, *reply_parameters=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.

Shortcuts

- `telegram.Chat.send_document()`
 - `telegram.Message.reply_document()`
 - `telegram.User.send_document()`
-

See also:

[Working with Files and Media](#)

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` | `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.

New in version 13.10.

- **message_thread_id** (`int`, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.

New in version 20.0.

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

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **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`.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **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.

New in version 20.2.

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

New in version 20.8.

Keyword Arguments

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

- **read_timeout** (float | None, optional) – Value to pass to `telegram.request.BaseRequest.post.read_timeout`. Defaults to `DEFAULT_NONE`.
- **write_timeout** (float | None, optional) – Value to pass to `telegram.request.BaseRequest.post.write_timeout`. By default, 20 seconds are used as write timeout.

Deprecated since version 20.7: In future versions, the default value will be 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_to_message_id=None,
                reply_markup=None, allow_sending_without_reply=None, protect_content=None,
                message_thread_id=None, reply_parameters=None, *, read_timeout=None,
                write_timeout=None, connect_timeout=None, pool_timeout=None,
                api_kwargs=None)
```

Use this method to send a game.

Parameters

- **chat_id** (int) – Unique identifier for the target chat.
- **game_short_name** (str) – Short name of the game, serves as the unique identifier for the game. Set up your games via `@BotFather`.
- **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

New in version 13.10.

- **message_thread_id** (int, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.

New in version 20.0.

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

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **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`.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`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.

New in version 20.8.

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, the sent Message is returned.

Return type

`telegram.Message`

Raises

`telegram.error.TelegramError` –

Shortcuts

- `telegram.Chat.send_game()`
- `telegram.Message.reply_game()`
- `telegram.User.send_game()`

```
async send_invoice(chat_id, title, description, payload, provider_token, currency, prices,
                  start_parameter=None, photo_url=None, photo_size=None, photo_width=None,
                  photo_height=None, need_name=None, need_phone_number=None,
                  need_email=None, need_shipping_address=None, is_flexible=None,
                  disable_notification=None, reply_to_message_id=None, reply_markup=None,
                  provider_data=None, send_phone_number_to_provider=None,
                  send_email_to_provider=None, allow_sending_without_reply=None,
                  max_tip_amount=None, suggested_tip_amounts=None, protect_content=None,
                  message_thread_id=None, reply_parameters=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.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 for your internal processes.
- **`provider_token`** (`str`) – Payments provider token, obtained via `@BotFather`.
- **`currency`** (`str`) – Three-letter ISO 4217 currency code, see [more on currencies](#).
- **`prices`** (`Sequence[telegram.LabeledPrice]`) – Price breakdown, a sequence of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.).

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.

New 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`.

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

New in version 13.10.

- **message_thread_id** (`int`, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.

New in version 20.0.

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

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **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`.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **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.

New in version 20.8.

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, the sent Message is returned.

Return type

telegram.Message

Raises

telegram.error.TelegramError –

async send_location(*chat_id*, *latitude=None*, *longitude=None*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *live_period=None*, *horizontal_accuracy=None*, *heading=None*, *proximity_alert_radius=None*, *allow_sending_without_reply=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=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*.

Parameters

- **chat_id** (*int* | *str*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **latitude** (*float*, optional) – Latitude of location.
- **longitude** (*float*, optional) – Longitude of location.
- **horizontal_accuracy** (*int*, optional) – The radius of uncertainty for the location, measured in meters; 0-1500.
- **live_period** (*int*, optional) – Period in seconds for which the location will be updated, should be between 60 and 86400.
- **heading** (*int*, optional) – For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.

- **proximity_alert_radius** (`int`, optional) – For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters. Must be between `1` and `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.

New in version 13.10.

- **message_thread_id** (`int`, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.

New in version 20.0.

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

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **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`.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **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.

New in version 20.8.

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, the sent Message is returned.

Return type`telegram.Message`**Raises**`telegram.error.TelegramError` –

Shortcuts

- `telegram.Chat.send_location()`
 - `telegram.Message.reply_location()`
 - `telegram.User.send_location()`
-

async send_media_group(*chat_id*, *media*, *disable_notification=None*, *reply_to_message_id=None*, *allow_sending_without_reply=None*, *protect_content=None*, *message_thread_id=None*, *reply_parameters=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.

Shortcuts

- `telegram.Chat.send_media_group()`
 - `telegram.Message.reply_media_group()`
 - `telegram.User.send_media_group()`
-

See also:

[Working with Files and Media](#)

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`]) – 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.

New in version 13.10.

- **message_thread_id** (*int*, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.

New in version 20.0.

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

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **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*.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

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

New in version 20.8.

Keyword Arguments

- **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*.

New in version 20.0.

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

New 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*.

New 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.

Deprecated since version 20.7: In future versions, the default value will be 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 typeTuple[[telegram.Message](#)]**Raises**[telegram.error.TelegramError](#) –

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

Use this method to send text messages.

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](#). New in version 20.8.
- **disable_web_page_preview** ([bool](#), optional) – Disables link previews for links in this message. 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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **disable_notification** ([bool](#), optional) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** ([bool](#), optional) – Protects the contents of the sent message from forwarding and saving. New in version 13.10.

- **reply_to_message_id** ([int](#), optional) – If the message is a reply, ID of the original message. Mutually exclusive with [reply_parameters](#).

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **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](#).

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`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 the forum; for forum supergroups only.

New in version 20.0.

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

New in version 20.8.

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, 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.

Shortcuts

- `telegram.Chat.send_message()`
 - `telegram.Message.reply_html()`
 - `telegram.Message.reply_markdown_v2()`
 - `telegram.Message.reply_markdown()`
 - `telegram.Message.reply_text()`
 - `telegram.User.send_message()`
-

```

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

```

Use this method to send photos.

Shortcuts

- `telegram.Chat.send_photo()`
 - `telegram.Message.reply_photo()`
 - `telegram.User.send_photo()`
-

See also:

[Working with Files and Media](#)

Parameters

- **chat_id** (`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **photo** (`str` | `file object` | `bytes` | `pathlib.Path` | `telegram.PhotoSize`) – Photo to send. Pass a `file_id` as String to send a 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.

New in version 13.10.

- **`message_thread_id`** (`int`, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.

New in version 20.0.

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

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`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`.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`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.

New in version 20.0.

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

New in version 20.8.

Keyword Arguments

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

New in version 13.1.

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

- **`write_timeout`** (`float` | `None`, optional) – Value to pass to `telegram.request.BaseRequest.post.write_timeout`. By default, 20 seconds are used as write timeout.

Deprecated since version 20.7: In future versions, the default value will be 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_to_message_id=None, reply_markup=None,
               explanation=None, explanation_parse_mode=None, open_period=None,
               close_date=None, allow_sending_without_reply=None, explanation_entities=None,
               protect_content=None, message_thread_id=None, reply_parameters=None, *,
               read_timeout=None, write_timeout=None, connect_timeout=None,
               pool_timeout=None, api_kwargs=None)
```

Use this method to send a native 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]`) – Sequence of answer options, 2- 10 strings 1- 100 characters each.

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

- **is_anonymous** (`bool`, optional) – `True`, if the poll needs to be anonymous, defaults to `True`.
- **type** (`str`, optional) – Poll type, `'quiz'` or `'regular'`, defaults to `'regular'`.
- **allows_multiple_answers** (`bool`, optional) – `True`, if the poll allows multiple answers, ignored for polls in quiz mode, defaults to `False`.
- **correct_option_id** (`int`, optional) – 0-based identifier of the correct answer option, required for polls in quiz mode.
- **explanation** (`str`, optional) – Text that is shown when a user chooses an incorrect answer or taps on the lamp icon in a quiz-style poll, 0-200 characters with at most 2 line feeds after entities parsing.
- **explanation_parse_mode** (`str`, optional) – Mode for parsing entities in the explanation. See the constants in `telegram.constants.ParseMode` for the available modes.
- **explanation_entities** (`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`, optional) – Amount of time in seconds the poll will be active after creation, 5- 600. Can't be used together with `close_date`.
- **close_date** (`int` | `datetime.datetime`, optional) – Point in time (Unix timestamp) when the poll will be automatically closed. Must be at least 5 and no more than 600 seconds in the future. Can't be used together with `open_period`. For timezone naive

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

- **`is_closed`** (`bool`, optional) – Pass `True`, if the poll needs to be immediately closed. This can be useful for poll preview.
- **`disable_notification`** (`bool`, optional) – Sends the message silently. Users will receive a notification with no sound.
- **`protect_content`** (`bool`, optional) – Protects the contents of the sent message from forwarding and saving.

New in version 13.10.

- **`message_thread_id`** (`int`, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.

New in version 20.0.

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

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`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`.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`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.

New in version 20.8.

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, the sent Message is returned.

Return type`telegram.Message`**Raises**`telegram.error.TelegramError` –

Shortcuts

- `telegram.Chat.send_poll()`
 - `telegram.Message.reply_poll()`
 - `telegram.User.send_poll()`
-

async send_sticker(*chat_id*, *sticker*, *disable_notification=None*, *reply_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *protect_content=None*, *message_thread_id=None*, *emoji=None*, *reply_parameters=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.

Shortcuts

- `telegram.Chat.send_sticker()`
 - `telegram.Message.reply_sticker()`
 - `telegram.User.send_sticker()`
-

See also:

[Working with Files and Media](#)

Parameters

- **`chat_id`** (`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **`sticker`** (`str` | `file object` | `bytes` | `pathlib.Path` | `telegram.Sticker`) – Sticker to send. Pass a `file_id` as String to send a file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a 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`. 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

New 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.

New in version 13.10.

- **`message_thread_id`** (`int`, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.

New in version 20.0.

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

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`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`.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`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.

New in version 20.8.

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

Deprecated since version 20.7: In future versions, the default value will be 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_to_message_id=None,
                 reply_markup=None, foursquare_type=None, google_place_id=None,
                 google_place_type=None, allow_sending_without_reply=None,
                 protect_content=None, message_thread_id=None, reply_parameters=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.
-

Parameters

- **`chat_id`** (`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **`latitude`** (`float`, optional) – Latitude of venue.
- **`longitude`** (`float`, optional) – Longitude of venue.
- **`title`** (`str`, optional) – Name of the venue.
- **`address`** (`str`, optional) – Address of the venue.
- **`foursquare_id`** (`str`, optional) – Foursquare identifier of the venue.
- **`foursquare_type`** (`str`, optional) – Foursquare type of the venue, if known. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”.)
- **`google_place_id`** (`str`, optional) – Google Places identifier of the venue.
- **`google_place_type`** (`str`, optional) – Google Places type of the venue. (See supported types.)
- **`disable_notification`** (`bool`, optional) – Sends the message silently. Users will receive a notification with no sound.
- **`protect_content`** (`bool`, optional) – Protects the contents of the sent message from forwarding and saving.
New in version 13.10.
- **`message_thread_id`** (`int`, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.
New in version 20.0.
- **`reply_to_message_id`** (`int`, optional) – If the message is a reply, ID of the original message. Mutually exclusive with `reply_parameters`.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- `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`.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- `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.

New in version 20.8.

Keyword Arguments

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

Returns

On success, the sent Message is returned.

Return type

`telegram.Message`

Raises

`telegram.error.TelegramError` –

Shortcuts

- `telegram.Chat.send_venue()`
 - `telegram.Message.reply_venue()`
 - `telegram.User.send_venue()`
-

```
async send_video(chat_id, video, duration=None, caption=None, disable_notification=None,
                  reply_to_message_id=None, reply_markup=None, width=None, height=None,
                  parse_mode=None, supports_streaming=None,
                  allow_sending_without_reply=None, caption_entities=None,
                  protect_content=None, message_thread_id=None, has_spoiler=None,
                  thumbnail=None, reply_parameters=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.

Shortcuts

- `telegram.Chat.send_video()`
 - `telegram.Message.reply_video()`
 - `telegram.User.send_video()`
-

See also:

Working with Files and Media

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` | `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`, optional) – Duration of sent video in seconds.
- **width** (`int`, optional) – Video width.
- **height** (`int`, optional) – Video height.
- **caption** (`str`, optional) – Video caption (may also be used when resending videos by `file_id`), 0-1024 characters after entities parsing.
- **parse_mode** (`str`, optional) – 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.

New in version 13.10.

- **`message_thread_id`** (`int`, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.

New in version 20.0.

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

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`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`.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`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.

New 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.

New in version 20.2.

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

New in version 20.8.

Keyword Arguments

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

New in version 13.1.

- **read_timeout** (*float* | *None*, optional) – Value to pass to `telegram.request.BaseRequest.post.read_timeout`. Defaults to `DEFAULT_NONE`.
 - **write_timeout** (*float* | *None*, optional) – Value to pass to `telegram.request.BaseRequest.post.write_timeout`. By default, 20 seconds are used as write timeout.
- Deprecated since version 20.7: In future versions, the default value will be 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 def send_video_note(chat_id, video_note, duration=None, length=None,
                          disable_notification=None, reply_to_message_id=None,
                          reply_markup=None, allow_sending_without_reply=None,
                          protect_content=None, message_thread_id=None, thumbnail=None,
                          reply_parameters=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.

Shortcuts

- `telegram.Chat.send_video_note()`
 - `telegram.Message.reply_video_note()`
 - `telegram.User.send_video_note()`
-

See also:

[Working with Files and Media](#)

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` | `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`, optional) – Duration of sent video in seconds.
- **length** (`int`, optional) – Video width and height, i.e. diameter of the video message.
- **disable_notification** (`bool`, optional) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (`bool`, optional) – Protects the contents of the sent message from forwarding and saving.

New in version 13.10.

- **message_thread_id** (`int`, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.

New in version 20.0.

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

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **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`.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **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.

New in version 20.2.

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

New in version 20.8.

Keyword Arguments

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

New in version 13.1.

- **`read_timeout`** (`float` | `None`, optional) – Value to pass to `telegram.request.BaseRequest.post.read_timeout`. Defaults to `DEFAULT_NONE`.
- **`write_timeout`** (`float` | `None`, optional) – Value to pass to `telegram.request.BaseRequest.post.write_timeout`. By default, 20 seconds are used as write timeout.

Deprecated since version 20.7: In future versions, the default value will be 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_to_message_id=None, reply_markup=None, parse_mode=None,
                 allow_sending_without_reply=None, caption_entities=None,
                 protect_content=None, message_thread_id=None, reply_parameters=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 (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.

Shortcuts

- `telegram.Chat.send_voice()`

- `telegram.Message.reply_voice()`
 - `telegram.User.send_voice()`
-

See also:

Working with Files and Media

Parameters

- **`chat_id`** (`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **`voice`** (`str` | file object | `bytes` | `pathlib.Path` | `telegram.Voice`) – Voice file to send. Pass a `file_id` as String to send 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`, optional) – Duration of the voice message in seconds.
- **`disable_notification`** (`bool`, optional) – Sends the message silently. Users will receive a notification with no sound.
- **`protect_content`** (`bool`, optional) – Protects the contents of the sent message from forwarding and saving.

New in version 13.10.

- **`message_thread_id`** (`int`, optional) – Unique identifier for the target message thread (topic) of the forum; for forum supergroups only.

New in version 20.0.

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

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`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`.

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.

Deprecated since version 20.8: In future versions, this argument will become a keyword-only argument.

- **`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.

New in version 20.8.

Keyword Arguments

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

New in version 13.1.

- **`read_timeout`** (`float` | `None`, optional) – Value to pass to `telegram.request.BaseRequest.post.read_timeout`. Defaults to `DEFAULT_NONE`.
- **`write_timeout`** (`float` | `None`, optional) – Value to pass to `telegram.request.BaseRequest.post.write_timeout`. By default, 20 seconds are used as write timeout.

Deprecated since version 20.7: In future versions, the default value will be 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 setChatAdministratorCustomTitle(chat_id, user_id, custom_title, *, read_timeout=None,
                                       write_timeout=None, connect_timeout=None,
                                       pool_timeout=None, api_kwargs=None)
```

Alias for `set_chat_administrator_custom_title()`

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

Alias for `set_chat_description()`

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

Alias for `set_chat_menu_button()`

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

Alias for `set_webhook()`

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

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

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` –

Shortcuts

`telegram.Chat.set_administrator_custom_title()`

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.

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` –

Shortcuts

`telegram.Chat.set_description()`

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.

Shortcuts

- `telegram.Chat.set_menu_button()`
 - `telegram.User.set_menu_button()`
-

See also:

`get_chat_menu_button()`, `telegram.Chat.get_menu_button()` `telegram.User.get_menu_button()`

New in version 20.0.

Parameters

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

Keyword Arguments

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

Returns

On success, `True` is returned.

Return type

`bool`

`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.

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` –

Shortcuts

`telegram.Chat.set_permissions()`

`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.

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.

Deprecated since version 20.7: In future versions, the default value will be 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` –

Shortcuts

`telegram.Chat.set_photo()`

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

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

Parameters

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

Keyword Arguments

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

Returns

On success, `True` is returned.

Return type

`bool`

```
async def 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.

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` –

Shortcuts

`telegram.Chat.set_title()`

```
async def 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.

New 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 def 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.

Shortcuts

- `telegram.CallbackQuery.set_game_score()`
 - `telegram.Message.set_game_score()`
-

See also:

`telegram.Game.text`

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_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 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.

Shortcuts

- `telegram.Chat.set_message_reaction()`
 - `telegram.Message.set_reaction()`
-

New 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) – New 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.

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`.

New in version 13.7.

- **language_code** (str, optional) – A two-letter ISO 639-1 language code. If empty, commands will be applied to all users from the given scope, for whose language there are no dedicated commands.

New in version 13.7.

Keyword Arguments

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

New in version 20.0.

Parameters

- **rights** (`telegram.ChatAdministratorRights`, optional) – A `telegram.ChatAdministratorRights` object describing new default administrator rights. If not specified, the default administrator rights will be cleared.
- **for_channels** (`bool`, optional) – Pass `True` to change the default administrator rights of the bot in channels. Otherwise, the default administrator rights of the bot for groups and supergroups will be changed.

Keyword Arguments

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

Returns

Returns `True` on success.

Return type

`bool`

Raises

`telegram.error.TelegramError` –

```
async def 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.

New 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.

New 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_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.

New 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.

New in version 20.2.

Parameters

- **sticker** (str) – File identifier of the sticker.
- **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.

New in version 20.2.

Parameters

- **sticker** (str) – File identifier of the sticker.

- **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.

New in version 20.2.

Parameters

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

Keyword Arguments

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

New in version 20.2.

Parameters

- **name** (`str`) – Sticker set name
- **user_id** (`int`) – User identifier of created sticker set owner.
- **thumbnail** (`str` | `file object` | `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.

New 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_webhook(url, certificate=None, max_connections=None, allowed_updates=None,
                  ip_address=None, drop_pending_updates=None, secret_token=None, *,
                  read_timeout=None, write_timeout=None, connect_timeout=None,
                  pool_timeout=None, api_kwargs=None)
```

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

If you'd like to make sure that the Webhook was set by you, you can specify secret data in the parameter `secret_token`. If specified, the request will contain a header `X-Telegram-Bot-API-Secret-Token` with the secret token as content.

Note:

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](#).

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](#) (default). If not specified, the previous setting will be used. Please note that this parameter doesn't affect updates created before the call to the `set_webhook`, so unwanted updates may be received for a short period of time.

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.

New in version 20.0.

Keyword Arguments

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

New in version 20.0.

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

Alias for `stop_message_live_location()`

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

Alias for `stop_poll()`


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

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

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.

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`

Shortcuts

- `telegram.CallbackQuery.stop_message_live_location()`
 - `telegram.Message.stop_live_location()`
-

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

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

Parameters

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

Keyword Arguments

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

Returns

On success, the stopped Poll is returned.

Return type

`telegram.Poll`

Raises

`telegram.error.TelegramError` –

Shortcuts

`telegram.Message.stop_poll()`

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.

New in version 20.0.

Type

`str`

`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`.

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` –

Shortcuts

`telegram.Chat.unban_member()`

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()`
-

New 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()`

New 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, *, 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.

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` –

Shortcuts

- `telegram.Chat.unpin_all_messages()`
 - `telegram.User.unpin_all_messages()`
-

```
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. 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_forum_topic_messages()`
 - `telegram.Message.unpin_all_forum_topic_messages()`
-

New 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()`

New 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`, *, `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.

Parameters

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

Keyword Arguments

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

Returns

On success, `True` is returned.

Return type

`bool`

Raises

`telegram.error.TelegramError` –

Shortcuts

- `telegram.Chat.unpin_message()`
 - `telegram.Message.unpin()`
 - `telegram.User.unpin_message()`
-

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` | `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.

New 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`.

New 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.

Deprecated since version 20.7: In future versions, the default value will be 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`

Available Types**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.Message.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 sender.
- ***height*** (`int`) – Video height as defined by sender.
- ***duration*** (`int`) – Duration of the video in seconds as defined by sender.
- ***file_name*** (`str`, optional) – Original animation filename as defined by sender.
- ***mime_type*** (`str`, optional) – MIME type of the file as defined by sender.
- ***file_size*** (`int`, optional) – File size in bytes.
- ***thumbnail*** (*telegram.PhotoSize*, optional) – Animation thumbnail as defined by sender.

New 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 sender.

Type

`int`

height

Video height as defined by sender.

Type

`int`

duration

Duration of the video in seconds as defined by sender.

Type

`int`

file_name

Optional. Original animation filename as defined by sender.

Type

`str`

mime_type

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

Type

`str`

file_size

Optional. File size in bytes.

Type

`int`

thumbnail

Optional. Animation thumbnail as defined by sender.

New in version 20.2.

Type

`telegram.PhotoSize`

classmethod de_json(data, bot)

See `telegram.TelegramObject.de_json()`.

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

Convenience wrapper over `telegram.Bot.get_file()`

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

Returns

`telegram.File`

Raises

`telegram.error.TelegramError` –

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.ExternalReplyInfo.audio](#)
 - [telegram.Message.audio](#)
-

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*) – Duration of the audio in seconds as defined by sender.
- ***performer*** (*str*, optional) – Performer of the audio as defined by sender or by audio tags.
- ***title*** (*str*, optional) – Title of the audio as defined by sender or by audio tags.
- ***file_name*** (*str*, optional) – Original filename as defined by sender.
- ***mime_type*** (*str*, optional) – MIME type of the file as defined by sender.
- ***file_size*** (*int*, optional) – File size in bytes.
- ***thumbnail*** ([telegram.PhotoSize](#), optional) – Thumbnail of the album cover to which the music file belongs.

New 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 sender.

Type

`int`

performer

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

Type

`str`

title

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

Type

`str`

file_name

Optional. Original filename as defined by sender.

Type

`str`

mime_type

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

Type

`str`

file_size

Optional. File size in bytes.

Type

`int`

thumbnail

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

New in version 20.2.

Type

`telegram.PhotoSize`

classmethod `de_json(data, bot)`

See `telegram.TelegramObject.de_json()`.

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

Convenience wrapper over `telegram.Bot.get_file()`

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

Returns

`telegram.File`

Raises

`telegram.error.TelegramError` –

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.

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

Use In

telegram.Bot.set_my_commands()

MAX_COMMAND = 32

telegram.constants.BotCommandLimit.MAX_COMMAND

New in version 20.0.

MAX_DESCRIPTION = 256

telegram.constants.BotCommandLimit.MAX_DESCRIPTION

New in version 20.0.

MIN_COMMAND = 1

telegram.constants.BotCommandLimit.MIN_COMMAND

New in version 20.0.

MIN_DESCRIPTION = 1

telegram.constants.BotCommandLimit.MIN_DESCRIPTION

New 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()*
-

New in version 13.7.

Parameters

type (*str*) – Scope type.

type

Scope type.

Type

str

ALL_CHAT_ADMINISTRATORS = 'all_chat_administrators'

telegram.constants.BotCommandScopeType.ALL_CHAT_ADMINISTRATORS

ALL_GROUP_CHATS = 'all_group_chats'

telegram.constants.BotCommandScopeType.ALL_GROUP_CHATS

ALL_PRIVATE_CHATS = 'all_private_chats'

telegram.constants.BotCommandScopeType.ALL_PRIVATE_CHATS

CHAT = 'chat'

telegram.constants.BotCommandScopeType.CHAT

CHAT_ADMINISTRATORS = 'chat_administrators'

telegram.constants.BotCommandScopeType.CHAT_ADMINISTRATORS

CHAT_MEMBER = 'chat_member'

telegram.constants.BotCommandScopeType.CHAT_MEMBER

DEFAULT = 'default'

telegram.constants.BotCommandScopeType.DEFAULT

classmethod `de_json(data, bot)`

Converts JSON data to the appropriate *BotCommandScope* object, i.e. takes care of selecting the correct subclass.

Parameters

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

Returns

The Telegram object.

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()*
-

New 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()*
-

New 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\(\)](#)
-

New 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\(\)](#)
-

New in version 13.7.

Parameters

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

type

Scope type '[chat](#)'.

Type

[str](#)

chat_id

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

Type

[str](#) | [int](#)

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\(\)](#)
-

New in version 13.7.

Parameters

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

type

Scope type '[chat_administrators](#)'.

Type

[str](#)

chat_id

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

Type

[str](#) | [int](#)

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\(\)](#)
-

New in version 13.7.

Parameters

- [chat_id](#) ([str](#) | [int](#)) – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername).
- [user_id](#) ([int](#)) – Unique identifier of the target user.

type

Scope type `'chat_member'`.

Type

`str`

chat_id

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

Type

`str | int`

user_id

Unique identifier of the target user.

Type

`int`

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\(\)](#)
-

New 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\(\)](#)

New 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()

New 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()

New in version 20.2.

Parameters

short_description (*str*) – The bot’s short description.

short_description

The bot’s short description.

Type

str

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.

New in version 13.6.

Parameters

- [id](#) ([str](#)) – Unique identifier for this query.
- [from_user](#) ([telegram.User](#)) – Sender.
- [chat_instance](#) ([str](#)) – Global identifier, uniquely corresponding to the chat to which the message with the callback button was sent. Useful for high scores in games.
- [message](#) ([telegram.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`

Available In

`telegram.Update.callback_query`

MAX_ANSWER_TEXT_LENGTH = 200

`telegram.constants.CallbackQueryLimit.ANSWER_CALLBACK_QUERY_TEXT_LENGTH`

New in version 13.2.

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

Shortcut for:

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

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

Returns

On success, `True` is returned.

Return type

`bool`

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

Shortcut for:

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

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

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)`

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

Shortcut for either:

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

or:

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

For the documentation of the arguments, please see [telegram.Bot.edit_message_live_location\(\)](#) and [telegram.Message.edit_live_location\(\)](#).

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, disable_web_page_preview=None,
                        reply_markup=None, entities=None, link_preview_options=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,
```

(continues on next page)

(continued from previous page)

```

    ↪ **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,
                    photo=None, description=None, invite_link=None, pinned_message=None,
                    permissions=None, sticker_set_name=None, can_set_sticker_set=None,
                    slow_mode_delay=None, bio=None, linked_chat_id=None, location=None,
                    message_auto_delete_time=None, has_private_forwards=None,
                    has_protected_content=None, join_to_send_messages=None, join_by_request=None,
                    has_restricted_voice_and_video_messages=None, is_forum=None,
                    active_usernames=None, emoji_status_custom_emoji_id=None,
                    emoji_status_expiration_date=None, has_aggressive_anti_spam_enabled=None,
                    has_hidden_members=None, available_reactions=None, accent_color_id=None,
                    background_custom_emoji_id=None, profile_accent_color_id=None,
                    profile_background_custom_emoji_id=None, has_visible_history=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.ChatBoostRemoved.chat`
- `telegram.ChatBoostUpdated.chat`
- `telegram.ChatJoinRequest.chat`
- `telegram.ChatMemberUpdated.chat`
- `telegram.ExternalReplyInfo.chat`
- `telegram.Giveaway.chats`
- `telegram.GiveawayWinners.chat`
- `telegram.InaccessibleMessage.chat`
- `telegram.MaybeInaccessibleMessage.chat`
- `telegram.Message.chat`
- `telegram.Message.forward_from_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.Update.effective_chat`
-

Returned In

`telegram.Bot.get_chat()`

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`.

Parameters

- **`id`** (`int`) – Unique identifier for this chat. This number may be greater than 32 bits and some programming languages may have difficulty/silent defects in interpreting it. But it is smaller than 52 bits, so a signed 64-bit integer or double-precision float type are safe for storing this identifier.
- **`type`** (`str`) – Type of chat, can be either `PRIVATE`, `GROUP`, `SUPERGROUP` or `CHANNEL`.
- **`title`** (`str`, optional) – Title, for supergroups, channels and group chats.
- **`username`** (`str`, optional) – Username, for private chats, supergroups and channels if available.
- **`first_name`** (`str`, optional) – First name of the other party in a private chat.
- **`last_name`** (`str`, optional) – Last name of the other party in a private chat.
- **`photo`** (`telegram.ChatPhoto`, optional) – Chat photo. Returned only in `telegram.Bot.get_chat()`.
- **`bio`** (`str`, optional) – Bio of the other party in a private chat. Returned only in `telegram.Bot.get_chat()`.
- **`has_private_forwards`** (`bool`, optional) – `True`, if privacy settings of the other party in the private chat allows to use `tg://user?id=<user_id>` links only in chats with the user. Returned only in `telegram.Bot.get_chat()`.

New in version 13.9.

- **`description`** (`str`, optional) – Description, for groups, supergroups and channel chats. Returned only in `telegram.Bot.get_chat()`.
- **`invite_link`** (`str`, optional) – Primary invite link, for groups, supergroups and channel. Returned only in `telegram.Bot.get_chat()`.
- **`pinned_message`** (`telegram.Message`, optional) – The most recent pinned message (by sending date). Returned only in `telegram.Bot.get_chat()`.
- **`permissions`** (`telegram.ChatPermissions`) – Optional. Default chat member permissions, for groups and supergroups. Returned only in `telegram.Bot.get_chat()`.
- **`slow_mode_delay`** (`int`, optional) – For supergroups, the minimum allowed delay between consecutive messages sent by each unprivileged user. Returned only in `telegram.Bot.get_chat()`.

- **`message_auto_delete_time`** (`int`, optional) – The time after which all messages sent to the chat will be automatically deleted; in seconds. Returned only in `telegram.Bot.get_chat()`.

New in version 13.4.

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

New in version 13.9.

- **`has_visible_history`** (`bool`, optional) – `True`, if new chat members will have access to old messages; available only to chat administrators. Returned only in `telegram.Bot.get_chat()`.

New in version 20.8.

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

New in version 20.0.

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

New in version 20.0.

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

New in version 20.0.

- **`is_forum`** (`bool`, optional) – `True`, if the supergroup chat is a forum (has `topics` enabled).

New in version 20.0.

- **`active_usernames`** (`Sequence[str]`, optional) – If set, the list of all active chat usernames; for private chats, supergroups and channels. Returned only in `telegram.Bot.get_chat()`.

New in version 20.0.

- **`available_reactions`** (`Sequence[telegram.ReactionType]`, optional) – List of available reactions allowed in the chat. If omitted, then all of `telegram.constants.ReactionEmoji` are allowed. Returned only in `telegram.Bot.get_chat()`.

New in version 20.8.

- **`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. Returned only in `telegram.Bot.get_chat()`. Always returned in `telegram.Bot.get_chat()`.

New 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. Returned only in `telegram.Bot.get_chat()`.

New 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. Returned only in `telegram.Bot.get_chat()`.

New 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. Returned only in `telegram.Bot.get_chat()`.

New 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. Returned only in `telegram.Bot.get_chat()`.

New 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, in seconds. Returned only in `telegram.Bot.get_chat()`. The default timezone of the bot is used for localization, which is UTC unless `telegram.ext.Defaults.tzinfo` is used.

New in version 20.5.

- **`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. Returned only in `telegram.Bot.get_chat()`.

New 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. Returned only in `telegram.Bot.get_chat()`.

New in version 20.0.

id

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

Type

`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`

photo

Optional. Chat photo. Returned only in `telegram.Bot.get_chat()`.

Type

`telegram.ChatPhoto`

bio

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

Type

`str`

has_private_forwards

Optional. `True`, if privacy settings of the other party in the private chat allows to use `tg://user?id=<user_id>` links only in chats with the user. Returned only in `telegram.Bot.get_chat()`.

New in version 13.9.

Type

`bool`

description

Optional. Description, for groups, supergroups and channel chats. Returned only in `telegram.Bot.get_chat()`.

Type

`str`

invite_link

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

Type

`str`

pinned_message

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

Type

`telegram.Message`

permissions

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

Type

`telegram.ChatPermissions`

slow_mode_delay

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

Type`int`**message_auto_delete_time**

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

New in version 13.4.

Type`int`**has_protected_content**

Optional. `True`, if messages from the chat can't be forwarded to other chats. Returned only in `telegram.Bot.get_chat()`.

New 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. Returned only in `telegram.Bot.get_chat()`.

New in version 20.8.

Type`bool`**sticker_set_name**

Optional. For supergroups, name of Group sticker set. Returned only in `telegram.Bot.get_chat()`.

Type`str`**can_set_sticker_set**

Optional. `True`, if the bot can change group the sticker set. Returned only in `telegram.Bot.get_chat()`.

Type`bool`**linked_chat_id**

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

Type`int`**location**

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

Type`telegram.ChatLocation`**join_to_send_messages**

Optional. `True`, if users need to join the supergroup before they can send messages. Returned only in `telegram.Bot.get_chat()`.

New in version 20.0.

Type`bool`

join_by_request

Optional. **True**, if all users directly joining the supergroup need to be approved by supergroup administrators. Returned only in `telegram.Bot.get_chat()`.

New in version 20.0.

Type

`bool`

has_restricted_voice_and_video_messages

Optional. **True**, if the privacy settings of the other party restrict sending voice and video note messages in the private chat. Returned only in `telegram.Bot.get_chat()`.

New in version 20.0.

Type

`bool`

is_forum

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

New in version 20.0.

Type

`bool`

active_usernames

Optional. If set, the list of all `active chat usernames`; for private chats, supergroups and channels. Returned only in `telegram.Bot.get_chat()`. This list is empty if the chat has no active usernames or this chat instance was not obtained via `get_chat()`.

New in version 20.0.

Type

`Tuple[str]`

available_reactions

Optional. List of available reactions allowed in the chat. If omitted, then all of `telegram.constants.ReactionEmoji` are allowed. Returned only in `telegram.Bot.get_chat()`.

New in version 20.8.

Type

`Tuple[telegram.ReactionType]`

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. Returned only in `telegram.Bot.get_chat()`. Always returned in `telegram.Bot.get_chat()`.

New in version 20.8.

Type

`int`

background_custom_emoji_id

Optional. Custom emoji identifier of emoji chosen by the chat for the reply header and link preview background. Returned only in `telegram.Bot.get_chat()`.

New 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. Returned only in *telegram.Bot.get_chat()*.

New 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. Returned only in *telegram.Bot.get_chat()*.

New 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. Returned only in *telegram.Bot.get_chat()*.

New 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, in seconds. Returned only in *telegram.Bot.get_chat()*. The default timezone of the bot is used for localization, which is UTC unless *telegram.ext.Defaults.tzinfo* is used.

New in version 20.5.

Type

datetime.datetime

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. Returned only in *telegram.Bot.get_chat()*.

New 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. Returned only in *telegram.Bot.get_chat()*.

New in version 20.0.

Type

bool

CHANNEL = 'channel'

telegram.constants.ChatType.CHANNEL

GROUP = 'group'

telegram.constants.ChatType.GROUP

PRIVATE = 'private'

telegram.constants.ChatType.PRIVATE

SENDER = 'sender'

[telegram.constants.ChatType.SENDER](#)

New in version 13.5.

SUPERGROUP = 'supergroup'

[telegram.constants.ChatType.SUPERGROUP](#)

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

Shortcut for:

```
await bot.approve_chat_join_request(chat_id=update.effective_chat.id, *args, kwargs)
```

For the documentation of the arguments, please see [telegram.Bot.approve_chat_join_request\(\)](#).

New in version 13.8.

Returns

On success, **True** is returned.

Return type

bool

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

Shortcut for:

```
await bot.ban_chat_sender_chat(  
    sender_chat_id=update.effective_chat.id, *args, kwargs  
)
```

For the documentation of the arguments, please see [telegram.Bot.ban_chat_sender_chat\(\)](#).

New in version 13.9.

Returns

On success, **True** is returned.

Return type

bool

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

Shortcut for:

```
await bot.ban_chat_member(update.effective_chat.id, *args, kwargs)
```

For the documentation of the arguments, please see [telegram.Bot.ban_chat_member\(\)](#).

Returns

On success, **True** is returned.

Return type

bool

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

Shortcut for:

```
await bot.ban_chat_sender_chat(chat_id=update.effective_chat.id, *args,  
↪ **kwargs)
```

For the documentation of the arguments, please see [`telegram.Bot.ban_chat_sender_chat\(\)`](#).

New in version 13.9.

Returns

On success, `True` is returned.

Return type

`bool`

```
async 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\(\)`](#).

New 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\(\)`](#).

New 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_to_message_id=None,  
                  allow_sending_without_reply=None, reply_markup=None,  
                  protect_content=None, message_thread_id=None, reply_parameters=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, instance representing the message posted.

Return type

`telegram.Message`

```
async copy_messages(chat_id, message_ids, disable_notification=None, protect_content=None,
                    message_thread_id=None, remove_caption=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()`.

New 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()`.

New 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()`.

New in version 13.4.

Changed in version 13.8: Edited signature according to the changes of `telegram.Bot.create_chat_invite_link()`.

Returns

`telegram.ChatInviteLink`

```
classmethod de_json(data, bot)
```

See `telegram.TelegramObject.de_json()`.

```
async def decline_join_request(user_id, *, read_timeout=None, write_timeout=None,
                              connect_timeout=None, pool_timeout=None, api_kwargs=None)
```

Shortcut for:

```
await bot.decline_chat_join_request(chat_id=update.effective_chat.id, *args,
→ **kwargs)
```

For the documentation of the arguments, please see [telegram.Bot.decline_chat_join_request\(\)](#).

New in version 13.8.

Returns

On success, `True` is returned.

Return type

`bool`

```
async def 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\(\)](#).

New in version 20.0.

Returns

On success, `True` is returned.

Return type

`bool`

```
async def 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\(\)](#).

New in version 20.8.

Returns

On success, `True` is returned.

Return type

`bool`

```
async def 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\(\)](#).

New 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\(\)`](#).

New in version 20.0.

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\(\)`](#).

New 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\(\)`](#).

New in version 20.0.

Returns

On success, `True` is returned.

Return type`bool`

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

Shortcut for:

```
await bot.edit_chat_invite_link(chat_id=update.effective_chat.id, *args,
↪ **kwargs)
```

For the documentation of the arguments, please see [telegram.Bot.edit_chat_invite_link\(\)](#).

New in version 13.4.

Changed in version 13.8: Edited signature according to the changes of [telegram.Bot.edit_chat_invite_link\(\)](#).

Returns

[telegram.ChatInviteLink](#)

property effective_name

Convenience property. Gives [title](#) if not [None](#), else [full_name](#) if not [None](#).

New 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\(\)](#).

New in version 13.4.

Returns

New invite link on success.

Return type

[str](#)

async forward_from(*from_chat_id*, *message_id*, *disable_notification=None*, *protect_content=None*, *message_thread_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\(\)](#)

New 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*, *, *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()`.

New 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, *,
                           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()`.

New 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, *, 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()`

New in version 20.0.

Returns

On success, instance representing the message posted.

Return type

`telegram.Message`

property full_name

Convenience property. If `first_name` is not `None`, gives `first_name` followed by (if available) `last_name`.

Note: `full_name` will always be `None`, if the chat is a (super)group or channel.

New in version 13.2.

Type`str`

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

Shortcut for:

```
await bot.get_chat_administrators(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.get_chat_administrators()`.

Returns

A 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_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\(\)`

New 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()`.

New 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()`.

New 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*)

New 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.

New 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`)

New 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`, *, `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, *,
                       read_timeout=None, write_timeout=None, connect_timeout=None,
                       pool_timeout=None, api_kwargs=None)
```

Shortcut for:

```
await bot.promote_chat_member(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see [telegram.Bot.promote_chat_member\(\)](#).

New in version 13.2.

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

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 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\(\)](#).

New 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\(\)](#).

New in version 20.0.

Returns

On success, `True` is returned.

Return type

`bool`

```
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()`.

New in version 13.2.

New 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()`.

New in version 13.4.

Returns

`telegram.ChatInviteLink`

```
async send_action(action, message_thread_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_to_message_id=None,
                    reply_markup=None, allow_sending_without_reply=None,
                    caption_entities=None, protect_content=None, message_thread_id=None,
                    has_spoiler=None, thumbnail=None, reply_parameters=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_to_message_id=None, reply_markup=None,
                  parse_mode=None, allow_sending_without_reply=None, caption_entities=None,
                  protect_content=None, message_thread_id=None, thumbnail=None,
                  reply_parameters=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, *, read_timeout=None,
                        write_timeout=None, connect_timeout=None, pool_timeout=None,
                        api_kwargs=None)
```

Shortcut for:

```
await bot.send_chat_action(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see [telegram.Bot.send_chat_action\(\)](#).

Returns

On success, `True` is returned.

Return type

`bool`

```
async send_contact(phone_number=None, first_name=None, last_name=None,
                    disable_notification=None, reply_to_message_id=None, reply_markup=None,
                    vcard=None, allow_sending_without_reply=None, protect_content=None,
                    message_thread_id=None, reply_parameters=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, *, 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\(\)`](#).

New 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_to_message_id=None,
                allow_sending_without_reply=None, reply_markup=None, protect_content=None,
                message_thread_id=None, reply_parameters=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_to_message_id=None, reply_markup=None,
                emoji=None, allow_sending_without_reply=None, protect_content=None,
                message_thread_id=None, reply_parameters=None, *, read_timeout=None,
                write_timeout=None, connect_timeout=None, pool_timeout=None,
                api_kwargs=None)
```

Shortcut for:

```
await bot.send_dice(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see [`telegram.Bot.send_dice\(\)`](#).

Returns

On success, instance representing the message posted.

Return type

[`telegram.Message`](#)

```
async send_document(document, caption=None, disable_notification=None,
                    reply_to_message_id=None, reply_markup=None, parse_mode=None,
                    disable_content_type_detection=None, allow_sending_without_reply=None,
                    caption_entities=None, protect_content=None, message_thread_id=None,
                    thumbnail=None, reply_parameters=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_to_message_id=None,
                 reply_markup=None, allow_sending_without_reply=None, protect_content=None,
                 message_thread_id=None, reply_parameters=None, *, read_timeout=None,
                 write_timeout=None, connect_timeout=None, pool_timeout=None,
                 api_kwargs=None)
```

Shortcut for:

```
await bot.send_game(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see [`telegram.Bot.send_game\(\)`](#).

Returns

On success, instance representing the message posted.

Return type

[`telegram.Message`](#)

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

Shortcut for:

```
await bot.send_invoice(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see [`telegram.Bot.send_invoice\(\)`](#).

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

Changed in version 13.5: As of Bot API 5.2, the parameter [`start_parameter`](#) is optional.

Returns

On success, instance representing the message posted.

Return type

[`telegram.Message`](#)

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

Shortcut for:

```
await bot.send_location(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see [telegram.Bot.send_location\(\)](#).

Returns

On success, instance representing the message posted.

Return type

[telegram.Message](#)

```
async send_media_group(media, disable_notification=None, reply_to_message_id=None,
                        allow_sending_without_reply=None, protect_content=None,
                        message_thread_id=None, reply_parameters=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_web_page_preview=None,
                   disable_notification=None, reply_to_message_id=None, reply_markup=None,
                   allow_sending_without_reply=None, entities=None, protect_content=None,
                   message_thread_id=None, link_preview_options=None, reply_parameters=None,
                   *, read_timeout=None, write_timeout=None, connect_timeout=None,
                   pool_timeout=None, api_kwargs=None)
```

Shortcut for:

```
await bot.send_message(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see [telegram.Bot.send_message\(\)](#).

Returns

On success, instance representing the message posted.

Return type

[telegram.Message](#)

```
async send_photo(photo, caption=None, disable_notification=None, reply_to_message_id=None,
                  reply_markup=None, parse_mode=None, allow_sending_without_reply=None,
                  caption_entities=None, protect_content=None, message_thread_id=None,
                  has_spoiler=None, reply_parameters=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_to_message_id=None, reply_markup=None, explanation=None,
                 explanation_parse_mode=None, open_period=None, close_date=None,
                 allow_sending_without_reply=None, explanation_entities=None,
                 protect_content=None, message_thread_id=None, reply_parameters=None, *,
                 read_timeout=None, write_timeout=None, connect_timeout=None,
                 pool_timeout=None, api_kwargs=None)
```

Shortcut for:

```
await bot.send_poll(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see [telegram.Bot.send_poll\(\)](#).

Returns

On success, instance representing the message posted.

Return type

[telegram.Message](#)

```
async send_sticker(sticker, disable_notification=None, reply_to_message_id=None,
                   reply_markup=None, allow_sending_without_reply=None,
                   protect_content=None, message_thread_id=None, emoji=None,
                   reply_parameters=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_to_message_id=None, reply_markup=None,
                 foursquare_type=None, google_place_id=None, google_place_type=None,
                 allow_sending_without_reply=None, protect_content=None,
                 message_thread_id=None, reply_parameters=None, *, venue=None,
                 read_timeout=None, write_timeout=None, connect_timeout=None,
                 pool_timeout=None, api_kwargs=None)
```

Shortcut for:

```
await bot.send_venue(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see [telegram.Bot.send_venue\(\)](#).

Returns

On success, instance representing the message posted.

Return type

[telegram.Message](#)

```
async send_video(video, duration=None, caption=None, disable_notification=None,
                 reply_to_message_id=None, reply_markup=None, width=None, height=None,
                 parse_mode=None, supports_streaming=None,
                 allow_sending_without_reply=None, caption_entities=None,
                 protect_content=None, message_thread_id=None, has_spoiler=None,
                 thumbnail=None, reply_parameters=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_to_message_id=None, reply_markup=None,
                      allow_sending_without_reply=None, protect_content=None,
                      message_thread_id=None, thumbnail=None, reply_parameters=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_to_message_id=None, reply_markup=None, parse_mode=None,
                 allow_sending_without_reply=None, caption_entities=None,
                 protect_content=None, message_thread_id=None, reply_parameters=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_description(description=None, *, read_timeout=None, write_timeout=None,
                      connect_timeout=None, pool_timeout=None, api_kwargs=None)
```

Shortcut for:

```
await bot.set_chat_description(
    chat_id=update.effective_chat.id, *args, **kwargs
)
```

For the documentation of the arguments, please see [telegram.Bot.set_chat_description\(\)](#).

New in version 20.0.

Returns

On success, `True` is returned.

Return type

`bool`

```
async set_menu_button(menu_button=None, *, read_timeout=None, write_timeout=None,
                      connect_timeout=None, pool_timeout=None, api_kwargs=None)
```

Shortcut for:

```
await bot.set_chat_menu_button(chat_id=update.effective_chat.id, *args,
                               ↪ **kwargs)
```

For the documentation of the arguments, please see [telegram.Bot.set_chat_menu_button\(\)](#).

Caution: Can only work, if the chat is a private chat.

See also:

[get_menu_button\(\)](#)

New in version 20.0.

Returns

On success, `True` is returned.

Return type

`bool`

```
async set_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\(\)`](#).

New 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\(\)`](#).

New 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\(\)`](#).

New in version 20.0.

Returns

On success, `True` is returned.

Return type

`bool`

```
async set_title(title, *, read_timeout=None, write_timeout=None, connect_timeout=None,
                pool_timeout=None, api_kwargs=None)
```

Shortcut for:

```
await bot.set_chat_title(
    chat_id=update.effective_chat.id, *args, **kwargs
)
```

For the documentation of the arguments, please see [`telegram.Bot.set_chat_title\(\)`](#).

New in version 20.0.

Returns

On success, `True` is returned.

Return type

`bool`

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

Shortcut for:

```
await bot.unban_chat_sender_chat(
    sender_chat_id=update.effective_chat.id, *args, **kwargs
)
```

For the documentation of the arguments, please see [`telegram.Bot.unban_chat_sender_chat\(\)`](#).

New in version 13.9.

Returns

On success, `True` is returned.

Return type

`bool`

```
async unban_member(user_id, only_if_banned=None, *, read_timeout=None, write_timeout=None,
    connect_timeout=None, pool_timeout=None, api_kwargs=None)
```

Shortcut for:

```
await bot.unban_chat_member(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see [`telegram.Bot.unban_chat_member\(\)`](#).

Returns

On success, `True` is returned.

Return type

`bool`

```
async unban_sender_chat(sender_chat_id, *, read_timeout=None, write_timeout=None,
    connect_timeout=None, pool_timeout=None, api_kwargs=None)
```

Shortcut for:

```
await bot.unban_chat_sender_chat(chat_id=update.effective_chat.id, *args,
    ↪ **kwargs)
```

For the documentation of the arguments, please see [`telegram.Bot.unban_chat_sender_chat\(\)`](#).

New in version 13.9.

Returns

On success, `True` is returned.

Return type

`bool`

```
async 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\(\)`](#).

New 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\(\)](#).

New 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\(\)](#).

New 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, *, 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`

ChatAdministratorRights

New 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_messages=None, can_edit_messages=None,
                                       can_pin_messages=None, can_manage_topics=None,
                                       can_post_stories=None, can_edit_stories=None,
                                       can_delete_stories=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`, and `can_edit_stories` are equal.

Use In

`telegram.Bot.set_my_default_administrator_rights()`

Returned In

`telegram.Bot.get_my_default_administrator_rights()`

New 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.

Parameters

- `is_anonymous` (`bool`) – `True`, if the user's presence in the chat is hidden.
- `can_manage_chat` (`bool`) – `True`, if the administrator can access the chat event log, chat statistics, boost list in channels, see channel members, report spam messages, see anonymous administrators in supergroups and ignore slow mode. Implied by any other administrator privilege.
- `can_delete_messages` (`bool`) – `True`, if the administrator can delete messages of other users.
- `can_manage_video_chats` (`bool`) – `True`, if the administrator can manage video chats.
- `can_restrict_members` (`bool`) – `True`, if the administrator can restrict, ban or unban chat members, 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; channels only.
- **`can_edit_messages`** (`bool`, optional) – `True`, if the administrator can edit messages of other users.
- **`can_pin_messages`** (`bool`, optional) – `True`, if the user is allowed to pin messages; groups and supergroups only.
- **`can_post_stories`** (`bool`, optional) – `True`, if the administrator can post stories in the channel; channels only.

New in version 20.6.

- **`can_edit_stories`** (`bool`, optional) – `True`, if the administrator can edit stories posted by other users; channels only.

New in version 20.6.

- **`can_delete_stories`** (`bool`, optional) – `True`, if the administrator can delete stories posted by other users; channels only.

New in version 20.6.

- **`can_manage_topics`** (`bool`, optional) – `True`, if the user is allowed to create, rename, close, and reopen forum topics; supergroups only.

New in version 20.0.

`is_anonymous`

`True`, if the user’s presence in the chat is hidden.

Type

`bool`

`can_manage_chat`

`True`, if the administrator can access the chat event log, chat statistics, boost list in channels, see channel members, report spam messages, see anonymous administrators in supergroups and ignore slow mode. Implied by any other administrator privilege.

Type

`bool`

`can_delete_messages`

`True`, if the administrator can delete messages of other users.

Type

`bool`

`can_manage_video_chats`

`True`, if the administrator can manage video chats.

Type

`bool`

`can_restrict_members`

`True`, if the administrator can restrict, ban or unban chat members, 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; channels only.

Type
`bool`

can_edit_messages

Optional. `True`, if the administrator can edit messages of other users.

Type
`bool`

can_pin_messages

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

Type
`bool`

can_post_stories

Optional. `True`, if the administrator can post stories in the channel; channels only.

New in version 20.6.

Type
`bool`

can_edit_stories

Optional. `True`, if the administrator can edit stories posted by other users; channels only.

New in version 20.6.

Type
`bool`

can_delete_stories

Optional. `True`, if the administrator can delete stories posted by other users; channels only.

New in version 20.6.

Type
`bool`

can_manage_topics

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

New in version 20.0.

Type

`bool`

classmethod all_rights()

This method returns the `ChatAdministratorRights` object with all attributes set to `True`. This is e.g. useful when changing the bot's default administrator rights with `telegram.Bot.set_my_default_administrator_rights()`.

New in version 20.0.

classmethod no_rights()

This method returns the `ChatAdministratorRights` object with all attributes set to `False`.

New in version 20.0.

ChatBoost

New 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`
-

New 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)

See `telegram.TelegramObject.de_json()`.

ChatBoostRemoved

New 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.

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`

Available In

`telegram.Update.removed_chat_boost`

classmethod `de_json(data, bot)`

See `telegram.TelegramObject.de_json()`.

ChatBoostSource

New 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`
-

New 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)`

See `telegram.TelegramObject.de_json()`.

ChatBoostSourceGiftCode

New 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](#)
-

New 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

New in version 20.8.

class telegram.ChatBoostSourceGiveaway(*giveaway_message_id*, *user=None*, *is_unclaimed=None*, *, *api_kwargs=None*)

Bases: [telegram.ChatBoostSource](#)

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

Available In

- [telegram.ChatBoost.source](#)
 - [telegram.ChatBoostRemoved.source](#)
-

New 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.
- **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

is_unclaimed

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

Type

bool

ChatBoostSourcePremium

New 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*
-

New 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

New 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](#)

New 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)`

See [telegram.TelegramObject.de_json\(\)](#).

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, *, 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.revoke_chat_invite_link\(\)](#)
-

Available In

- [telegram.ChatJoinRequest.invite_link](#)
- [telegram.ChatMemberUpdated.invite_link](#)

Returned In

- `telegram.Bot.create_chat_invite_link()`
 - `telegram.Bot.edit_chat_invite_link()`
 - `telegram.Bot.revoke_chat_invite_link()`
-

New in version 13.4.

Changed in version 20.0:

- The argument & attribute `creates_join_request` is now required to comply with the Bot API.
- Comparing objects of this class now also takes `creates_join_request` into account.

Parameters

- `invite_link` (`str`) – The invite link.
- `creator` (`telegram.User`) – Creator of the link.
- `creates_join_request` (`bool`) – `True`, if users joining the chat via the link need to be approved by chat administrators.

New in version 13.8.

- `is_primary` (`bool`) – `True`, if the link is primary.
- `is_revoked` (`bool`) – `True`, if the link is revoked.
- `expire_date` (`datetime.datetime`, optional) – Date when the link will expire or has been expired.

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.

New in version 13.8.

- `pending_join_request_count` (`int`, optional) – Number of pending join requests created using this link.

New in version 13.8.

invite_link

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

Type

`str`

creator

Creator of the link.

Type

`telegram.User`

creates_join_request

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

New in version 13.8.

Type`bool`**is_primary**

`True`, if the link is primary.

Type`bool`**is_revoked**

`True`, if the link is revoked.

Type`bool`**expire_date**

Optional. Date when the link will expire or has been expired.

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.

New in version 13.8.

Type`str`**pending_join_request_count**

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

New in version 13.8.

Type`int`**classmethod de_json(data, bot)**

See `telegram.TelegramObject.de_json()`.

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`

New 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.

New 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 24 hours to send messages until the join request is processed, assuming no other administrator contacted the user.

New 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 undocumented 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`)

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.

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`

Available In

`telegram.Chat.location`

MAX_ADDRESS = 64

`telegram.constants.LocationLimit.MAX_CHAT_LOCATION_ADDRESS`

New in version 20.0.

MIN_ADDRESS = 1

`telegram.constants.LocationLimit.MIN_CHAT_LOCATION_ADDRESS`

New in version 20.0.

classmethod de_json(data, bot)

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_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)`

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_messages=None, can_edit_messages=None,
                                         can_pin_messages=None, can_manage_topics=None,
                                         custom_title=None, can_post_stories=None,
                                         can_edit_stories=None, can_delete_stories=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_member()`

New in version 13.7.

Changed in version 20.0:

- Argument and attribute `can_manage_voice_chats` were renamed to `can_manage_video_chats` and `can_manage_video_chats` in accordance to Bot API 6.0.

- The argument `can_manage_topics` was added, which changes the position of the optional argument `custom_title`.

Parameters

- `user` (`telegram.User`) – Information about the user.
- `can_be_edited` (`bool`) – `True`, if the bot is allowed to edit administrator privileges of that user.
- `is_anonymous` (`bool`) – `True`, if the user's presence in the chat is hidden.
- `can_manage_chat` (`bool`) – `True`, if the administrator can access the chat event log, chat statistics, message statistics in channels, see channel members, see anonymous administrators in supergroups and ignore slow mode. Implied by any other administrator privilege.
- `can_delete_messages` (`bool`) – `True`, if the administrator can delete messages of other users.
- `can_manage_video_chats` (`bool`) – `True`, if the administrator can manage video chats.

New in version 20.0.

- `can_restrict_members` (`bool`) – `True`, if the administrator can restrict, ban or unban chat members.
- `can_promote_members` (`bool`) – `True`, if the administrator can add new administrators with a subset of his own privileges or demote administrators that he has promoted, directly or indirectly (promoted by administrators that were appointed by the user).
- `can_change_info` (`bool`) – `True`, if the user can change the chat title, photo and other settings.
- `can_invite_users` (`bool`) – `True`, if the user can invite new users to the chat.
- `can_post_messages` (`bool`, optional) – `True`, if the administrator can post messages in the channel, or access channel statistics; channels only.
- `can_edit_messages` (`bool`, optional) – `True`, if the administrator can edit messages of other users and can pin messages; channels only.
- `can_pin_messages` (`bool`, optional) – `True`, if the user is allowed to pin messages; groups and supergroups only.
- `can_post_stories` (`bool`, optional) – `True`, if the administrator can post stories in the channel; channels only.

New in version 20.6.

- `can_edit_stories` (`bool`, optional) – `True`, if the administrator can edit stories posted by other users; channels only.

New in version 20.6.

- `can_delete_stories` (`bool`, optional) – `True`, if the administrator can delete stories posted by other users; channels only.

New in version 20.6.

- `can_manage_topics` (`bool`, optional) – `True`, if the user is allowed to create, rename, close, and reopen forum topics; supergroups only.

New in version 20.0.

- `custom_title` (`str`, optional) – Custom title for this user.

status

The member's status in the chat, always *'administrator'*.

Type

str

user

Information about the user.

Type

telegram.User

can_be_edited

True, if the bot is allowed to edit administrator privileges of that user.

Type

bool

is_anonymous

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

Type

bool

can_manage_chat

True, if the administrator can access the chat event log, chat statistics, boost list in channels, see channel members, report spam messages, see anonymous administrators in supergroups and ignore slow mode. Implied by any other administrator privilege.

Type

bool

can_delete_messages

True, if the administrator can delete messages of other users.

Type

bool

can_manage_video_chats

True, if the administrator can manage video chats.

New in version 20.0.

Type

bool

can_restrict_members

True, if the administrator can restrict, ban or unban chat members, 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; channels only.

Type

`bool`

can_edit_messages

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

Type

`bool`

can_pin_messages

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

Type

`bool`

can_post_stories

Optional. `True`, if the administrator can post stories in the channel; channels only.

New in version 20.6.

Type

`bool`

can_edit_stories

Optional. `True`, if the administrator can edit stories posted by other users; channels only.

New in version 20.6.

Type

`bool`

can_delete_stories

Optional. `True`, if the administrator can delete stories posted by other users; channels only.

New in version 20.6.

Type

`bool`

can_manage_topics

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

New in version 20.0.

Type

`bool`

custom_title

Optional. Custom title for this user.

Type

`str`

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_member\(\)](#)

New in version 13.7.

Parameters

- **user** ([telegram.User](#)) – Information about the user.
- **until_date** ([datetime.datetime](#)) – Date when restrictions will be lifted for this user.

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_member()`

New in version 13.7.

Parameters

user (`telegram.User`) – Information about the user.

status

The member's status in the chat, always `'left'`.

Type

`str`

user

Information about the user.

Type

`telegram.User`

ChatMemberMember

class `telegram.ChatMemberMember`(*user*, *, *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_member()`

New in version 13.7.

Parameters

user (`telegram.User`) – Information about the user.

status

The member's status in the chat, always `'member'`.

Type

`str`

user

Information about the user.

Type

`telegram.User`

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_member()`

New in version 13.7.

Parameters

- **user** (`telegram.User`) – Information about the user.
- **is_anonymous** (`bool`) – `True`, if the user’s presence in the chat is hidden.
- **custom_title** (`str`, optional) – Custom title for this user.

status

The member’s status in the chat, always `'creator'`.

Type

`str`

user

Information about the user.

Type

`telegram.User`

is_anonymous

`True`, if the user’s presence in the chat is hidden.

Type

`bool`

custom_title

Optional. Custom title for this user.

Type

`str`

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*, *, *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_member()`

New 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.

New 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.

New in version 20.1.

- **`can_send_documents`** (`bool`) – `True`, if the user is allowed to send documents.

New in version 20.1.

- **`can_send_photos`** (`bool`) – `True`, if the user is allowed to send photos.

New in version 20.1.

- **`can_send_videos`** (`bool`) – `True`, if the user is allowed to send videos.
New in version 20.1.
- **`can_send_video_notes`** (`bool`) – `True`, if the user is allowed to send video notes.
New in version 20.1.
- **`can_send_voice_notes`** (`bool`) – `True`, if the user is allowed to send voice notes.
New in version 20.1.

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.

New 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.

New in version 20.1.

Type

`bool`

can_send_documents

`True`, if the user is allowed to send documents.

New in version 20.1.

Type

`bool`

can_send_photos

`True`, if the user is allowed to send photos.

New in version 20.1.

Type

`bool`

can_send_videos

`True`, if the user is allowed to send videos.

New in version 20.1.

Type

`bool`

can_send_video_notes

`True`, if the user is allowed to send video notes.

New in version 20.1.

Type

`bool`

can_send_voice_notes

`True`, if the user is allowed to send voice notes.

New in version 20.1.

Type
`bool`

ChatMemberUpdated

```
class telegram.ChatMemberUpdated(chat, from_user, date, old_chat_member, new_chat_member,
                                  invite_link=None, via_chat_folder_invite_link=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`

New 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

New in version 20.3.

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

New in version 20.3.

Type

`bool`

classmethod de_json(data, bot)

See *telegram.TelegramObject.de_json()*.

difference()

Computes the difference between *old_chat_member* and *new_chat_member*.

Example

```
>>> chat_member_updated.difference()
{'custom_title': ('old title', 'new title')}
```

Note: To determine, if the *telegram.ChatMember.user* attribute has changed, *every* attribute of the user will be checked.

New in version 13.5.

Returns

A dictionary mapping attribute names to tuples of the form (old_value, new_value)

Return type

Dict[str, Tuple[object, object]]

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, *,
                               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](#), and [can_manage_topics](#) are equal.

Use In

- [telegram.Bot.restrict_chat_member\(\)](#)
 - [telegram.Bot.set_chat_permissions\(\)](#)
-

Available In[telegram.Chat.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](#).

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`.

New in version 20.0.

- **`can_send_audios`** (`bool`) – `True`, if the user is allowed to send audios.

New in version 20.1.

- **`can_send_documents`** (`bool`) – `True`, if the user is allowed to send documents.

New in version 20.1.

- **`can_send_photos`** (`bool`) – `True`, if the user is allowed to send photos.

New in version 20.1.

- **`can_send_videos`** (`bool`) – `True`, if the user is allowed to send videos.

New in version 20.1.

- **`can_send_video_notes`** (`bool`) – `True`, if the user is allowed to send video notes.

New in version 20.1.

- **`can_send_voice_notes`** (`bool`) – `True`, if the user is allowed to send voice notes.

New in version 20.1.

`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`.

New in version 20.0.

Type`bool`**can_send_audios**

`True`, if the user is allowed to send audios.

New in version 20.1.

Type`bool`**can_send_documents**

`True`, if the user is allowed to send documents.

New in version 20.1.

Type`bool`**can_send_photos**

`True`, if the user is allowed to send photos.

New in version 20.1.

Type`bool`**can_send_videos**

`True`, if the user is allowed to send videos.

New in version 20.1.

Type`bool`**can_send_video_notes**

`True`, if the user is allowed to send video notes.

New in version 20.1.

Type`bool`**can_send_voice_notes**

`True`, if the user is allowed to send voice notes.

New in version 20.1.

Type`bool`**classmethod** `all_permissions()`

This method returns an `ChatPermissions` instance with all attributes set to `True`. This is e.g. useful when unrestricting a chat member with `telegram.Bot.restrict_chat_member()`.

New in version 20.0.

classmethod `de_json(data, bot)`

See `telegram.TelegramObject.de_json()`.

classmethod `no_permissions()`

This method returns an `ChatPermissions` instance with all attributes set to `False`.

New 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.

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`

Use In

`telegram.Bot.get_file()`

Available In

`telegram.Chat.photo`

SIZE_BIG = 640

`telegram.constants.ChatPhotoSize.BIG`

New in version 20.0.

SIZE_SMALL = 160

`telegram.constants.ChatPhotoSize.SMALL`

New 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*, *, *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

New 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.

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`

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.

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`

Use In

`telegram.Bot.send_contact()`

Available In

- `telegram.ExternalReplyInfo.contact`
 - `telegram.Message.contact`
-

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.

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`

Available In

- `telegram.ExternalReplyInfo.dice`
 - `telegram.Message.dice`
-

ALL_EMOJI = [`DiceEmoji.DICE`, `DiceEmoji.DARTS`, `DiceEmoji.BASKETBALL`, `DiceEmoji.FOOTBALL`, `DiceEmoji.SLOT_MACHINE`, `DiceEmoji.BOWLING`]

A list of all available dice emoji.

Type

`List[str]`

BASKETBALL = ''

`telegram.constants.DiceEmoji.BASKETBALL`

BOWLING = ''

`telegram.constants.DiceEmoji.BOWLING`

New in version 13.4.

DARTS = ''

`telegram.constants.DiceEmoji.DARTS`

DICE = ''

`telegram.constants.DiceEmoji.DICE`

FOOTBALL = ''

`telegram.constants.DiceEmoji.FOOTBALL`

MAX_VALUE_BASKETBALL = 5

`telegram.constants.DiceLimit.MAX_VALUE_BASKETBALL`

New in version 20.0.

MAX_VALUE_BOWLING = 6

`telegram.constants.DiceLimit.MAX_VALUE_BOWLING`

New in version 20.0.

MAX_VALUE_DARTS = 6

`telegram.constants.DiceLimit.MAX_VALUE_DARTS`

New in version 20.0.

MAX_VALUE_DICE = 6

`telegram.constants.DiceLimit.MAX_VALUE_DICE`

New in version 20.0.

MAX_VALUE_FOOTBALL = 5

telegram.constants.DiceLimit.MAX_VALUE_FOOTBALL

New in version 20.0.

MAX_VALUE_SLOT_MACHINE = 64

telegram.constants.DiceLimit.MAX_VALUE_SLOT_MACHINE

New in version 20.0.

MIN_VALUE = 1

telegram.constants.DiceLimit.MIN_VALUE

New in version 20.0.

SLOT_MACHINE = ''

telegram.constants.DiceEmoji.SLOT_MACHINE

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.ExternalReplyInfo.document*
 - *telegram.Message.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 sender.
- ***mime_type*** (*str*, optional) – MIME type of the file as defined by sender.
- ***file_size*** (*int*, optional) – File size in bytes.
- ***thumbnail*** (*telegram.PhotoSize*, optional) – Document thumbnail as defined by sender.

New 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 sender.

Type

`str`

mime_type

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

Type

`str`

file_size

Optional. File size in bytes.

Type

`int`

thumbnail

Optional. Document thumbnail as defined by sender.

New in version 20.2.

Type

`telegram.PhotoSize`

classmethod de_json(data, bot)

See `telegram.TelegramObject.de_json()`.

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

Convenience wrapper over `telegram.Bot.get_file()`

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

Returns

`telegram.File`

Raises

`telegram.error.TelegramError` –

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, *,
                                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](#)

New 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.

- **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.
- **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.
- **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.

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

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.

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.

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

classmethod `de_json(data, bot)`

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.

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`.
New in version 20.0.
- **`write_timeout`** (`float` | `None`, optional) – Value to pass to `telegram.request.BaseRequest.post.write_timeout`. Defaults to `DEFAULT_NONE`.
New in version 20.0.
- **`connect_timeout`** (`float` | `None`, optional) – Value to pass to `telegram.request.BaseRequest.post.connect_timeout`. Defaults to `DEFAULT_NONE`.
New in version 20.0.

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

New in version 20.0.

Returns

The same object as `buf` if it was specified. Otherwise a newly allocated `bytearray`.

Return type

`bytearray`

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 the file has no filename, the file ID will be used as filename. 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()`.

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`

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()`.

New in version 20.0.

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`.

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.

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_location()`

- `telegram.Bot.send_message()`
 - `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.

New 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.

New in version 13.7.

Type

`str`

`MAX_INPUT_FIELD_PLACEHOLDER = 64`

`telegram.constants.ReplyLimit.MAX_INPUT_FIELD_PLACEHOLDER`

New in version 20.0.

`MIN_INPUT_FIELD_PLACEHOLDER = 1`

`telegram.constants.ReplyLimit.MIN_INPUT_FIELD_PLACEHOLDER`

New in version 20.0.

ForumTopic

```
class telegram.ForumTopic(message_thread_id, name, icon_color, icon_custom_emoji_id=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\(\)`](#)

New 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.

`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`

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`

New in version 20.0.

ForumTopicCreated

```
class telegram.ForumTopicCreated(name, icon_color, icon_custom_emoji_id=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`

New 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.

`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`

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

New 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

New 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

New 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](#)

New in version 20.0.

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, *, 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](#)
-

New 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.
- **premium_subscription_month_count** ([int](#), optional) – The number of months the Telegram Premium subscription won from the giveaway will be active for.

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`]

premium_subscription_month_count

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

Type

`int`

classmethod de_json(data, bot)

See [*telegram.TelegramObject.de_json*](#).

GiveawayCompleted

```
class telegram.GiveawayCompleted(winner_count, unclaimed_prize_count=None,
                                  giveaway_message=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](#)

New 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

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](#)

classmethod de_json(data, bot)

See [telegram.TelegramObject.de_json](#).

GiveawayCreated

```
class telegram.GiveawayCreated(*, api_kwargs=None)
```

Bases: [telegram.TelegramObject](#)

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

Available In

[telegram.Message.giveaway_created](#)

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, *, 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`
-

New 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
- **`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`

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)

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](#)
-

New 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, *, 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](#) and [pay](#) are equal.

Note:

- You must use exactly one of the optional fields. Mind that [callback_game](#) is not working as expected. Putting a game short name in it might, but is not guaranteed to work.

- If your bot allows for arbitrary callback data, in keyboards returned in a response from telegram, `callback_data` maybe be an instance of `telegram.ext.InvalidCallbackData`. This will be the case, if the data associated with the button was already deleted.

New in version 13.6.

- Since Bot API 5.5, it's now allowed to mention users by their ID in inline keyboards. This will only work in Telegram versions released after December 7, 2021. Older clients will display *unsupported message*.
-

Warning:

- If your bot allows your arbitrary callback data, buttons whose callback data is a non-hashable object will become unhashable. Trying to evaluate `hash(button)` will result in a `TypeError`.

Changed in version 13.6.

- After Bot API 6.1, only HTTPS links will be allowed in `login_url`.
-

Examples

- [Inline Keyboard 1](#)
 - [Inline Keyboard 2](#)
-

Available In

`telegram.InlineKeyboardMarkup.inline_keyboard`

See also:

`telegram.InlineKeyboardMarkup`

Changed in version 20.0: `web_app` is considered as well when comparing objects of this type in terms of equality.

Parameters

- **text** (`str`) – Label text on the button.
- **url** (`str`, optional) – HTTP or tg:// url to be opened when the button is pressed. Links `tg://user?id=<user_id>` can be used to mention a user by their ID without using a username, if this is allowed by their privacy settings.

Changed in version 13.9: You can now mention a user using `tg://user?id=<user_id>`.

- **login_url** (`telegram.LoginUrl`, optional) – An HTTPS URL used to automatically authorize the user. Can be used as a replacement for the Telegram Login Widget.

Caution: Only HTTPS links are allowed after Bot API 6.1.

- **callback_data** (`str` | `object`, optional) – Data to be sent in a callback query to the bot when button is pressed, UTF-8 1- 64 bytes. If the bot instance allows arbitrary callback data, anything can be passed.

Tip: The value entered here will be available in `telegram.CallbackQuery.data`.

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.

New in version 20.0.

- **switch_inline_query** (*str*, optional) – If set, pressing the button will prompt the user to select one of their chats, open that chat and insert the bot's username and the specified inline query in the input field. Can be empty, in which case just the bot's username will be inserted. This offers an easy way for users to start using your bot in inline mode when they are currently in a private chat with it. Especially useful when combined with `switch_pm*` actions - in this case the user will be automatically returned to the chat they switched from, skipping the chat selection screen.

Tip: This is similar to the new 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. Can be empty, in which case only the bot's username will be inserted. This offers a quick way for the user to open your bot in inline mode in the same chat - good for selecting something from multiple options.
- **callback_game** (*telegram.CallbackGame*, optional) – Description of the game that will be launched when the user presses the button. This type of button **must** always be the **first** button in the first row.
- **pay** (*bool*, optional) – Specify `True`, to send a Pay button. This type of button **must** always be the **first** button in the first row and can only be used in invoice messages.
- **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.

New 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.

text

Label text on the button.

Type

str

url

Optional. HTTP or `tg://` url to be opened when the button is pressed. Links `tg://user?id=<user_id>` can be used to mention a user by their ID without using a username, if this is allowed by their privacy settings.

Changed in version 13.9: You can now mention a user using `tg://user?id=<user_id>`.

Type
`str`

`login_url`

Optional. An HTTPS URL used to automatically authorize the user. Can be used as a replacement for the Telegram Login Widget.

Caution: Only HTTPS links are allowed after Bot API 6.1.

Type
`telegram.LoginUrl`

`callback_data`

Optional. Data to be sent in a callback query to the bot when button is pressed, UTF-8 1- 64 bytes.

Type
`str | object`

`web_app`

Optional. Description of the [Web App](#) that will be launched when the user presses the button. The Web App will be able to send an arbitrary message on behalf of the user using the method [answer_web_app_query\(\)](#). Available only in private chats between a user and the bot.

New in version 20.0.

Type
`telegram.WebAppInfo`

`switch_inline_query`

Optional. If set, pressing the button will prompt the user to select one of their chats, open that chat and insert the bot's username and the specified inline query in the input field. Can be empty, in which case just the bot's username will be inserted. This offers an easy way for users to start using your bot in inline mode when they are currently in a private chat with it. Especially useful when combined with `switch_pm*` actions - in this case the user will be automatically returned to the chat they switched from, skipping the chat selection screen.

Tip: This is similar to the new 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. Can be empty, in which case only the bot's username will be inserted. This offers a quick way for the user to open your bot in inline mode in the same chat - good for selecting something from multiple options.

Type
`str`

`callback_game`

Optional. Description of the game that will be launched when the user presses the button. This type of button **must** always be the **first** button in the first row.

Type
`telegram.CallbackGame`

pay

Optional. Specify `True`, to send a Pay button. This type of button **must** always be the **first** button in the first row and can only be used in invoice messages.

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.

New 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`

MAX_CALLBACK_DATA = 64

`telegram.constants.InlineKeyboardButtonLimit.MAX_CALLBACK_DATA`

New in version 20.0.

MIN_CALLBACK_DATA = 1

`telegram.constants.InlineKeyboardButtonLimit.MIN_CALLBACK_DATA`

New in version 20.0.

classmethod de_json(data, bot)

See `telegram.TelegramObject.de_json()`.

update_callback_data(callback_data)

Sets `callback_data` to the passed object. Intended to be used by `telegram.ext.CallbackDataCache`.

New in version 13.6.

Parameters

`callback_data` (object) – The new callback data.

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.

Use In

- `telegram.Bot.copy_message()`
- `telegram.Bot.edit_message_caption()`
- `telegram.Bot.edit_message_live_location()`

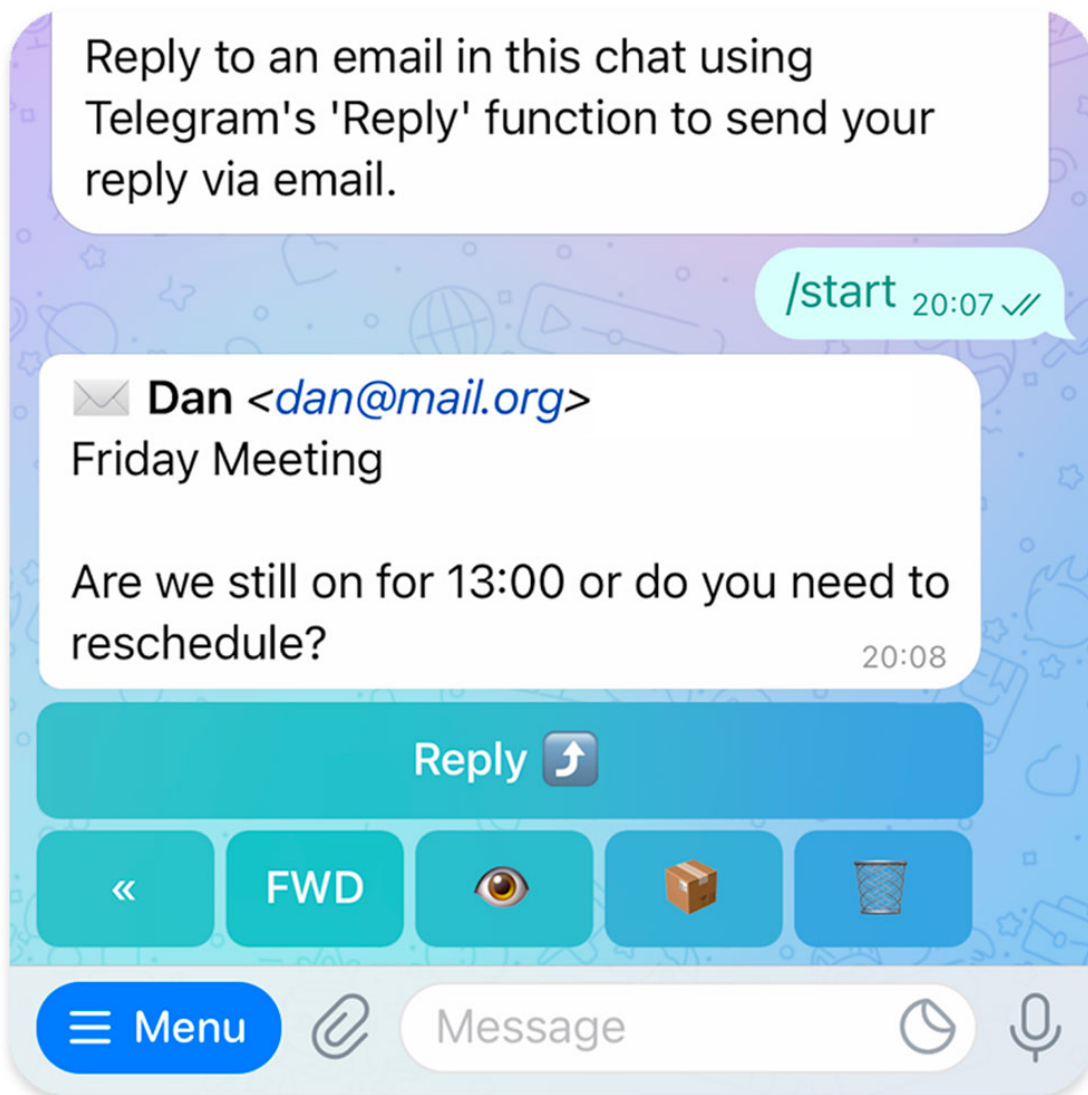


Fig. 1: An inline keyboard on a message

- `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_contact()`
 - `telegram.Bot.send_dice()`
 - `telegram.Bot.send_document()`
 - `telegram.Bot.send_game()`
 - `telegram.Bot.send_invoice()`
 - `telegram.Bot.send_location()`
 - `telegram.Bot.send_message()`
 - `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`
-

See also:

Another kind of keyboard would be the `telegram.ReplyKeyboardMarkup`.

Examples

- *Inline Keyboard 1*
 - *Inline Keyboard 2*
-

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*)

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.

InputFile

class `telegram.InputFile(obj, filename=None, attach=False)`

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_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.InputMediaPhoto.media`
- `telegram.InputMediaVideo.media`

- `telegram.InputMediaVideo.thumbnail`
 - `telegram.InputSticker.sticker`
-

Changed in version 20.0:

- The former attribute `attach` was renamed to `attach_name`.
- Method `is_image` was removed. If you pass `bytes` to `obj` and would like to have the mime type automatically guessed, please pass `filename` in addition.

Parameters

- **`obj`** (file object | bytes | str) – An open file descriptor or the files content as bytes or string.

Note: If `obj` is a string, it will be encoded as bytes via `obj.encode('utf-8')`.

Changed in version 20.0: Accept string input.

- **`filename`** (str, optional) – Filename for this InputFile.
- **`attach`** (bool, optional) – Pass `True` if the parameter this file belongs to in the request to Telegram should point to the multipart data via an `attach://` URI. Defaults to `False`.

input_file_content

The binary content of the file to send.

Type

bytes

attach_name

Optional. If present, the parameter this file belongs to in the request to Telegram should point to the multipart data via a an URI of the form `attach://<attach_name> URI`.

Type

str

filename

Filename for the file to be sent.

Type

str

mimetype

The mimetype inferred from the file to be sent.

Type

str

property attach_uri

URI to insert into the JSON data for uploading the file. Returns `None`, if `attach_name` is `None`.

property field_tuple

Field tuple representing the contents of the file for upload to the Telegram servers.

Return type

Tuple[str, bytes, str]

InputMedia

```
class telegram.InputMedia(media_type, media, caption=None, caption_entities=None,
                           parse_mode=None, *, api_kwargs=None)
```

Bases: [telegram.TelegramObject](#)

Base class for Telegram InputMedia Objects.

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](#) | [file object](#) | [bytes](#) | [pathlib.Path](#) | [telegram.Animation](#) | [telegram.Audio](#) | [telegram.Document](#) | [telegram.PhotoSize](#) | [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 media object of the corresponding type to send.
- **caption** ([str](#), optional) – Caption of the media to be sent, 0-[1024](#) characters after entities parsing.
- **caption_entities** ([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=None, has_spoiler=None, thumbnail=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 video, unless otherwise specified with the optional arguments.

Use In

[telegram.Bot.edit_message_media\(\)](#)

See also:

[Working with Files and Media](#)

Changed in version 20.5: Removed the deprecated argument and attribute `thumb`.

Parameters

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

New in version 13.1.

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

New 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.

New in version 20.2.

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.

TypeTuple[[telegram.MessageEntity](#)]**width**

Optional. Animation width.

Type[int](#)**height**

Optional. Animation height.

Type[int](#)**duration**

Optional. Animation duration in seconds.

Type[int](#)**has_spoiler**Optional. [True](#), if the animation is covered with a spoiler animation.

New 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.

New in version 20.2.

Type[telegram.InputFile](#)

InputMediaAudio

```
class telegram.InputMediaAudio(media, caption=None, parse_mode=None, duration=None,
                               performer=None, title=None, caption_entities=None, filename=None,
                               thumbnail=None, *, api_kwargs=None)
```

Bases: [telegram.InputMedia](#)

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

Use In

- [telegram.Bot.edit_message_media\(\)](#)
 - [telegram.Bot.send_media_group\(\)](#)
-

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.

Changed in version 20.5: Removed the deprecated argument and attribute `thumb`.

Parameters

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

New in version 13.1.

- **caption** (`str`, optional) – Caption of the audio to be sent, 0-1024 characters after entities parsing.
- **parse_mode** (`str`, optional) – 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`, optional) – Duration of the audio in seconds as defined by sender.
- **performer** (`str`, optional) – Performer of the audio as defined by sender or by audio tags.
- **title** (`str`, optional) – Title of the audio as defined by sender or by audio tags.
- **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.

New in version 20.2.

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.

Type

[int](#)

performer

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

Type

[str](#)

title

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

Type

[str](#)

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.

New 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=None, thumbnail=None, *, api_kwargs=None)
```

Bases: [telegram.InputMedia](#)

Represents a general file to be sent.

Use In

- [telegram.Bot.edit_message_media\(\)](#)
- [telegram.Bot.send_media_group\(\)](#)

See also:

[Working with Files and Media](#)

Changed in version 20.5: Removed the deprecated argument and attribute `thumb`.

Parameters

- **`media`** (`str` | `file object` | `bytes` | `pathlib.Path` | `telegram.Document`) – File to send. Pass a `file_id` as `String` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL as a `String` for Telegram to get a file from the Internet, or upload a new one. 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`** (`str`, optional) – Custom file name for the document, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the `tempfile` module.

New in version 13.1.

- **`caption`** (`str`, optional) – Caption of the document to be sent, 0-1024 characters after entities parsing.
- **`parse_mode`** (`str`, optional) – 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.

New in version 20.2.

`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.

TypeTuple[[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.

New in version 20.2.

Type[telegram.InputFile](#)

InputMediaPhoto

```
class telegram.InputMediaPhoto(media, caption=None, parse_mode=None, caption_entities=None,
                               filename=None, has_spoiler=None, *, api_kwargs=None)
```

Bases: [telegram.InputMedia](#)

Represents a photo to be sent.

Use In

- [telegram.Bot.edit_message_media\(\)](#)
 - [telegram.Bot.send_media_group\(\)](#)
-

See also:

[Working with Files and Media](#)

Parameters

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

New in version 13.1.

- **caption** (`str`, optional) – Caption of the photo to be sent, 0-1024 characters after entities parsing.
- **parse_mode** (`str`, optional) – 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.

New in version 20.0.

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.

TypeTuple[[telegram.MessageEntity](#)]**has_spoiler**Optional. **True**, if the photo is covered with a spoiler animation.

New in version 20.0.

Type

bool

InputMediaVideo

```
class telegram.InputMediaVideo(media, caption=None, width=None, height=None, duration=None,
                               supports_streaming=None, parse_mode=None, caption_entities=None,
                               filename=None, has_spoiler=None, thumbnail=None, *,
                               api_kwargs=None)
```

Bases: [telegram.InputMedia](#)

Represents a video to be sent.

Use In

- [telegram.Bot.edit_message_media\(\)](#)
 - [telegram.Bot.send_media_group\(\)](#)
-

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.
-

Changed in version 20.5: Removed the deprecated argument and attribute `thumb`.**Parameters**

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

New in version 13.1.

- ***caption*** ([str](#), optional) – Caption of the video to be sent, 0-1024 characters after entities parsing.

- **`parse_mode`** (`str`, optional) – 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`, optional) – Video duration in seconds.
- **`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.

New 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.

New in version 20.2.

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.

TypeTuple[*telegram.MessageEntity*]**width**

Optional. Video width.

Type*int***height**

Optional. Video height.

Type*int***duration**

Optional. Video duration in seconds.

Type*int***supports_streaming**Optional. *True*, if the uploaded video is suitable for streaming.**Type***bool***has_spoiler**Optional. *True*, if the video is covered with a spoiler animation.

New 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.

New in version 20.2.

Type*telegram.InputFile*

InputSticker

```
class telegram.InputSticker(sticker, emoji_list, 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()*
-

New in version 20.2.

Parameters

- **sticker** (`str` | `file object` | `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.

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.

Type

`Tuple[str]`

KeyboardButton

```
class telegram.KeyboardButton(text, request_contact=None, request_location=None, request_poll=None,
                              web_app=None, request_user=None, request_chat=None,
                              request_users=None, *, api_kwargs=None)
```

Bases: `telegram.TelegramObject`

This object represents one button of the reply keyboard. 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` and `request_chat` 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.
-

Available In

`telegram.ReplyKeyboardMarkup.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 20.5: `request_users` and `request_chat` 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 optional fields are used, it will be sent to the bot as a message when the button is pressed.
- `request_contact` (`bool`, optional) – If `True`, the user's phone number will be sent as a contact when the button is pressed. Available in private chats only.
- `request_location` (`bool`, optional) – If `True`, the user's current location will be sent when the button is pressed. Available in private chats only.
- `request_poll` (`KeyboardButtonPollType`, optional) – If specified, the user will be asked to create a poll and send it to the bot when the button is pressed. Available in private chats only.
- `web_app` (`WebAppInfo`, optional) – If specified, the described Web App will be launched when the button is pressed. The Web App will be able to send a `Message.web_app_data` service message. Available in private chats only.

New in version 20.0.

- `request_user` (`KeyboardButtonRequestUser` | `KeyboardButtonRequestUsers`, optional) – Alias for `request_users`.

New in version 20.1.

Deprecated since version 20.8: Bot API 7.0 deprecates this argument in favor of `request_users``.

- `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.user_shared` service message. Available in private chats only.

New 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.

New in version 20.1.

`text`

Text of the button. If none of the optional fields are used, it will be sent to the bot as a message when the button is pressed.

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.

New 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.user_shared` service message. Available in private chats only.

New 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.

New in version 20.1.

Type

`KeyboardButtonRequestChat`

classmethod de_json(data, bot)

See `telegram.TelegramObject.de_json()`.

property request_user

Alias for `request_users`.

New in version 20.1.

Deprecated since version 20.8: Bot API 7.0 deprecates this attribute in favor of `request_users`.

Type

Optional[`KeyboardButtonRequestUsers`]

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

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](#)

Available In

[telegram.KeyboardButton.request_poll](#)

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, *, 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.

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](#)

See also:

[Telegram Docs on requesting chats](#)

New 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_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`

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`, optional

`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***classmethod de_json(data, bot)**

See *telegram.TelegramObject.de_json()*.

KeyboardButtonRequestUser

```
class telegram.KeyboardButtonRequestUser(request_id, user_is_bot=None, user_is_premium=None,
                                         max_quantity=None, *, api_kwargs=None)
```

Bases: *telegram.KeyboardButtonRequestUsers*

Alias for *KeyboardButtonRequestUsers*, kept for backward compatibility.

Available In*telegram.KeyboardButton.request_users*

New in version 20.1.

Deprecated since version 20.8: Use *KeyboardButtonRequestUsers* instead.

KeyboardButtonRequestUsers

```
class telegram.KeyboardButtonRequestUsers(request_id, user_is_bot=None, user_is_premium=None,
                                           max_quantity=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.

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*

See also:

[Telegram Docs on requesting users](#)

New 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`.

New in version 20.8.

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`.

New in version 20.8.

Type

`int`

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.Message.link_preview_options`
 - `telegram.ext.Defaults.link_preview_options`
-

New 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`

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.

Parameters

- [longitude](#) ([float](#)) – Longitude as defined by sender.
- [latitude](#) ([float](#)) – Latitude as defined by sender.
- [horizontal_accuracy](#) ([float](#), optional) – The radius of uncertainty for the location, measured in meters; 0-1500.
- [live_period](#) ([int](#), optional) – Time relative to the message sending date, during which the location can be updated, in seconds. For active live locations only.
- [heading](#) ([int](#), optional) – The direction in which user is moving, in degrees; 1-360. For active live locations only.
- [proximity_alert_radius](#) ([int](#), optional) – Maximum distance for proximity alerts about approaching another chat member, in meters. For sent live locations only.

longitude

Longitude as defined by sender.

Type

[float](#)

latitude

Latitude as defined by sender.

Type

[float](#)

horizontal_accuracy

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

Type

[int](#)

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](#)

Use In

- `telegram.Bot.edit_message_live_location()`
 - `telegram.Bot.send_location()`
-

Available In

- `telegram.ChatLocation.location`
 - `telegram.ChosenInlineResult.location`
 - `telegram.ExternalReplyInfo.location`
 - `telegram.InlineQuery.location`
 - `telegram.Message.location`
 - `telegram.Venue.location`
-

HORIZONTAL_ACCURACY = 1500

`telegram.constants.LocationLimit.HORIZONTAL_ACCURACY`

New in version 20.0.

MAX_HEADING = 360

`telegram.constants.LocationLimit.MAX_HEADING`

New in version 20.0.

MIN_HEADING = 1

`telegram.constants.LocationLimit.MIN_HEADING`

New in version 20.0.

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](#)

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`

Available In

`telegram.InlineKeyboardButton.login_url`

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`
-

New 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`

__bool__()

Overrides `object.__bool__()` to return the value of `is_accessible`. This is intended to ease migration to Bot API 7.0, as this allows checks like

```
if message.pinned_message:
    ...
```

to work as before, when `message.pinned_message` was `None`. Note that this does not help with check like

```
if message.pinned_message is None:
    ...
```

for cases where `message.pinned_message` is now no longer `None`.

Tip: Since objects that can only be of type `Message` or `None` are not affected by this change, `Message.__bool__()` is not overridden and will continue to work as before.

New in version 20.8.

Deprecated since version 20.8: This behavior is introduced only temporarily to ease migration to Bot API 7.0. It will be removed along with other functionality deprecated by Bot API 7.0.

classmethod de_json(data, bot)

See `telegram.TelegramObject.de_json()`.

property is_accessible

Convenience attribute. `True`, if the date is not 0 in Unix time.

New 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()

New in version 20.0.

Parameters

type (*str*) – Type of menu button that the instance represents.

type

Type of menu button that the instance represents.

Type

str

COMMANDS = 'commands'

telegram.constants.MenuButtonType.COMMANDS

DEFAULT = 'default'

telegram.constants.MenuButtonType.DEFAULT

WEB_APP = 'web_app'

telegram.constants.MenuButtonType.WEB_APP

classmethod **de_json**(*data*, *bot*)

Converts JSON data to the appropriate *MenuButton* object, i.e. takes care of selecting the correct subclass.

Parameters

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

Returns

The Telegram object.

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()

New 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()

New 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\(\)](#)

New in version 20.0.

Parameters

- **text** ([str](#)) – Text of the button.
- **web_app** ([telegram.WebAppInfo](#)) – Description of the Web App that will be launched when the user presses the button. The Web App will be able to send an arbitrary message on behalf of the user using the method [answerWebAppQuery\(\)](#) of [Bot](#).

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](#).

Type

[telegram.WebAppInfo](#)

classmethod **de_json**(*data*, *bot*)

See [telegram.TelegramObject.de_json\(\)](#).

Message

```
class telegram.Message(message_id, date, chat, from_user=None, forward_from=None,
                        forward_from_chat=None, forward_from_message_id=None,
                        forward_date=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, forward_signature=None,
                        author_signature=None, media_group_id=None, connected_website=None,
                        animation=None, passport_data=None, poll=None, forward_sender_name=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, user_shared=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, *, 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.

Available In

- [telegram.CallbackQuery.message](#)
 - [telegram.Chat.pinned_message](#)
 - [telegram.GiveawayCompleted.giveaway_message](#)
 - [telegram.Message.pinned_message](#)
 - [telegram.Message.reply_to_message](#)
 - [telegram.Update.channel_post](#)
 - [telegram.Update.edited_channel_post](#)
 - [telegram.Update.edited_message](#)
 - [telegram.Update.effective_message](#)
 - [telegram.Update.message](#)
-

Returned In

- `telegram.Bot.edit_message_caption()`
 - `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.send_animation()`
 - `telegram.Bot.send_audio()`
 - `telegram.Bot.send_contact()`
 - `telegram.Bot.send_dice()`
 - `telegram.Bot.send_document()`
 - `telegram.Bot.send_game()`
 - `telegram.Bot.send_invoice()`
 - `telegram.Bot.send_location()`
 - `telegram.Bot.send_message()`
 - `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 20.8: * This class is now a subclass of `telegram.MaybeInaccessibleMessage`. * The `pinned_message` now can be either class:`telegram.Message` or class:`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`, `location`, `venue`. Use a named argument for those, and notice that some positional arguments changed position as a result.

Parameters

- **`message_id`** (`int`) – Unique message identifier inside this chat.
- **`from_user`** (`telegram.User`, optional) – Sender of the message; empty for messages sent to channels. For backward compatibility, this will contain a fake sender user in non-channel chats, if the message was sent on behalf of a chat.

- **sender_chat** (*telegram.Chat*, optional) – Sender of the message, sent on behalf of a chat. For example, the channel itself for channel posts, the supergroup itself for messages from anonymous group administrators, the linked channel for messages automatically forwarded to the discussion group. For backward compatibility, *from_user* contains a fake sender user in non-channel chats, if the message was sent on behalf of a chat.
- **date** (*datetime.datetime*) – Date the message was sent in Unix time. Converted to *datetime.datetime*.

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.
- **forward_from** (*telegram.User*, optional) – For forwarded messages, sender of the original message.

Deprecated since version 20.8: Bot API 7.0 deprecates *forward_from* in favor of *forward_origin*.

- **forward_from_chat** (*telegram.Chat*, optional) – For messages forwarded from channels or from anonymous administrators, information about the original sender chat.

Deprecated since version 20.8: Bot API 7.0 deprecates *forward_from_chat* in favor of *forward_origin*.

- **forward_from_message_id** (*int*, optional) – For forwarded channel posts, identifier of the original message in the channel.

Deprecated since version 20.8: Bot API 7.0 deprecates *forward_from_message_id* in favor of *forward_origin*.

- **forward_sender_name** (*str*, optional) – Sender’s name for messages forwarded from users who disallow adding a link to their account in forwarded messages.

Deprecated since version 20.8: Bot API 7.0 deprecates *forward_sender_name* in favor of *forward_origin*.

- **forward_date** (*datetime.datetime*, optional) – For forwarded messages, date the original 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.

Deprecated since version 20.8: Bot API 7.0 deprecates *forward_date* in favor of *forward_origin*.

- **is_automatic_forward** (*bool*, optional) – *True*, if the message is a channel post that was automatically forwarded to the connected discussion group.

New in version 13.9.

- **reply_to_message** (*telegram.Message*, optional) – For replies, the original message. 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.

New in version 13.9.

- **media_group_id** (*str*, optional) – The unique identifier of a media message group this message belongs to.

- **text** (*str*, optional) – For text messages, the actual UTF-8 text of the message, 0-4096 characters.
- **entities** (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.

New in version 20.8.

- **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.

- **audio** (*telegram.Audio*, optional) – Message is an audio file, information about the file.
- **document** (*telegram.Document*, optional) – Message is a general file, information about the file.
- **animation** (*telegram.Animation*, optional) – Message is an animation, information about the animation. For backward compatibility, when this field is set, the document field will also be set.
- **game** (*telegram.Game*, optional) – Message is a game, information about the game.
- **photo** (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.

New 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, 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.

New 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 class:*telegram.Message* or class:*telegram.InaccessibleMessage*.

- **invoice** (*telegram.Invoice*, optional) – Message is an invoice for a payment, information about the invoice.
- **successful_payment** (*telegram.SuccessfulPayment*, optional) – Message is a service message about a successful payment, information about the payment.
- **connected_website** (*str*, optional) – The domain name of the website on which the user has logged in.
- **forward_signature** (*str*, optional) – For messages forwarded from channels, signature of the post author if present.

Deprecated since version 20.8: Bot API 7.0 deprecates *forward_signature* in favor of *forward_origin*.

- **author_signature** (*str*, optional) – Signature of the post author for messages in channels, or the custom title of an anonymous group administrator.
- **passport_data** (*telegram.PassportData*, optional) – Telegram Passport data.
- **poll** (*telegram.Poll*, optional) – Message is a native poll, information about the poll.
- **dice** (*telegram.Dice*, optional) – Message is a dice with random value.
- **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.

New in version 20.0.

- **video_chat_started** (*telegram.VideoChatStarted*, optional) – Service message: video chat started.

New in version 20.0.

- **video_chat_ended** (*telegram.VideoChatEnded*, optional) – Service message: video chat ended.

New in version 20.0.

- **video_chat_participants_invited** (*telegram.VideoChatParticipantsInvited*, optional) – Service message: new participants invited to a video chat.

New in version 20.0.

- **web_app_data** (*telegram.WebAppData*, optional) – Service message: data sent by a Web App.

New in version 20.0.

- **reply_markup** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message. *login_url* buttons are represented as ordinary url buttons.
- **is_topic_message** (*bool*, optional) – *True*, if the message is sent to a forum topic.

New in version 20.0.

- **message_thread_id** (*int*, optional) – Unique identifier of a message thread to which the message belongs; for supergroups only.

New in version 20.0.

- **forum_topic_created** (*telegram.ForumTopicCreated*, optional) – Service message: forum topic created.

New in version 20.0.

- **forum_topic_closed** (*telegram.ForumTopicClosed*, optional) – Service message: forum topic closed.

New in version 20.0.

- **forum_topic_reopened** (*telegram.ForumTopicReopened*, optional) – Service message: forum topic reopened.

New in version 20.0.

- **`forum_topic_edited`** (`telegram.ForumTopicEdited`, optional) – Service message: forum topic edited.

New in version 20.0.

- **`general_forum_topic_hidden`** (`telegram.GeneralForumTopicHidden`, optional) – Service message: General forum topic hidden.

New in version 20.0.

- **`general_forum_topic_unhidden`** (`telegram.GeneralForumTopicUnhidden`, optional) – Service message: General forum topic unhidden.

New 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`.

New in version 20.0.

- **`has_media_spoiler`** (`bool`, optional) – `True`, if the message media is covered by a spoiler animation.

New in version 20.0.

- **`user_shared`** (`telegram.UserShared`, optional) – Service message: a user was shared with the bot.

New in version 20.1.

Deprecated since version 20.8: Bot API 7.0 deprecates `user_shared` in favor of `users_shared`.

- **`users_shared`** (`telegram.UsersShared`, optional) – Service message: users were shared with the bot

New in version 20.8.

- **`chat_shared`** (`telegram.ChatShared`, optional) – Service message: a chat was shared with the bot.

New in version 20.1.

- **`giveaway_created`** (`telegram.GiveawayCreated`, optional) – Service message: a scheduled giveaway was created

New in version 20.8.

- **`giveaway`** (`telegram.Giveaway`, optional) – The message is a scheduled giveaway message

New in version 20.8.

- **`giveaway_winners`** (`telegram.GiveawayWinners`, optional) – A giveaway with public winners was completed

New in version 20.8.

- **`giveaway_completed`** (`telegram.GiveawayCompleted`, optional) – Service message: a giveaway without public winners was completed

New in version 20.8.

- **`external_reply`** (`telegram.ExternalReplyInfo`, optional) – Information about the message that is being replied to, which may come from another chat or forum topic.

New in version 20.8.

- **quote** (*telegram.TextQuote*, optional) – For replies that quote part of the original message, the quoted part of the message.

New in version 20.8.

- **forward_origin** (*telegram.MessageOrigin*, optional) – Information about the original message for forwarded messages

New in version 20.8.

message_id

Unique message identifier inside this chat.

Type

int

from_user

Optional. Sender of the message; empty for messages sent to channels. For backward compatibility, this will contain a fake sender user in non-channel chats, if the message was sent on behalf of a chat.

Type

telegram.User

sender_chat

Optional. Sender of the message, sent on behalf of a chat. For example, the channel itself for channel posts, the supergroup itself for messages from anonymous group administrators, the linked channel for messages automatically forwarded to the discussion group. For backward compatibility, *from_user* contains a fake sender user in non-channel chats, if the message was sent on behalf of a chat.

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.

New in version 13.9.

Type

bool

reply_to_message

Optional. For replies, the original message. Note that the Message object in this field will not contain further *reply_to_message* fields even if it itself is a reply.

Type

telegram.Message

edit_date

Optional. Date the message was last edited 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.

New in version 13.9.

Type

`bool`

media_group_id

Optional. The unique identifier of a media message group this message belongs to.

Type

`str`

text

Optional. 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.

New in version 20.8.

Type

`telegram.LinkPreviewOptions`

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`]

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.

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.

New 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.

TypeTuple[*telegram.User*]**caption**

Optional. Caption for the animation, audio, document, 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.

New in version 13.4.

Type

[*telegram.MessageAutoDeleteTimerChanged*](#)

migrate_to_chat_id

Optional. The group has been migrated to a supergroup with the specified identifier.

Type

`int`

migrate_from_chat_id

Optional. The supergroup has been migrated from a group with the specified identifier.

Type

`int`

pinned_message

Optional. Specified message was pinned. 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 class:*telegram.Message* or class:*telegram.InaccessibleMessage*.

Type

telegram.MaybeInaccessibleMessage

invoice

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

Type

telegram.Invoice

successful_payment

Optional. Message is a service message about a successful payment, information about the payment.

Type

telegram.SuccessfulPayment

connected_website

Optional. The domain name of the website on which the user has logged in.

Type

str

author_signature

Optional. Signature of the post author for messages in channels, or the custom title of an anonymous group administrator.

Type

str

passport_data

Optional. Telegram Passport data.

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.

New in version 20.0.

Type

telegram.VideoChatScheduled

video_chat_started

Optional. Service message: video chat started.

New in version 20.0.

Type

telegram.VideoChatStarted

video_chat_ended

Optional. Service message: video chat ended.

New in version 20.0.

Type

telegram.VideoChatEnded

video_chat_participants_invited

Optional. Service message: new participants invited to a video chat.

New in version 20.0.

Type

telegram.VideoChatParticipantsInvited

web_app_data

Optional. Service message: data sent by a Web App.

New in version 20.0.

Type

telegram.WebAppData

reply_markup

Optional. Inline keyboard attached to the message. *login_url* buttons are represented as ordinary url buttons.

Type

telegram.InlineKeyboardMarkup

is_topic_message

Optional. *True*, if the message is sent to a forum topic.

New in version 20.0.

Type

bool

message_thread_id

Optional. Unique identifier of a message thread to which the message belongs; for supergroups only.

New in version 20.0.

Type`int`**forum_topic_created**

Optional. Service message: forum topic created.

New in version 20.0.

Type`telegram.ForumTopicCreated`**forum_topic_closed**

Optional. Service message: forum topic closed.

New in version 20.0.

Type`telegram.ForumTopicClosed`**forum_topic_reopened**

Optional. Service message: forum topic reopened.

New in version 20.0.

Type`telegram.ForumTopicReopened`**forum_topic_edited**

Optional. Service message: forum topic edited.

New in version 20.0.

Type`telegram.ForumTopicEdited`**general_forum_topic_hidden**

Optional. Service message: General forum topic hidden.

New in version 20.0.

Type`telegram.GeneralForumTopicHidden`**general_forum_topic_unhidden**

Optional. Service message: General forum topic unhidden.

New 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.

New in version 20.0.

Type`telegram.WriteAccessAllowed`**has_media_spoiler**

Optional. `True`, if the message media is covered by a spoiler animation.

New in version 20.0.

Type`bool`

users_shared

Optional. Service message: users were shared with the bot

New in version 20.8.

Type

telegram.UsersShared

chat_shared

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

New in version 20.1.

Type

telegram.ChatShared

giveaway_created

Optional. Service message: a scheduled giveaway was created

New in version 20.8.

Type

telegram.GiveawayCreated

giveaway

Optional. The message is a scheduled giveaway message

New in version 20.8.

Type

telegram.Giveaway

giveaway_winners

Optional. A giveaway with public winners was completed

New in version 20.8.

Type

telegram.GiveawayWinners

giveaway_completed

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

New in version 20.8.

Type

telegram.GiveawayCompleted

external_reply

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

New in version 20.8.

Type

telegram.ExternalReplyInfo

quote

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

New in version 20.8.

Type

telegram.TextQuote

forward_origin

Information about the original message for forwarded messages

New in version 20.8.

Type

`telegram.MessageOrigin`, optional

__bool__()

Overrides `telegram.MaybeInaccessibleMessage.__bool__()` to use Python's default implementation of `object.__bool__()` instead.

Tip: The current behavior is the same as before the introduction of `telegram.MaybeInaccessibleMessage`. This documentation is relevant only until `telegram.MaybeInaccessibleMessage.__bool__()` is removed.

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
    )
)
```

New 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()`.

New 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)
```

New 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_to_message_id=None,
           allow_sending_without_reply=None, reply_markup=None, protect_content=None,
           message_thread_id=None, reply_parameters=None, *, read_timeout=None,
           write_timeout=None, connect_timeout=None, pool_timeout=None, api_kwargs=None)
```

Shortcut for:

```
await bot.copy_message(
    chat_id=chat_id,
    from_chat_id=update.effective_message.chat_id,
    message_id=update.effective_message.message_id,
    *args,
    **kwargs
)
```

For the documentation of the arguments, please see [telegram.Bot.copy_message\(\)](#).

Returns

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

Return type

[telegram.MessageId](#)

classmethod `de_json(data, bot)`

See [telegram.TelegramObject.de_json\(\)](#).

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

Shortcut for:

```
await bot.delete_message(
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs
)
```

For the documentation of the arguments, please see [telegram.Bot.delete_message\(\)](#).

Returns

On success, `True` is returned.

Return type

`bool`

```
async 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\(\)](#).

New in version 20.0.

Returns

On success, `True` is returned.

Return type

`bool`


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

Shortcut for:

```
await bot.edit_message_caption(  
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs  
)
```

For the documentation of the arguments, please see [telegram.Bot.edit_message_caption\(\)](#).

Note: You can only edit messages that the bot sent itself (i.e. of the `bot.send_*` family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

Returns

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

Return type

[telegram.Message](#)

```
async edit_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\(\)](#).

New 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, *, location=None, read_timeout=None,  
                           write_timeout=None, connect_timeout=None, pool_timeout=None,  
                           api_kwargs=None)
```

Shortcut for:

```
await bot.edit_message_live_location(  
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs  
)
```

For the documentation of the arguments, please see [telegram.Bot.edit_message_live_location\(\)](#).

Note: You can only edit messages that the bot sent itself (i.e. of the `bot.send_*` family of methods)

or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

Returns

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

Return type

`telegram.Message`

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

Shortcut for:

```
await bot.edit_message_media(
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs
)
```

For the documentation of the arguments, please see `telegram.Bot.edit_message_media()`.

Note: You can only edit messages that the bot sent itself (i.e. of the `bot.send_*` family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

Returns

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

Return type

`telegram.Message`

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

Shortcut for:

```
await bot.edit_message_reply_markup(
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs
)
```

For the documentation of the arguments, please see `telegram.Bot.edit_message_reply_markup()`.

Note: You can only edit messages that the bot sent itself (i.e. of the `bot.send_*` family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

Returns

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

Return type

`telegram.Message`

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

Shortcut for:

```
await bot.edit_message_text(
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs
)
```

For the documentation of the arguments, please see `telegram.Bot.edit_message_text()`.

Note: You can only edit messages that the bot sent itself (i.e. of the `bot.send_*` family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

Returns

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

Return type

`telegram.Message`

property `effective_attachment`

If this message is neither a plain text message nor a status update, this gives the attachment that this message was sent with. This may be one of

- `telegram.Audio`
- `telegram.Dice`
- `telegram.Contact`
- `telegram.Document`
- `telegram.Animation`
- `telegram.Game`
- `telegram.Invoice`
- `telegram.Location`
- `telegram.PassportData`
- `List[telegram.PhotoSize]`
- `telegram.Poll`
- `telegram.Sticker`
- `telegram.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.

```
async forward(chat_id, disable_notification=None, protect_content=None, message_thread_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,
    *args,
    **kwargs
)
```

For the documentation of the arguments, please see [telegram.Bot.forward_message\(\)](#).

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

As a workaround, it is still possible to use [copy\(\)](#). However, this behaviour is undocumented and might be changed by Telegram.

Returns

On success, instance representing the message forwarded.

Return type

[telegram.Message](#)

property forward_date

Optional. For forwarded messages, date the original 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.

Deprecated since version 20.8: Bot API 7.0 deprecates [forward_date](#) in favor of [forward_origin](#).

Type

[datetime.datetime](#)

property forward_from

Optional. For forwarded messages, sender of the original message.

Deprecated since version 20.8: Bot API 7.0 deprecates [forward_from](#) in favor of [forward_origin](#).

Type

[telegram.User](#)

property forward_from_chat

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

Deprecated since version 20.8: Bot API 7.0 deprecates [forward_from_chat](#) in favor of [forward_origin](#).

Type

[telegram.Chat](#)

property forward_from_message_id

Optional. For forwarded channel posts, identifier of the original message in the channel.

Deprecated since version 20.8: Bot API 7.0 deprecates `forward_from_message_id` in favor of `forward_origin`.

Type

`int`

property forward_sender_name

Optional. Sender's name for messages forwarded from users who disallow adding a link to their account in forwarded messages.

Deprecated since version 20.8: Bot API 7.0 deprecates `forward_sender_name` in favor of `forward_origin`.

Type

`telegram.User`

property forward_signature

Optional. For messages forwarded from channels, signature of the post author if present.

Deprecated since version 20.8: Bot API 7.0 deprecates `forward_signature` in favor of `forward_origin`.

Type

`str`

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`.

New in version 20.0.

Type

`int`

property link

Convenience property. If the chat of the message is not a private chat or normal group, returns a t.me link of the message.

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, *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 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\(\)](#).

New 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_to_message_id=None,
    reply_markup=None, allow_sending_without_reply=None,
    caption_entities=None, protect_content=None, message_thread_id=None,
    has_spoiler=None, thumbnail=None, reply_parameters=None, *,
    filename=None, quote=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, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_animation()`.

Keyword Arguments

- **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.

Deprecated since version 20.8: This argument is deprecated in favor of `do_quote`

- **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. Default: `True` in group chats and `False` in private chats. Mutually exclusive with `quote`.

New 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_to_message_id=None, reply_markup=None,
    parse_mode=None, allow_sending_without_reply=None, caption_entities=None,
    protect_content=None, message_thread_id=None, thumbnail=None,
    reply_parameters=None, *, filename=None, quote=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, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_audio()`.

Keyword Arguments

- **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.

Deprecated since version 20.8: This argument is deprecated in favor of `do_quote`

- **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. Default: `True` in group chats and `False` in private chats. Mutually exclusive with `quote`.

New in version 20.8.

Returns

On success, instance representing the message posted.

Return type

`telegram.Message`

```
async def 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, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_chat_action()`.

New in version 13.2.

Returns

On success, `True` is returned.

Return type

`bool`

```
async def reply_contact(phone_number=None, first_name=None, last_name=None,
                       disable_notification=None, reply_to_message_id=None, reply_markup=None,
                       vcard=None, allow_sending_without_reply=None, protect_content=None,
                       message_thread_id=None, reply_parameters=None, *, contact=None,
                       quote=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, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_contact()`.

Keyword Arguments

- **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.

Deprecated since version 20.8: This argument is deprecated in favor of `do_quote`

- **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. Default: `True` in group chats and `False` in private chats. Mutually exclusive with `quote`.

New 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_to_message_id=None,
                  allow_sending_without_reply=None, reply_markup=None, protect_content=None,
                  message_thread_id=None, reply_parameters=None, *, quote=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_id=message_id,
    *args,
    **kwargs
)
```

For the documentation of the arguments, please see `telegram.Bot.copy_message()`.

Keyword Arguments

- **quote** (`bool`, optional) – If set to `True`, the 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.

New in version 13.1.

Deprecated since version 20.8: This argument is deprecated in favor of `do_quote`

- **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. Default: `True` in group chats and `False` in private chats. Mutually exclusive with `quote`.

New 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_to_message_id=None, reply_markup=None,
                 emoji=None, allow_sending_without_reply=None, protect_content=None,
                 message_thread_id=None, reply_parameters=None, *, quote=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, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_dice()`.

Keyword Arguments

- **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.

Deprecated since version 20.8: This argument is deprecated in favor of `do_quote`

- **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. Default: `True` in group chats and `False` in private chats. Mutually exclusive with `quote`.

New 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_to_message_id=None, reply_markup=None, parse_mode=None,
                      disable_content_type_detection=None, allow_sending_without_reply=None,
                      caption_entities=None, protect_content=None, message_thread_id=None,
                      thumbnail=None, reply_parameters=None, *, filename=None, quote=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, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_document()`.

Keyword Arguments

- **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.

Deprecated since version 20.8: This argument is deprecated in favor of `do_quote`

- **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. Default: `True` in group chats and `False` in private chats. Mutually exclusive with `quote`.

New 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_to_message_id=None,
                  reply_markup=None, allow_sending_without_reply=None, protect_content=None,
                  message_thread_id=None, reply_parameters=None, *, quote=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, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_game()`.

Keyword Arguments

- **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.

Deprecated since version 20.8: This argument is deprecated in favor of `do_quote`

- **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. Default: **True** in group chats and **False** in private chats. Mutually exclusive with `quote`.

New in version 20.8.

New in version 13.2.

Returns

On success, instance representing the message posted.

Return type

`telegram.Message`

```
async def reply_html(text, disable_web_page_preview=None, disable_notification=None,
                    reply_to_message_id=None, reply_markup=None,
                    allow_sending_without_reply=None, entities=None, protect_content=None,
                    message_thread_id=None, link_preview_options=None, reply_parameters=None, *,
                    quote=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,
    parse_mode=ParseMode.HTML,
    *args,
    **kwargs,
)
```

Sends a message with HTML formatting.

For the documentation of the arguments, please see `telegram.Bot.send_message()`.

Keyword Arguments

- **quote** (*bool*, optional) – If set to **True**, the 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.

Deprecated since version 20.8: This argument is deprecated in favor of `do_quote`

- **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. Default: **True** in group chats and **False** in private chats. Mutually exclusive with `quote`.

New in version 20.8.

Returns

On success, instance representing the message posted.

Return type

`telegram.Message`

```
async reply_invoice(title, description, payload, provider_token, currency, prices,
                    start_parameter=None, photo_url=None, photo_size=None, photo_width=None,
                    photo_height=None, need_name=None, need_phone_number=None,
                    need_email=None, need_shipping_address=None, is_flexible=None,
                    disable_notification=None, reply_to_message_id=None, reply_markup=None,
                    provider_data=None, send_phone_number_to_provider=None,
                    send_email_to_provider=None, allow_sending_without_reply=None,
                    max_tip_amount=None, suggested_tip_amounts=None, protect_content=None,
                    message_thread_id=None, reply_parameters=None, *, quote=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, *args, **kwargs)
```

For the documentation of the arguments, please see [telegram.Bot.send_invoice\(\)](#).

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

New in version 13.2.

Changed in version 13.5: As of Bot API 5.2, the parameter [start_parameter](#) is optional.

Keyword Arguments

- [quote](#) (`bool`, optional) – If set to `True`, the 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.

Deprecated since version 20.8: This argument is deprecated in favor of [do_quote](#)

- [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. Default: `True` in group chats and `False` in private chats. Mutually exclusive with [quote](#).

New in version 20.8.

Returns

On success, instance representing the message posted.

Return type

[telegram.Message](#)

```
async reply_location(latitude=None, longitude=None, disable_notification=None,
                    reply_to_message_id=None, reply_markup=None, live_period=None,
                    horizontal_accuracy=None, heading=None, proximity_alert_radius=None,
                    allow_sending_without_reply=None, protect_content=None,
                    message_thread_id=None, reply_parameters=None, *, location=None,
                    quote=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, *args, **kwargs)
```

For the documentation of the arguments, please see [telegram.Bot.send_location\(\)](#).

Keyword Arguments

- **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.

Deprecated since version 20.8: This argument is deprecated in favor of `do_quote`

- **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. Default: **True** in group chats and **False** in private chats. Mutually exclusive with `quote`.

New in version 20.8.

Returns

On success, instance representing the message posted.

Return type

`telegram.Message`

```
async def reply_markdown(text, disable_web_page_preview=None, disable_notification=None,
                        reply_to_message_id=None, reply_markup=None,
                        allow_sending_without_reply=None, entities=None, protect_content=None,
                        message_thread_id=None, link_preview_options=None,
                        reply_parameters=None, *, quote=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,
    parse_mode=ParseMode.MARKDOWN,
    *args,
    **kwargs,
)
```

Sends a message with Markdown version 1 formatting.

For the documentation of the arguments, please see `telegram.Bot.send_message()`.

Note: `'Markdown'` is a legacy mode, retained by Telegram for backward compatibility. You should use `reply_markdown_v2()` instead.

Keyword Arguments

- **quote** (*bool*, optional) – If set to **True**, the 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.

Deprecated since version 20.8: This argument is deprecated in favor of `do_quote`

- **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. Default: **True** in group chats and **False** in private chats. Mutually exclusive with `quote`.

New in version 20.8.

Returns

On success, instance representing the message posted.

Return type*telegram.Message*

```
async reply_markdown_v2(text, disable_web_page_preview=None, disable_notification=None,
                        reply_to_message_id=None, reply_markup=None,
                        allow_sending_without_reply=None, entities=None,
                        protect_content=None, message_thread_id=None,
                        link_preview_options=None, reply_parameters=None, *, quote=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,
    parse_mode=ParseMode.MARKDOWN_V2,
    *args,
    **kwargs,
)
```

Sends a message with markdown version 2 formatting.

For the documentation of the arguments, please see [telegram.Bot.send_message\(\)](#).

Keyword Arguments

- **quote** (*bool*, optional) – If set to **True**, the 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.

Deprecated since version 20.8: This argument is deprecated in favor of *do_quote*

- **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. Default: **True** in group chats and **False** in private chats. Mutually exclusive with *quote*.

New in version 20.8.

Returns

On success, instance representing the message posted.

Return type*telegram.Message*

```
async reply_media_group(media, disable_notification=None, reply_to_message_id=None,
                        allow_sending_without_reply=None, protect_content=None,
                        message_thread_id=None, reply_parameters=None, *, quote=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, *args, **kwargs)
```

For the documentation of the arguments, please see [telegram.Bot.send_media_group\(\)](#).

Keyword Arguments

- **quote** (*bool*, optional) – If set to **True**, the 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.

Deprecated since version 20.8: This argument is deprecated in favor of *do_quote*

- **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. Default: `True` in group chats and `False` in private chats. Mutually exclusive with `quote`.

New in version 20.8.

Returns

An array of the sent Messages.

Return type

Tuple[[`telegram.Message`](#)]

Raises

[`telegram.error.TelegramError`](#) –

```
async def reply_photo(photo, caption=None, disable_notification=None, reply_to_message_id=None,
                      reply_markup=None, parse_mode=None, allow_sending_without_reply=None,
                      caption_entities=None, protect_content=None, message_thread_id=None,
                      has_spoiler=None, reply_parameters=None, *, filename=None, quote=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, *args, **kwargs)
```

For the documentation of the arguments, please see [`telegram.Bot.send_photo\(\)`](#).

Keyword Arguments

- **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.

Deprecated since version 20.8: This argument is deprecated in favor of `do_quote`

- **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. Default: `True` in group chats and `False` in private chats. Mutually exclusive with `quote`.

New in version 20.8.

Returns

On success, instance representing the message posted.

Return type

[`telegram.Message`](#)

```
async def reply_poll(question, options, is_anonymous=None, type=None,
                    allows_multiple_answers=None, correct_option_id=None, is_closed=None,
                    disable_notification=None, reply_to_message_id=None, reply_markup=None,
                    explanation=None, explanation_parse_mode=None, open_period=None,
                    close_date=None, allow_sending_without_reply=None, explanation_entities=None,
                    protect_content=None, message_thread_id=None, reply_parameters=None, *,
                    quote=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, *args, **kwargs)
```

For the documentation of the arguments, please see [`telegram.Bot.send_poll\(\)`](#).

Keyword Arguments

- **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.

Deprecated since version 20.8: This argument is deprecated in favor of `do_quote`

- **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. Default: **True** in group chats and **False** in private chats. Mutually exclusive with `quote`.

New in version 20.8.

Returns

On success, instance representing the message posted.

Return type

`telegram.Message`

```
async reply_sticker(sticker, disable_notification=None, reply_to_message_id=None,
                    reply_markup=None, allow_sending_without_reply=None,
                    protect_content=None, message_thread_id=None, emoji=None,
                    reply_parameters=None, *, quote=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, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_sticker()`.

Keyword Arguments

- **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.

Deprecated since version 20.8: This argument is deprecated in favor of `do_quote`

- **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. Default: **True** in group chats and **False** in private chats. Mutually exclusive with `quote`.

New in version 20.8.

Returns

On success, instance representing the message posted.

Return type

`telegram.Message`

```
async reply_text(text, parse_mode=None, disable_web_page_preview=None,
                 disable_notification=None, reply_to_message_id=None, reply_markup=None,
                 allow_sending_without_reply=None, entities=None, protect_content=None,
                 message_thread_id=None, link_preview_options=None, reply_parameters=None, *,
                 quote=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, *args, **kwargs)
```

For the documentation of the arguments, please see [telegram.Bot.send_message\(\)](#).

Keyword Arguments

- **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.

Deprecated since version 20.8: This argument is deprecated in favor of [do_quote](#)

- **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. Default: **True** in group chats and **False** in private chats. Mutually exclusive with [quote](#).

New in version 20.8.

Returns

On success, instance representing the message posted.

Return type

[telegram.Message](#)

```
async bot.reply_venue(latitude=None, longitude=None, title=None, address=None, foursquare_id=None,
                     disable_notification=None, reply_to_message_id=None, reply_markup=None,
                     foursquare_type=None, google_place_id=None, google_place_type=None,
                     allow_sending_without_reply=None, protect_content=None,
                     message_thread_id=None, reply_parameters=None, *, venue=None, quote=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, *args, **kwargs)
```

For the documentation of the arguments, please see [telegram.Bot.send_venue\(\)](#).

Keyword Arguments

- **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.

Deprecated since version 20.8: This argument is deprecated in favor of [do_quote](#)

- **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. Default: **True** in group chats and **False** in private chats. Mutually exclusive with [quote](#).

New 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_to_message_id=None, reply_markup=None, width=None, height=None,
    parse_mode=None, supports_streaming=None,
    allow_sending_without_reply=None, caption_entities=None,
    protect_content=None, message_thread_id=None, has_spoiler=None,
    thumbnail=None, reply_parameters=None, *, filename=None, quote=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, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_video()`.

Keyword Arguments

- **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.

Deprecated since version 20.8: This argument is deprecated in favor of `do_quote`

- **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. Default: `True` in group chats and `False` in private chats. Mutually exclusive with `quote`.

New 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_to_message_id=None, reply_markup=None,
    allow_sending_without_reply=None, protect_content=None,
    message_thread_id=None, thumbnail=None, reply_parameters=None, *,
    filename=None, quote=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, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_video_note()`.

Keyword Arguments

- **quote** (`bool`, optional) – If set to `True`, the 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.

Deprecated since version 20.8: This argument is deprecated in favor of `do_quote`

- **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. Default: `True` in group chats and `False` in private chats. Mutually exclusive with `quote`.

New in version 20.8.

Returns

On success, instance representing the message posted.

Return type

[`telegram.Message`](#)

```
async def reply_voice(voice, duration=None, caption=None, disable_notification=None,
                      reply_to_message_id=None, reply_markup=None, parse_mode=None,
                      allow_sending_without_reply=None, caption_entities=None,
                      protect_content=None, message_thread_id=None, reply_parameters=None, *,
                      filename=None, quote=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, *args, **kwargs)
```

For the documentation of the arguments, please see [`telegram.Bot.send_voice\(\)`](#).

Keyword Arguments

- **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.

Deprecated since version 20.8: This argument is deprecated in favor of [`do_quote`](#)

- **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. Default: `True` in group chats and `False` in private chats. Mutually exclusive with [`quote`](#).

New in version 20.8.

Returns

On success, instance representing the message posted.

Return type

[`telegram.Message`](#)

```
async def 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\(\)`](#).

New 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, *args, **kwargs  
)
```

For the documentation of the arguments, please see [`telegram.Bot.stop_message_live_location\(\)`](#).

Note: You can only edit messages that the bot sent itself (i.e. of the `bot.send_*` family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

Returns

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

Return type

[`telegram.Message`](#)

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

Shortcut for:

```
await bot.stop_poll(  
    chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs  
)
```

For the documentation of the arguments, please see [`telegram.Bot.stop_poll\(\)`](#).

Returns

On success, the stopped Poll with the final results is returned.

Return type

[`telegram.Poll`](#)

property text_html

Creates an HTML-formatted string from the markup entities found in the message.

Use this if you want to retrieve the message text with the entities formatted as HTML in the same way the original message was formatted.

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, *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 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()`.

New in version 20.0.

Returns

On success, `True` is returned.

Return type

`bool`

property user_shared

Optional. Service message. A user was shared with the bot.

Hint: In case a single user was shared, `user_shared` will be present in addition to `users_shared`. If multiple users were shared, only `users_shared` will be present. However, this behavior is not documented and may be changed by Telegram.

New in version 20.1.

Deprecated since version 20.8: Bot API 7.0 deprecates `user_shared` in favor of `users_shared`.

Type

`telegram.UserShared`

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`

New in version 13.4.

Parameters

`message_auto_delete_time` (`int`) – New auto-delete time for messages in the chat.

message_auto_delete_time

New auto-delete time for messages in the chat.

Type

`int`

MessageEntity

```
class telegram.MessageEntity(type, offset, length, url=None, user=None, language=None,
                             custom_emoji_id=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.

Parameters

- **type** (`str`) – Type of the entity. Can be `MENTION` (@username), `HASHTAG` (#hashtag), `CASHTAG` (\$USD), `BOT_COMMAND` (/start@jobs_bot), `URL` (https://telegram.org), `EMAIL` (do-not-reply@telegram.org), `PHONE_NUMBER` (+1-212-555-0123), **BOLD** (**bold text**), *ITALIC* (*italic text*), UNDERLINE (underlined text), ~~STRIKETHROUGH~~, *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).

New in version 20.0: Added inline custom emoji

New in version 20.8: Added block quotation

- **offset** (`int`) – Offset in UTF-16 code units to the start of the entity.
- **length** (`int`) – Length of the entity in UTF-16 code units.
- **url** (`str`, optional) – For `TEXT_LINK` only, url that will be opened after user taps on the text.
- **user** (`telegram.User`, optional) – For `TEXT_MENTION` only, the mentioned user.
- **language** (`str`, optional) – For `PRE` only, the programming language of the entity text.
- **custom_emoji_id** (`str`, optional) – For `CUSTOM_EMOJI` only, unique identifier of the custom emoji. Use `telegram.Bot.get_custom_emoji_stickers()` to get full information about the sticker.

New in version 20.0.

type

Type of the entity. Can be `MENTION` (@username), `HASHTAG` (#hashtag), `CASHTAG` (\$USD), `BOT_COMMAND` (/start@jobs_bot), `URL` (https://telegram.org), `EMAIL` (do-not-reply@telegram.org), `PHONE_NUMBER` (+1-212-555-0123), **BOLD** (**bold text**), *ITALIC* (*italic text*), UNDERLINE (underlined text), ~~STRIKETHROUGH~~, *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).

New in version 20.0: Added inline custom emoji

New in version 20.8: Added block quotation

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.

New in version 20.0.

Type

`str`

Use In

- `telegram.Bot.copy_message()`
 - `telegram.Bot.edit_message_caption()`
 - `telegram.Bot.edit_message_text()`
 - `telegram.Bot.send_animation()`
 - `telegram.Bot.send_audio()`
 - `telegram.Bot.send_document()`
 - `telegram.Bot.send_media_group()`
 - `telegram.Bot.send_message()`
 - `telegram.Bot.send_photo()`
 - `telegram.Bot.send_poll()`
 - `telegram.Bot.send_video()`
 - `telegram.Bot.send_voice()`
-

Available In

- `telegram.Game.text_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.InputMedia.caption_entities`
 - `telegram.InputMediaAnimation.caption_entities`
 - `telegram.InputMediaAudio.caption_entities`
 - `telegram.InputMediaDocument.caption_entities`
 - `telegram.InputMediaPhoto.caption_entities`
 - `telegram.InputMediaVideo.caption_entities`
 - `telegram.InputTextMessageContent.entities`
 - `telegram.Message.caption_entities`
 - `telegram.Message.entities`
 - `telegram.Poll.explanation_entities`
-

```
ALL_TYPES = [MessageEntityType.MENTION, MessageEntityType.HASHTAG,
MessageEntityType.CASHTAG, MessageEntityType.PHONE_NUMBER,
MessageEntityType.BOT_COMMAND, MessageEntityType.URL, MessageEntityType.EMAIL,
MessageEntityType.BOLD, MessageEntityType.ITALIC, MessageEntityType.CODE,
MessageEntityType.PRE, MessageEntityType.TEXT_LINK,
MessageEntityType.TEXT_MENTION, MessageEntityType.UNDERLINE,
MessageEntityType.STRIKETHROUGH, MessageEntityType.SPOILER,
MessageEntityType.CUSTOM_EMOJI, MessageEntityType.BLOCKQUOTE]
```

A list of all available message entity types.

Type

List[str]

BLOCKQUOTE = 'blockquote'

`telegram.constants.MessageEntityType.BLOCKQUOTE`

New 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
    New in version 20.0.
EMAIL = 'email'
    telegram.constants.MessageEntityType.EMAIL
HASHTAG = 'hashtag'
    telegram.constants.MessageEntityType.HASHTAG
ITALIC = 'italic'
    telegram.constants.MessageEntityType.ITALIC
MENTION = 'mention'
    telegram.constants.MessageEntityType.MENTION
PHONE_NUMBER = 'phone_number'
    telegram.constants.MessageEntityType.PHONE_NUMBER
PRE = 'pre'
    telegram.constants.MessageEntityType.PRE
SPOILER = 'spoiler'
    telegram.constants.MessageEntityType.SPOILER
    New in version 13.10.
STRIKETHROUGH = 'strikethrough'
    telegram.constants.MessageEntityType.STRIKETHROUGH
TEXT_LINK = 'text_link'
    telegram.constants.MessageEntityType.TEXT_LINK
TEXT_MENTION = 'text_mention'
    telegram.constants.MessageEntityType.TEXT_MENTION
UNDERLINE = 'underline'
    telegram.constants.MessageEntityType.UNDERLINE
URL = 'url'
    telegram.constants.MessageEntityType.URL
classmethod de_json(data, bot)
    See telegram.TelegramObject.de_json().
```

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.

Parameters

message_id (*int*) – Unique message identifier.

message_id

Unique message identifier.

Type

int

Returned In

[telegram.Bot.copy_message\(\)](#)

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](#)
-

New 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)`

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`
-

New 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`
-

New 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](#)
-

New 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](#)

New 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`

New 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)`

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`

New 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)

See `telegram.TelegramObject.de_json()`.

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.

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`

Use In

- `telegram.Bot.get_file()`
 - `telegram.Bot.send_photo()`
-

Available In

- `telegram.Animation.thumbnail`
 - `telegram.Audio.thumbnail`
 - `telegram.Document.thumbnail`
 - `telegram.ExternalReplyInfo.photo`
 - `telegram.Game.photo`
 - `telegram.Message.new_chat_photo`
 - `telegram.Message.photo`
 - `telegram.Sticker.thumbnail`
 - `telegram.StickerSet.thumbnail`
 - `telegram.UserProfilePhotos.photos`
 - `telegram.Video.thumbnail`
 - `telegram.VideoNote.thumbnail`
-

```
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, *,
                   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

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.
- **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.
- **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.

- **`open_period`** (`int`, optional) – Amount of time in seconds the poll will be active after creation.
- **`close_date`** (`datetime.datetime`, optional) – Point in time (Unix timestamp) when the poll will be automatically closed. Converted to `datetime.datetime`.

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.

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`

correct_option_id

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.

Type

`int`

explanation

Optional. Text that is shown when a user chooses an incorrect answer or taps on the lamp icon in a quiz-style poll, 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*]

open_period

Optional. Amount of time in seconds the poll will be active after creation.

Type

`int`

close_date

Optional. Point in time when the poll will be automatically closed.

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`

Available In

- *telegram.ExternalReplyInfo.poll*
 - *telegram.Message.poll*
 - *telegram.Update.poll*
-

Returned In

telegram.Bot.stop_poll()

MAX_EXPLANATION_LENGTH = 200

telegram.constants.PollLimit.MAX_EXPLANATION_LENGTH

New in version 20.0.

MAX_EXPLANATION_LINE_FEEDS = 2

telegram.constants.PollLimit.MAX_EXPLANATION_LINE_FEEDS

New in version 20.0.

MAX_OPEN_PERIOD = 600

telegram.constants.PollLimit.MAX_OPEN_PERIOD

New in version 20.0.

MAX_OPTION_LENGTH = 100

telegram.constants.PollLimit.MAX_OPTION_LENGTH

New in version 20.0.

MAX_OPTION_NUMBER = 10

`telegram.constants.PollLimit.MAX_OPTION_NUMBER`

New in version 20.0.

MAX_QUESTION_LENGTH = 300

`telegram.constants.PollLimit.MAX_QUESTION_LENGTH`

New in version 20.0.

MIN_OPEN_PERIOD = 5

`telegram.constants.PollLimit.MIN_OPEN_PERIOD`

New in version 20.0.

MIN_OPTION_LENGTH = 1

`telegram.constants.PollLimit.MIN_OPTION_LENGTH`

New in version 20.0.

MIN_OPTION_NUMBER = 2

`telegram.constants.PollLimit.MIN_OPTION_NUMBER`

New in version 20.0.

MIN_QUESTION_LENGTH = 1

`telegram.constants.PollLimit.MIN_QUESTION_LENGTH`

New in version 20.0.

QUIZ = 'quiz'

`telegram.constants.PollType.QUIZ`

REGULAR = 'regular'

`telegram.constants.PollType.REGULAR`

classmethod de_json(data, bot)

See `telegram.TelegramObject.de_json()`.

parse_explanation_entities(types=None)

Returns a `dict` that maps `telegram.MessageEntity` to `str`. It contains entities from this poll's explanation filtered by their `type` attribute as the key, and the text that each entity belongs to as the value of the `dict`.

Note: This method should always be used instead of the `explanation_entities` attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See `parse_explanation_entity` for more info.

Parameters

types (List[str], optional) – List of `MessageEntity` types as strings. If the `type` attribute of an entity is contained in this list, it will be returned. Defaults to `telegram.MessageEntity.ALL_TYPES`.

Returns

A dictionary of entities mapped to the text that belongs to them, calculated based on UTF-16 codepoints.

Return type

Dict[`telegram.MessageEntity`, str]

parse_explanation_entity(*entity*)

Returns the text from a given *telegram.MessageEntity*.

Note: This method is present because Telegram calculates the offset and length in UTF-16 codepoint pairs, which some versions of Python don't handle automatically. (That is, you can't just slice `Message.text` with the offset and length.)

Parameters

entity (*telegram.MessageEntity*) – The entity to extract the text from. It must be an entity that belongs to this message.

Returns

The text of the given entity.

Return type

`str`

Raises

RuntimeError – If the poll has no explanation.

PollAnswer

class telegram.**PollAnswer**(*poll_id*, *option_ids*, *user*=None, *voter_chat*=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 compatibility 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.

- *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.

New 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`]

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.

New in version 20.5.

Type

`telegram.Chat`

classmethod de_json(data, bot)

See `telegram.TelegramObject.de_json()`.

PollOption

class telegram.PollOption(text, voter_count, *, 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` and `voter_count` are equal.

Parameters

- **text** (`str`) – Option text, *1-100* characters.
- **voter_count** (`int`) – Number of users that voted for this option.

text

Option text, *1-100* characters.

Type

`str`

voter_count

Number of users that voted for this option.

Type

`int`

Available In

`telegram.Poll.options`

MAX_LENGTH = 100

telegram.constants.PollLimit.MAX_OPTION_LENGTH

New in version 20.0.

MIN_LENGTH = 1

telegram.constants.PollLimit.MIN_OPTION_LENGTH

New in version 20.0.

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.

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

Available In

telegram.Message.proximity_alert_triggered

classmethod **de_json**(*data, bot*)

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](#)

New 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*)

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](#) and [telegram.ReactionTypeCustomEmoji](#).

Use In

[telegram.Bot.set_message_reaction\(\)](#)

Available In

- [telegram.Chat.available_reactions](#)
 - [telegram.MessageReactionUpdated.new_reaction](#)
 - [telegram.MessageReactionUpdated.old_reaction](#)
 - [telegram.ReactionCount.type](#)
-

New in version 20.8.

Parameters

type (*str*) – Type of the reaction. Can be *EMOJI* or *CUSTOM_EMOJI*.

type

Type of the reaction. Can be *EMOJI* or *CUSTOM_EMOJI*.

Type

str

CUSTOM_EMOJI = 'custom_emoji'

telegram.constants.ReactionType.CUSTOM_EMOJI

EMOJI = 'emoji'

telegram.constants.ReactionType.EMOJI

classmethod **de_json**(*data*, *bot*)

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.Chat.available_reactions*
 - *telegram.MessageReactionUpdated.new_reaction*
 - *telegram.MessageReactionUpdated.old_reaction*
 - *telegram.ReactionCount.type*
-

New 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.Chat.available_reactions](#)
 - [telegram.MessageReactionUpdated.new_reaction](#)
 - [telegram.MessageReactionUpdated.old_reaction](#)
 - [telegram.ReactionCount.type](#)
-

New 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``.

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.

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.

Use In

- [telegram.Bot.copy_message\(\)](#)
- [telegram.Bot.send_animation\(\)](#)
- [telegram.Bot.send_audio\(\)](#)

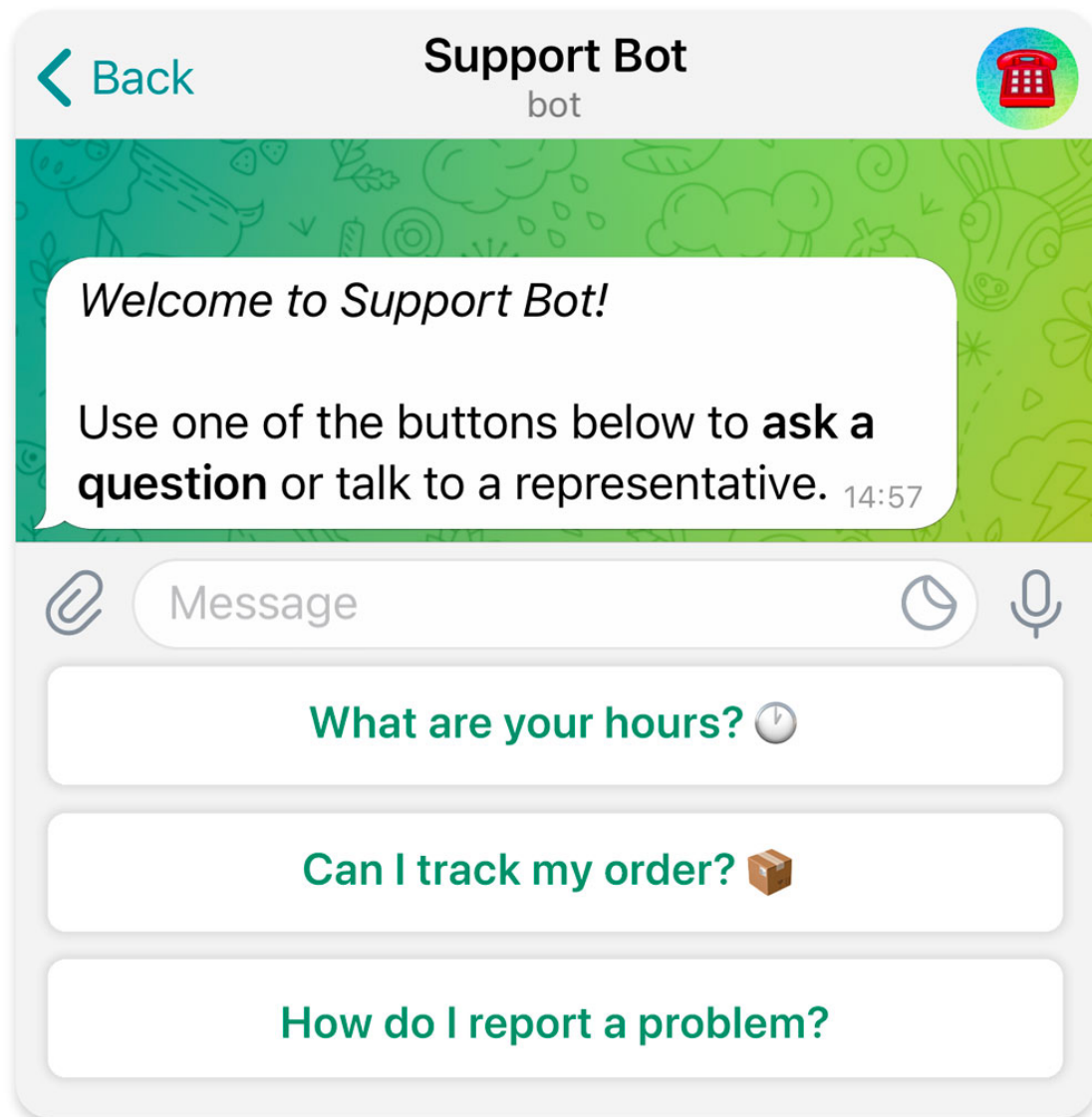


Fig. 2: A reply keyboard with reply options.

- `telegram.Bot.send_contact()`
 - `telegram.Bot.send_dice()`
 - `telegram.Bot.send_document()`
 - `telegram.Bot.send_location()`
 - `telegram.Bot.send_message()`
 - `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()`
-

See also:

An 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*
-

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.
New 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.

New 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.

New 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.

New in version 20.0.

Type

`bool`

MAX_INPUT_FIELD_PLACEHOLDER = 64

`telegram.constants.ReplyLimit.MAX_INPUT_FIELD_PLACEHOLDER`

New in version 20.0.

MIN_INPUT_FIELD_PLACEHOLDER = 1

`telegram.constants.ReplyLimit.MIN_INPUT_FIELD_PLACEHOLDER`

New 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.

New 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.

New 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.

New 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.

New 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.

New 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.

New 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`](#)).

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](#)
-

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`

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_location()`
 - `telegram.Bot.send_message()`
 - `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()`
-

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*, *, *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_contact()`
- `telegram.Bot.send_dice()`
- `telegram.Bot.send_document()`
- `telegram.Bot.send_game()`
- `telegram.Bot.send_invoice()`
- `telegram.Bot.send_location()`
- `telegram.Bot.send_media_group()`
- `telegram.Bot.send_message()`
- `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()`
-

New in version 20.8.

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`).
- **`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, and custom_emoji 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.

`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`).

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, and custom_emoji 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`

classmethod `de_json(data, bot)`

See [telegram.TelegramObject.de_json](#).

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\(\)](#)

New in version 20.0.

Parameters

inline_message_id (`str`, optional) – Identifier of the sent inline message. Available only if there is an *inline keyboard* attached to the message.

inline_message_id

Optional. Identifier of the sent inline message. Available only if there is an *inline keyboard* attached to the message.

Type

`str`

Story

class telegram.Story(*, api_kwargs=None)

Bases: [telegram.TelegramObject](#)

This object represents a message about a forwarded story in the chat. Currently holds no information.

Available In

- [telegram.ExternalReplyInfo.story](#)
 - [telegram.Message.story](#)
-

New in version 20.5.

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.

New 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.

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.

New 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.

New 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)`

Converts JSON data to a Telegram object.

Parameters

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

Returns

The Telegram object.

classmethod **de_list**(data, bot)

Converts 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*) – The bot associated with these objects.

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`.

New 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

New 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, and custom_emoji 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, and custom_emoji 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)`

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, *, 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](#)

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.

New in version 13.4.

- **`chat_member`** (`telegram.ChatMemberUpdated`, optional) – A chat member’s status was updated in a chat. The bot must be an administrator in the chat and must explicitly specify `CHAT_MEMBER` in the list of `telegram.ext.Application.run_polling.allowed_updates` to receive these updates (see `telegram.Bot.get_updates()`, `telegram.Bot.set_webhook()`, `telegram.ext.Application.run_polling()` and `telegram.ext.Application.run_webhook()`).

New in version 13.4.

- **`chat_join_request`** (`telegram.ChatJoinRequest`, optional) – A request to join the chat has been sent. The bot must have the `telegram.ChatPermissions.can_invite_users` administrator right in the chat to receive these updates.

New in version 13.8.

- **`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.

New 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.

New 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.

New 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.

New in version 20.8.

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.

New in version 13.4.

Type

telegram.ChatMemberUpdated

chat_member

Optional. A chat member's status was updated in a chat. The bot must be an administrator in the chat and must explicitly specify `CHAT_MEMBER` in the list of *telegram.ext.Application.run_polling.allowed_updates* to receive these updates (see *telegram.Bot.get_updates()*, *telegram.Bot.set_webhook()*, *telegram.ext.Application.run_polling()* and *telegram.ext.Application.run_webhook()*).

New in version 13.4.

Type

telegram.ChatMemberUpdated

chat_join_request

Optional. A request to join the chat has been sent. The bot must have the *telegram.ChatPermissions.can_invite_users* administrator right in the chat to receive these updates.

New in version 13.8.

Type

telegram.ChatJoinRequest

chat_boost

Optional. A chat boost was added or changed. The bot must be an administrator in the chat to receive these updates.

New 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.

New 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.

New 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.

New in version 20.8.

Type

`telegram.MessageReactionCountUpdated`

`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]`

A list of all available update types.

New in version 13.5.

Type

`List[str]`

`CALLBACK_QUERY = 'callback_query'`

`telegram.constants.UpdateType.CALLBACK_QUERY`

New in version 13.5.

`CHANNEL_POST = 'channel_post'`

`telegram.constants.UpdateType.CHANNEL_POST`

New in version 13.5.

`CHAT_BOOST = 'chat_boost'`

`telegram.constants.UpdateType.CHAT_BOOST`

New in version 20.8.

`CHAT_JOIN_REQUEST = 'chat_join_request'`

`telegram.constants.UpdateType.CHAT_JOIN_REQUEST`

New in version 13.8.

`CHAT_MEMBER = 'chat_member'`

`telegram.constants.UpdateType.CHAT_MEMBER`

New in version 13.5.

`CHOSEN_INLINE_RESULT = 'chosen_inline_result'`

`telegram.constants.UpdateType.CHOSEN_INLINE_RESULT`

New in version 13.5.

EDITED_CHANNEL_POST = 'edited_channel_post'

telegram.constants.UpdateType.EDITED_CHANNEL_POST

New in version 13.5.

EDITED_MESSAGE = 'edited_message'

telegram.constants.UpdateType.EDITED_MESSAGE

New in version 13.5.

INLINE_QUERY = 'inline_query'

telegram.constants.UpdateType.INLINE_QUERY

New in version 13.5.

MESSAGE = 'message'

telegram.constants.UpdateType.MESSAGE

New in version 13.5.

MESSAGE_REACTION = 'message_reaction'

telegram.constants.UpdateType.MESSAGE_REACTION

New in version 20.8.

MESSAGE_REACTION_COUNT = 'message_reaction_count'

telegram.constants.UpdateType.MESSAGE_REACTION_COUNT

New in version 20.8.

MY_CHAT_MEMBER = 'my_chat_member'

telegram.constants.UpdateType.MY_CHAT_MEMBER

New in version 13.5.

POLL = 'poll'

telegram.constants.UpdateType.POLL

New in version 13.5.

POLL_ANSWER = 'poll_answer'

telegram.constants.UpdateType.POLL_ANSWER

New in version 13.5.

PRE_CHECKOUT_QUERY = 'pre_checkout_query'

telegram.constants.UpdateType.PRE_CHECKOUT_QUERY

New in version 13.5.

REMOVED_CHAT_BOOST = 'removed_chat_boost'

telegram.constants.UpdateType.REMOVED_CHAT_BOOST

New in version 20.8.

SHIPPING_QUERY = 'shipping_query'

telegram.constants.UpdateType.SHIPPING_QUERY

New in version 13.5.

classmethod de_json(data, bot)

See *telegram.TelegramObject.de_json()*.

property effective_chat

The chat that this update was sent in, no matter what kind of update this is. If no chat is associated with this update, this gives `None`. This is the case, if `inline_query`, `chosen_inline_result`, `callback_query` from inline messages, `shipping_query`, `pre_checkout_query`, `poll` or `poll_answer` is present.

Example

If `message` is present, this will give `telegram.Message.chat`.

Type

`telegram.Chat`

property effective_message**The message included in this update, no matter what kind of**

update this is. More precisely, this will be the message contained in `message`, `edited_message`, `channel_post`, `edited_channel_post` or `callback_query` (i.e. `telegram.CallbackQuery.message`) or `None`, if none of those are present.

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_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

- `channel_post`
- `edited_channel_post`
- `poll`
- `chat_boost`
- `removed_chat_boost`
- `message_reaction_count`

is present.

Example

- If `message` is present, this will give `telegram.Message.from_user`.
 - If `poll_answer` is present, this will give `telegram.PollAnswer.user`.
-

Type

`telegram.User`

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, *, 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.Bot.bot`
- `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.ChosenInlineResult.from_user`
- `telegram.GameHighScore.user`
- `telegram.GiveawayWinners.winners`
- `telegram.InlineQuery.from_user`
- `telegram.Message.forward_from`
- `telegram.Message.forward_sender_name`
- `telegram.Message.from_user`
- `telegram.Message.left_chat_member`
- `telegram.Message.new_chat_members`
- `telegram.Message.via_bot`
- `telegram.MessageEntity.user`
- `telegram.MessageOriginUser.sender_user`
- `telegram.MessageReactionUpdated.user`
- `telegram.PollAnswer.user`

- `telegram.PreCheckoutQuery.from_user`
 - `telegram.ProximityAlertTriggered.traveler`
 - `telegram.ProximityAlertTriggered.watcher`
 - `telegram.ShippingQuery.from_user`
 - `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` requests.
 - **`can_read_all_group_messages`** (`str`, optional) – `True`, if privacy mode is disabled for the bot. Returned only in `telegram.Bot.get_me` requests.
 - **`supports_inline_queries`** (`str`, optional) – `True`, if the bot supports inline queries. Returned only in `telegram.Bot.get_me` requests.
 - **`is_premium`** (`bool`, optional) – `True`, if this user is a Telegram Premium user.
- New in version 20.0.
- **`added_to_attachment_menu`** (`bool`, optional) – `True`, if this user added the bot to the attachment menu.

New in version 20.0.

`id`

Unique identifier for this user or bot.

Type

`int`

`is_bot`

`True`, if this user is a bot.

Type

`bool`

`first_name`

User’s or bot’s first name.

Type

`str`

last_name

Optional. User's or bot's last name.

Type

`str`

username

Optional. User's or bot's username.

Type

`str`

language_code

Optional. IETF language tag of the user's language.

Type

`str`

can_join_groups

Optional. `True`, if the bot can be invited to groups. Returned only in `telegram.Bot.get_me` requests.

Type

`str`

can_read_all_group_messages

Optional. `True`, if privacy mode is disabled for the bot. Returned only in `telegram.Bot.get_me` requests.

Type

`str`

supports_inline_queries

Optional. `True`, if the bot supports inline queries. Returned only in `telegram.Bot.get_me` requests.

Type

`str`

is_premium

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

New in version 20.0.

Type

`bool`

added_to_attachment_menu

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

New in version 20.0.

Type

`bool`

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

Shortcut for:

```
await bot.approve_chat_join_request(user_id=update.effective_user.id, *args,  
                                     **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.approve_chat_join_request()`.

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.

New in version 13.8.

Returns

On success, `True` is returned.

Return type

`bool`

```
async copy_message(chat_id, message_id, caption=None, parse_mode=None, caption_entities=None,
                  disable_notification=None, reply_to_message_id=None,
                  allow_sending_without_reply=None, reply_markup=None,
                  protect_content=None, message_thread_id=None, reply_parameters=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, instance representing the message posted.

Return type

`telegram.Message`

```
async copy_messages(chat_id, message_ids, disable_notification=None, protect_content=None,
                  message_thread_id=None, remove_caption=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, *args,
↳ **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.copy_messages()`.

See also:

`copy_message()`, `send_copy()`, `send_copies()`.

New in version 20.8.

Returns

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

Return type

Tuple[`telegram.MessageId`]

```
async def 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.

New in version 13.8.

Returns

On success, `True` is returned.

Return type

`bool`

```
async def 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\(\)](#).

New in version 20.8.

Returns

On success, `True` is returned.

Return type

`bool`

```
async def 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\(\)](#).

New in version 20.8.

Returns

On success, `True` is returned.

Return type

`bool`

```
async def forward_from(from_chat_id, message_id, disable_notification=None, protect_content=None,
                      message_thread_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\(\)`](#)

New 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, *,
                             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\(\)`](#).

New 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, *,
                             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\(\)`](#).

New 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, *, 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, *args,  
↪ **kwargs)
```

For the documentation of the arguments, please see [telegram.Bot.forward_message\(\)](#).

See also:

[forward_from\(\)](#), [forward_messages_from\(\)](#), [forward_messages_to\(\)](#)

New 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\(\)](#).

New in version 20.8.

Returns

On success, returns the boosts applied by the user.

Return type

[telegram.UserChatBoosts](#)

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.

New in version 20.0.

Returns

On success, the current menu button is returned.

Return type

[telegram.MenuButton](#)

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

Shortcut for:

```
await bot.get_user_profile_photos(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.get_user_profile_photos()`.

property link

Convenience property. If `username` is available, returns a t.me link of the user.

Type

`str`

mention_button(*name=None*)

Shortcut for:

```
InlineKeyboardButton(text=name, url=f"tg://user?id={update.effective_user.id}"  
↪")
```

New in version 13.9.

Parameters

name (`str`) – The name used as a link for the user. Defaults to `full_name`.

Returns

`InlineButton` with url set to the user mention

Return type

`telegram.InlineKeyboardButton`

mention_html(*name=None*)

Parameters

name (`str`) – The name used as a link for the user. Defaults to `full_name`.

Returns

The inline mention for the user as HTML.

Return type

`str`

mention_markdown(*name=None*)

Note: `'Markdown'` is a legacy mode, retained by Telegram for backward compatibility. You should use `mention_markdown_v2()` instead.

Parameters

name (`str`) – The name used as a link for the user. Defaults to `full_name`.

Returns

The inline mention for the user as markdown (version 1).

Return type

`str`

mention_markdown_v2(*name=None*)

Parameters

name (`str`) – The name used as a link for the user. Defaults to `full_name`.

Returns

The inline mention for the user as markdown (version 2).

Return type`str`**property name**

Convenience property. If available, returns the user's `username` prefixed with "@". If `username` is not available, returns `full_name`.

Type`str`

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

Shortcut for:

```
await bot.pin_chat_message(chat_id=update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.pin_chat_message()`.

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 `send_action`(*action*, *message_thread_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_to_message_id=None*, *reply_markup=None*, *allow_sending_without_reply=None*, *caption_entities=None*, *protect_content=None*, *message_thread_id=None*, *has_spoiler=None*, *thumbnail=None*, *reply_parameters=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_to_message_id=None, reply_markup=None,  
                  parse_mode=None, allow_sending_without_reply=None, caption_entities=None,  
                  protect_content=None, message_thread_id=None, thumbnail=None,  
                  reply_parameters=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, *, 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_to_message_id=None, reply_markup=None,  
                    vcard=None, allow_sending_without_reply=None, protect_content=None,  
                    message_thread_id=None, reply_parameters=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, *, 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\(\)](#).

New 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_to_message_id=None,
                allow_sending_without_reply=None, reply_markup=None, protect_content=None,
                message_thread_id=None, reply_parameters=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_to_message_id=None, reply_markup=None,
                emoji=None, allow_sending_without_reply=None, protect_content=None,
                message_thread_id=None, reply_parameters=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_to_message_id=None, reply_markup=None, parse_mode=None,
                    disable_content_type_detection=None, allow_sending_without_reply=None,
                    caption_entities=None, protect_content=None, message_thread_id=None,
                    thumbnail=None, reply_parameters=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_to_message_id=None,
                reply_markup=None, allow_sending_without_reply=None, protect_content=None,
                message_thread_id=None, reply_parameters=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_invoice(title, description, payload, provider_token, currency, prices,
                   start_parameter=None, photo_url=None, photo_size=None, photo_width=None,
                   photo_height=None, need_name=None, need_phone_number=None,
                   need_email=None, need_shipping_address=None, is_flexible=None,
                   disable_notification=None, reply_to_message_id=None, reply_markup=None,
                   provider_data=None, send_phone_number_to_provider=None,
                   send_email_to_provider=None, allow_sending_without_reply=None,
                   max_tip_amount=None, suggested_tip_amounts=None, protect_content=None,
                   message_thread_id=None, reply_parameters=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_location(latitude=None, longitude=None, disable_notification=None,
                   reply_to_message_id=None, reply_markup=None, live_period=None,
                   horizontal_accuracy=None, heading=None, proximity_alert_radius=None,
                   allow_sending_without_reply=None, protect_content=None,
                   message_thread_id=None, reply_parameters=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, reply_to_message_id=None,
                       allow_sending_without_reply=None, protect_content=None,
                       message_thread_id=None, reply_parameters=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_web_page_preview=None,
                   disable_notification=None, reply_to_message_id=None, reply_markup=None,
                   allow_sending_without_reply=None, entities=None, protect_content=None,
                   message_thread_id=None, link_preview_options=None, reply_parameters=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_photo(photo, caption=None, disable_notification=None, reply_to_message_id=None,
                  reply_markup=None, parse_mode=None, allow_sending_without_reply=None,
                  caption_entities=None, protect_content=None, message_thread_id=None,
                  has_spoiler=None, reply_parameters=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_to_message_id=None, reply_markup=None, explanation=None,
                explanation_parse_mode=None, open_period=None, close_date=None,
                allow_sending_without_reply=None, explanation_entities=None,
                protect_content=None, message_thread_id=None, reply_parameters=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_to_message_id=None,
                  reply_markup=None, allow_sending_without_reply=None,
                  protect_content=None, message_thread_id=None, emoji=None,
                  reply_parameters=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_to_message_id=None, reply_markup=None,
                 foursquare_type=None, google_place_id=None, google_place_type=None,
                 allow_sending_without_reply=None, protect_content=None,
                 message_thread_id=None, reply_parameters=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_to_message_id=None, reply_markup=None, width=None, height=None,
                 parse_mode=None, supports_streaming=None,
                 allow_sending_without_reply=None, caption_entities=None,
                 protect_content=None, message_thread_id=None, has_spoiler=None,
                 thumbnail=None, reply_parameters=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_to_message_id=None, reply_markup=None,
                      allow_sending_without_reply=None, protect_content=None,
                      message_thread_id=None, thumbnail=None, reply_parameters=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_to_message_id=None, reply_markup=None, parse_mode=None,
                 allow_sending_without_reply=None, caption_entities=None,
                 protect_content=None, message_thread_id=None, reply_parameters=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_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, *args,
                               ↪ **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.

New in version 20.0.

Returns

On success, `True` is returned.

Return type

`bool`

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

Shortcut for:

```
await bot.unpin_all_chat_messages(chat_id=update.effective_user.id, *args,
    ↪ **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.unpin_all_chat_messages()`.

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, *, 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`

UserChatBoosts

New 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\(\)](#)

New 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*)

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.

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](#)]]

Returned In

`telegram.Bot.get_user_profile_photos()`

classmethod `de_json(data, bot)`

See `telegram.TelegramObject.de_json()`.

UserShared

class `telegram.UserShared(request_id, user_id, *, api_kwargs=None)`

Bases: `telegram.UsersShared`

Alias for `UsersShared`, kept for backward compatibility.

Available In

- `telegram.Message.user_shared`
 - `telegram.Message.users_shared`
-

New in version 20.1.

Deprecated since version 20.8: Use `UsersShared` instead.

property `user_id`

Alias for the first entry of `UsersShared.user_ids`.

Deprecated since version 20.8: Bot API 7.0 deprecates this attribute in favor of `UsersShared.user_ids`.

UsersShared

class `telegram.UsersShared(request_id, user_ids, *, 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 `user_ids` are equal.

Available In

`telegram.Message.users_shared`

New 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.

Parameters

- **`request_id`** (`int`) – Identifier of the request.
- **`user_ids`** (`Sequence[int]`) – Identifiers of the shared users. These numbers may have more than 32 significant bits and some programming languages may have difficulty/silent defects in interpreting them. But they have at most 52 significant bits, so 64-bit integers or double-precision float types are safe for storing these identifiers. The bot may not have access to the users and could be unable to use these identifiers, unless the users are already known to the bot by some other means.

request_id

Identifier of the request.

Type

`int`

user_ids

Identifiers of the shared users. These numbers may have more than 32 significant bits and some programming languages may have difficulty/silent defects in interpreting them. But they have at most 52 significant bits, so 64-bit integers or double-precision float types are safe for storing these identifiers. The bot may not have access to the users and could be unable to use these identifiers, unless the users are already known to the bot by some other means.

Type

Tuple[`int`]

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.

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`

Use In

`telegram.Bot.send_venue()`

Available In

- `telegram.ExternalReplyInfo.venue`
 - `telegram.Message.venue`
-

classmethod `de_json(data, bot)`

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, *, 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.Message.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 sender.
- **`height`** (`int`) – Video height as defined by sender.
- **`duration`** (`int`) – Duration of the video in seconds as defined by sender.
- **`file_name`** (`str`, optional) – Original filename as defined by sender.
- **`mime_type`** (`str`, optional) – MIME type of a file as defined by sender.
- **`file_size`** (`int`, optional) – File size in bytes.
- **`thumbnail`** (`telegram.PhotoSize`, optional) – Video thumbnail.

New 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 sender.

Type

`int`

`height`

Video height as defined by sender.

Type

`int`

`duration`

Duration of the video in seconds as defined by sender.

Type

`int`

`file_name`

Optional. Original filename as defined by sender.

Type`str`**mime_type**

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

Type`str`**file_size**

Optional. File size in bytes.

Type`int`**thumbnail**

Optional. Video thumbnail.

New in version 20.2.

Type`telegram.PhotoSize`**classmethod** `de_json(data, bot)`

See `telegram.TelegramObject.de_json()`.

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

Convenience wrapper over `telegram.Bot.get_file()`

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

Returns`telegram.File`**Raises**`telegram.error.TelegramError` –

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`

New in version 13.4.

Changed in version 20.0: This class was renamed from `VoiceChatEnded` in accordance to Bot API 6.0.

Parameters

`duration` (`int`) – Voice chat duration in seconds.

duration

Voice chat duration in seconds.

Type`int`

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](#)

New in version 13.4.

Changed in version 20.0: This class was renamed from `VoiceChatParticipantsInvited` in accordance to Bot API 6.0.

Parameters

[users](#) (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*)

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)`

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`

New 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.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`) – Duration of the video in seconds as defined by sender.
- **file_size** (`int`, optional) – File size in bytes.
- **thumbnail** (`telegram.PhotoSize`, optional) – Video thumbnail.

New 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 sender.

Type

`int`

file_size

Optional. File size in bytes.

Type

`int`

thumbnail

Optional. Video thumbnail.

New in version 20.2.

Type

`telegram.PhotoSize`

classmethod de_json(data, bot)

See `telegram.TelegramObject.de_json()`.

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

Convenience wrapper over `telegram.Bot.get_file()`

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

Returns

`telegram.File`

Raises

`telegram.error.TelegramError` –

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.

Parameters

- `file_id` (`str`) – Identifier for this file, which can be used to download or reuse the file.
- `file_unique_id` (`str`) – Unique identifier for this file, which is supposed to be the same over time and for different bots. Can't be used to download or reuse the file.
- `duration` (`int`) – Duration of the audio in seconds as defined by sender.
- `mime_type` (`str`, optional) – MIME type of the file as defined by 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 sender.

Type

`int`

`mime_type`

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

Type

`str`

`file_size`

Optional. File size in bytes.

Type

`int`

Use In

- `telegram.Bot.get_file()`
 - `telegram.Bot.send_voice()`
-

Available In

- `telegram.ExternalReplyInfo.voice`

- `telegram.Message.voice`
-

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`

New in version 20.0.

Parameters

- ***data*** (`str`) – The data. Be aware that a bad client can send arbitrary data in this field.
- ***button_text*** (`str`) – Text of the [web_app](#) keyboard button, from which the Web App was opened.

data

The data. Be aware that a bad client can send arbitrary data in this field.

Type

`str`

button_text

Text of the [web_app](#) keyboard button, from which the Web App was opened.

Warning: Be aware that a bad client can send arbitrary data in this field.

Type

`str`

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.InlineQueryResultsButton.web_app](#)
 - [telegram.KeyboardButton.web_app](#)
 - [telegram.MenuButtonWebApp.web_app](#)
-

New in version 20.0.

Parameters

url ([str](#)) – An HTTPS URL of a Web App to be opened with additional data as specified in [Initializing Web Apps](#).

url

An HTTPS URL of a Web App to be opened with additional data as specified in [Initializing Web Apps](#).

Type

[str](#)

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 list of update types the bot is subscribed to. Defaults to all update types, except `telegram.Update.chat_member`.

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.

New 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 list of update types the bot is subscribed to. Defaults to all update types, except `telegram.Update.chat_member`.

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.

New 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)

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`

New 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.

New 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`.

New 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.

New 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.

New 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.

New 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.

New in version 20.6.

Type

`bool`

Stickers

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.

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`

Use In`telegram.Bot.set_sticker_mask_position()`

Available In`telegram.Sticker.mask_position`

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.get_file()`
 - `telegram.Bot.send_sticker()`
-

Available In

- `telegram.ExternalReplyInfo.sticker`
 - `telegram.Message.sticker`
 - `telegram.StickerSet.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.

New in version 13.11.

- **`type`** (`str`) – Type of the sticker. Currently one of `REGULAR`, `MASK`, `CUSTOM_EMOJI`. The type of the sticker is independent from its format, which is determined by the fields `is_animated` and `is_video`.

New in version 20.0.

- **`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.

New in version 20.0.

- **`custom_emoji_id`** (`str`, optional) – For custom emoji stickers, unique identifier of the custom emoji.

New in version 20.0.

- **thumbnail** (*telegram.PhotoSize*, optional) – Sticker thumbnail in the .WEBP or .JPG format.

New 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.

New 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.

New in version 13.11.

Type

bool

type

Type of the sticker. Currently one of *REGULAR*, *MASK*, *CUSTOM_EMOJI*. The type of the sticker is independent from its format, which is determined by the fields *is_animated* and *is_video*.

New in version 20.0.

Type

str

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.

New in version 20.0.

Type

telegram.File

custom_emoji_id

Optional. For custom emoji stickers, unique identifier of the custom emoji.

New in version 20.0.

Type

str

thumbnail

Optional. Sticker thumbnail in the `.WEBP` or `.JPG` format.

New 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.

New 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*)

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, is_animated, stickers, is_video, 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 20.5: Removed the deprecated argument and attribute `thumb`.

Parameters

- **`name` (str)** – Sticker set name.
- **`title` (str)** – Sticker set title.
- **`is_animated` (bool)** – `True`, if the sticker set contains animated stickers.
- **`is_video` (bool)** – `True`, if the sticker set contains video stickers.

New in version 13.11.

- **`stickers`** (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`.

New in version 20.0.

- **`thumbnail` (`telegram.PhotoSize`, optional)** – Sticker set thumbnail in the `.WEBP`, `.TGS`, or `.WEBM` format.

New in version 20.2.

name

Sticker set name.

Type

`str`

title

Sticker set title.

Type

`str`

is_animated

`True`, if the sticker set contains animated stickers.

Type

`bool`

is_video

`True`, if the sticker set contains video stickers.

New in version 13.11.

Type

`bool`

stickers

List of all set stickers.

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`.

New in version 20.0.

Type

`str`

thumbnail

Optional. Sticker set thumbnail in the `.WEBP`, `.TGS`, or `.WEBM` format.

New in version 20.2.

Type

`telegram.PhotoSize`

classmethod de_json(data, bot)

See `telegram.TelegramObject.de_json()`.

Inline Mode

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.
-

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`

Available In

`telegram.Update.chosen_inline_result`

classmethod `de_json(data, bot)`

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.

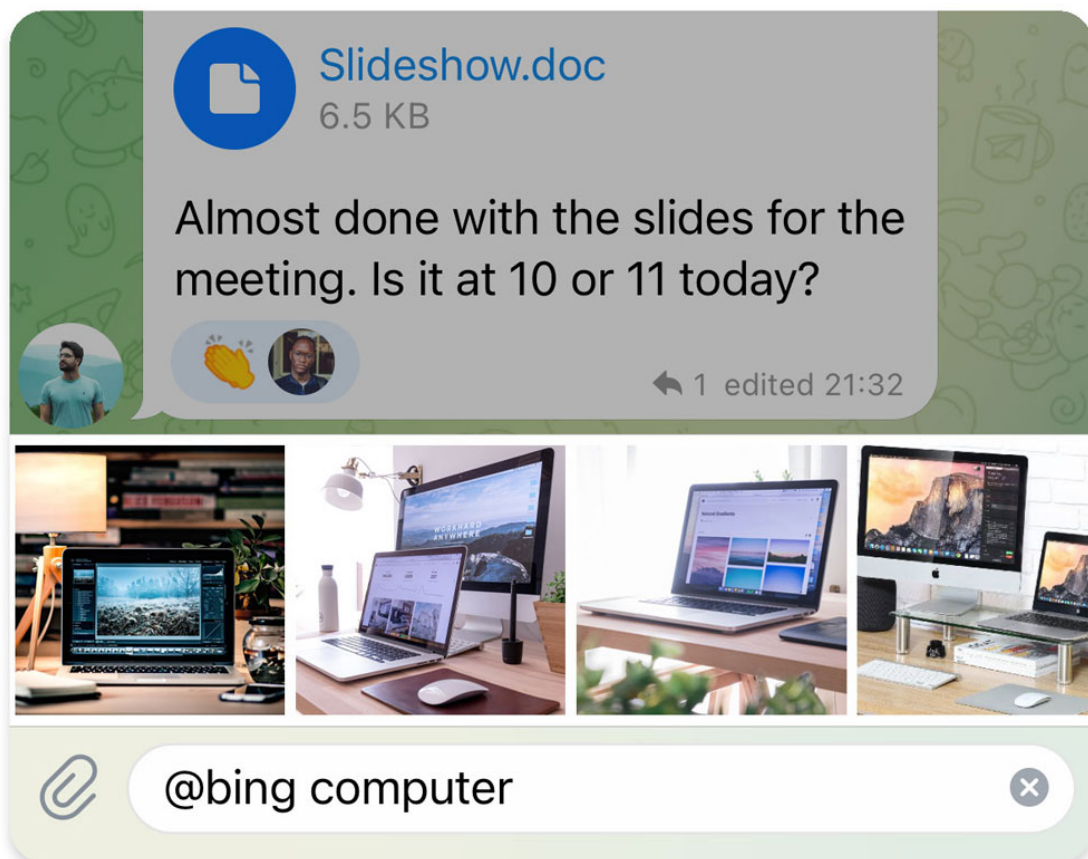


Fig. 3: Inline queries on Telegram

Available In

telegram.Update.inline_query

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.

Changed in version 20.0: The following are now keyword-only arguments in Bot methods: {`read`, `write`, `connect`, `pool`}_timeout, `answer.api_kwargs`, `auto_pagination`. Use a named argument for those, and notice that some positional arguments changed position as a result.

Parameters

- **`id`** (`str`) – Unique identifier for this query.
- **`from_user`** (`telegram.User`) – Sender.
- **`query`** (`str`) – Text of the query (up to 256 characters).
- **`offset`** (`str`) – Offset of the results to be returned, can be controlled by the bot.
- **`chat_type`** (`str`, optional) – Type of the chat, from which the inline query was sent. Can be either `'sender'` for a private chat with the inline query sender, `'private'`, `'group'`, `'supergroup'` or `'channel'`. The chat type should be always known for requests sent from official clients and most third-party clients, unless the request was sent from a secret chat.

New in version 13.5.

- **`location`** (`telegram.Location`, optional) – Sender location, only for bots that request user location.

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.

New 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`

New in version 20.0.

MAX_QUERY_LENGTH = 256

`telegram.constants.InlineQueryLimit.MAX_QUERY_LENGTH`

New in version 20.0.

MAX_RESULTS = 50

`telegram.constants.InlineQueryLimit.RESULTS`

New in version 13.2.

MAX_SWITCH_PM_TEXT_LENGTH = 64

`telegram.constants.InlineQueryLimit.MAX_SWITCH_PM_TEXT_LENGTH`

New in version 20.0.

MIN_SWITCH_PM_TEXT_LENGTH = 1

`telegram.constants.InlineQueryLimit.MIN_SWITCH_PM_TEXT_LENGTH`

New in version 20.0.

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*)

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

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`

Use In

- `telegram.Bot.answer_inline_query()`
 - `telegram.Bot.answer_web_app_query()`
-

MAX_ID_LENGTH = 64

`telegram.constants.InlineQueryResultLimit.MAX_ID_LENGTH`

New in version 20.0.

MIN_ID_LENGTH = 1

`telegram.constants.InlineQueryResultLimit.MIN_ID_LENGTH`

New in version 20.0.

InlineQueryResultArticle

```
class telegram.InlineQueryResultArticle(id, title, input_message_content, reply_markup=None,
                                         url=None, hide_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_inline_query()`
 - `telegram.Bot.answer_web_app_query()`
-

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 of the result.
- **`input_message_content`** (`telegram.InputMessageContent`) – Content of the message to be sent.
- **`reply_markup`** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **`url`** (`str`, optional) – URL of the result.
- **`hide_url`** (`bool`, optional) – Pass `True`, if you don't want the URL to be shown in the message.
- **`description`** (`str`, optional) – Short description of the result.
- **`thumbnail_url`** (`str`, optional) – Url of the thumbnail for the result.
New in version 20.2.
- **`thumbnail_width`** (`int`, optional) – Thumbnail width.
New in version 20.2.
- **`thumbnail_height`** (`int`, optional) – Thumbnail height.
New 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`

hide_url

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

Type

`bool`

description

Optional. Short description of the result.

Type

`str`

thumbnail_url

Optional. Url of the thumbnail for the result.

New in version 20.2.

Type

`str`

thumbnail_width

Optional. Thumbnail width.

New in version 20.2.

Type

`int`

thumbnail_height

Optional. Thumbnail height.

New 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.

Use In

- `telegram.Bot.answer_inline_query()`
 - `telegram.Bot.answer_web_app_query()`
-

See also:

[Working with Files and Media](#)

Parameters

- **id** (*str*) – Unique identifier for this result, 1- 64 Bytes.
- **audio_url** (*str*) – A valid URL for the audio file.
- **title** (*str*) – Title.
- **performer** (*str*, optional) – Performer.
- **audio_duration** (*str*, optional) – Audio duration in seconds.
- **caption** (*str*, optional) – Caption, 0-1024 characters after entities parsing.
- **parse_mode** (*str*, optional) – 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.

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`

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.

Use In

- `telegram.Bot.answer_inline_query()`
 - `telegram.Bot.answer_web_app_query()`
-

See also:

[Working with Files and Media](#)

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.

Use In

- `telegram.Bot.answer_inline_query()`
 - `telegram.Bot.answer_web_app_query()`
-

See also:

[Working with Files and Media](#)

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, *,
                                           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.

Use In

- *telegram.Bot.answer_inline_query()*
 - *telegram.Bot.answer_web_app_query()*
-

See also:

[Working with Files and Media](#)

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.

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`

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, *, 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.

Use In

- [telegram.Bot.answer_inline_query\(\)](#)
 - [telegram.Bot.answer_web_app_query\(\)](#)
-

See also:

[Working with Files and Media](#)

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.

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`

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, *, 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.

Use In

- `telegram.Bot.answer_inline_query\(\)`

- `telegram.Bot.answer_web_app_query()`
-

See also:

Working with Files and Media

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.

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`

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.

Use In

- `telegram.Bot.answer_inline_query()`
 - `telegram.Bot.answer_web_app_query()`
-

See also:

[Working with Files and Media](#)

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, *, 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.

Use In

- `telegram.Bot.answer_inline_query()`
 - `telegram.Bot.answer_web_app_query()`
-

See also:

[Working with Files and Media](#)

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.

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`](#)

[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.

Use In

- [`telegram.Bot.answer_inline_query\(\)`](#)
 - [`telegram.Bot.answer_web_app_query\(\)`](#)
-

See also:

[Working with Files and Media](#)

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_inline_query()`
 - `telegram.Bot.answer_web_app_query()`
-

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.

New in version 20.2.

- **thumbnail_width** (`int`, optional) – Thumbnail width.

New in version 20.2.

- **thumbnail_height** (`int`, optional) – Thumbnail height.

New 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.

New in version 20.2.

Type

`str`

thumbnail_width

Optional. Thumbnail width.

New in version 20.2.

Type

`int`

thumbnail_height

Optional. Thumbnail height.

New 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.

Use In

- `telegram.Bot.answer_inline_query()`
 - `telegram.Bot.answer_web_app_query()`
-

See also:

[Working with Files and Media](#)

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.
New in version 20.2.
- **`thumbnail_width`** (`int`, optional) – Thumbnail width.
New in version 20.2.
- **`thumbnail_height`** (`int`, optional) – Thumbnail height.
New 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.

New in version 20.2.

Type*str***thumbnail_width**

Optional. Thumbnail width.

New in version 20.2.

Type*int***thumbnail_height**

Optional. Thumbnail height.

New 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*.

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`

Use In

- `telegram.Bot.answer_inline_query()`
 - `telegram.Bot.answer_web_app_query()`
-

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, *, 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.

Use In

- `telegram.Bot.answer_inline_query()`
 - `telegram.Bot.answer_web_app_query()`
-

See also:

[Working with Files and Media](#)

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. File size must not exceed 1MB.
- **`gif_width`** (`int`, optional) – Width of the GIF.
- **`gif_height`** (`int`, optional) – Height of the GIF.
- **`gif_duration`** (`int`, optional) – Duration of the GIF in seconds.
- **`thumbnail_url`** (`str`, optional) – URL of the static (JPEG or GIF) or animated (MPEG4) thumbnail for the result.

Warning: The Bot API does **not** define this as an optional argument. It is formally optional for backwards compatibility with the deprecated `thumb_url`. If you pass neither `thumbnail_url` nor `thumb_url`, `ValueError` will be raised.

New in version 20.2.

- **`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'.

New 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.

Raises

`ValueError` – If neither `thumbnail_url` nor `thumb_url` is supplied or if both are supplied and are not equal.

type

'gif'.

Type

`str`

id

Unique identifier for this result, 1- 64 Bytes.

Type

`str`

gif_url

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

Type

`str`

gif_width

Optional. Width of the GIF.

Type

`int`

gif_height

Optional. Height of the GIF.

Type

`int`

gif_duration

Optional. Duration of the GIF in seconds.

Type

`int`

thumbnail_url

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

New 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'.

New 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`](#)

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_inline_query()`
- `telegram.Bot.answer_web_app_query()`

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`, optional) – Period in seconds for which the location will be updated, should be between 60 and 86400.
- **`heading`** (`int`, optional) – For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.
- **`proximity_alert_radius`** (`int`, optional) – For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and 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.
New in version 20.2.
- **`thumbnail_width`** (`int`, optional) – Thumbnail width.
New in version 20.2.
- **`thumbnail_height`** (`int`, optional) – Thumbnail height.
New 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.

Type

`int`

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.

New in version 20.2.

Type

`str`

thumbnail_width

Optional. Thumbnail width.

New in version 20.2.

Type

`int`

thumbnail_height

Optional. Thumbnail height.

New in version 20.2.

Type

`int`

HORIZONTAL_ACCURACY = 1500

`telegram.constants.LocationLimit.HORIZONTAL_ACCURACY`

New in version 20.0.

MAX_HEADING = 360

`telegram.constants.LocationLimit.MAX_HEADING`

New in version 20.0.

MAX_LIVE_PERIOD = 86400

`telegram.constants.LocationLimit.MAX_LIVE_PERIOD`

New in version 20.0.

MAX_PROXIMITY_ALERT_RADIUS = 100000

`telegram.constants.LocationLimit.MAX_PROXIMITY_ALERT_RADIUS`

New in version 20.0.

MIN_HEADING = 1

`telegram.constants.LocationLimit.MIN_HEADING`

New in version 20.0.

MIN_LIVE_PERIOD = 60

`telegram.constants.LocationLimit.MIN_LIVE_PERIOD`

New in version 20.0.

MIN_PROXIMITY_ALERT_RADIUS = 1

`telegram.constants.LocationLimit.MIN_PROXIMITY_ALERT_RADIUS`

New 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, *,
                                          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.

Use In

- `telegram.Bot.answer_inline_query()`
 - `telegram.Bot.answer_web_app_query()`
-

See also:

[Working with Files and Media](#)

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. File size must not exceed 1MB.
- **`mpeg4_width`** (`int`, optional) – Video width.
- **`mpeg4_height`** (`int`, optional) – Video height.
- **`mpeg4_duration`** (`int`, optional) – Video duration in seconds.
- **`thumbnail_url`** (`str`, optional) – URL of the static (JPEG or GIF) or animated (MPEG4) thumbnail for the result.

Warning: The Bot API does **not** define this as an optional argument. It is formally optional for backwards compatibility with the deprecated `thumb_url`. If you pass neither `thumbnail_url` nor `thumb_url`, `ValueError` will be raised.

New in version 20.2.

- **`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'.

New 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.

Raises

`ValueError` – If neither `thumbnail_url` nor `thumb_url` is supplied or if both are supplied and are not equal.

type

`'mpeg4_gif'.`

Type

`str`

id

Unique identifier for this result, 1- 64 Bytes.

Type

`str`

mpeg4_url

A valid URL for the MP4 file. File size must not exceed 1MB.

Type

`str`

mpeg4_width

Optional. Video width.

Type

`int`

mpeg4_height

Optional. Video height.

Type

`int`

mpeg4_duration

Optional. Video duration in seconds.

Type

`int`

thumbnail_url

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

New 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'.

New 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](#)

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, *, 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.

Use In

- [telegram.Bot.answer_inline_query\(\)](#)
 - [telegram.Bot.answer_web_app_query\(\)](#)
-

See also:

[Working with Files and Media](#)

Changed in version 20.5: Removed the deprecated argument and attribute `thumb_url`.

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`, optional) – URL of the thumbnail for the photo.

Warning: The Bot API does **not** define this as an optional argument. It is formally optional for backwards compatibility with the deprecated `thumb_url`. If you pass neither `thumbnail_url` nor `thumb_url`, `ValueError` will be raised.

New in version 20.2.

- `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.

Raises

`ValueError` – If neither `thumbnail_url` nor `thumb_url` is supplied or if both are supplied and are not equal.

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`

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.

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 `/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.

Type

`str`

Use In

[telegram.Bot.answer_inline_query\(\)](#)

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)
```

```
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_inline_query\(\)](#)
 - [telegram.Bot.answer_web_app_query\(\)](#)
-

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.

New in version 20.2.

- **`thumbnail_width`** (`int`, optional) – Thumbnail width.
New in version 20.2.
- **`thumbnail_height`** (`int`, optional) – Thumbnail height.
New 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.

New in version 20.2.

Type

`str`

thumbnail_width

Optional. Thumbnail width.

New in version 20.2.

Type

`int`

thumbnail_height

Optional. Thumbnail height.

New 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, *,
                                       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`.

Use In

- `telegram.Bot.answer_inline_query()`
- `telegram.Bot.answer_web_app_query()`

See also:

Working with Files and Media

Changed in version 20.5: Removed the deprecated argument and attribute `thumb_url`.

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.

Warning: The Bot API does **not** define this as an optional argument. It is formally optional for backwards compatibility with the deprecated `thumb_url`. If you pass neither `thumbnail_url` nor `thumb_url`, `ValueError` will be raised.

New in version 20.2.

- **`title`** (`str`, optional) – Title for the result.

Warning: The Bot API does **not** define this as an optional argument. It is formally optional to ensure backwards compatibility of `thumbnail_url` with the deprecated `thumb_url`, which required that `thumbnail_url` become optional. `TypeError` will be raised if no `title` is passed.

- **`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`, optional) – Video duration in seconds.
- **`description`** (`str`, optional) – Short description of the result.
- **`reply_markup`** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **`input_message_content`** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the video. This field is required if `InlineQueryResultVideo` is used to send an HTML-page as a result (e.g., a YouTube video).

Raises

- **`ValueError`** – If neither `thumbnail_url` nor `thumb_url` is supplied or if both are supplied and are not equal.
- **`TypeError`** – If no `title` is passed.

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.

New 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.

Type

`int`

description

Optional. Short description of the result.

Type

`str`

reply_markup

Optional. Inline keyboard attached to the message.

Type

`telegram.InlineKeyboardMarkup`

input_message_content

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

Type

`telegram.InputMessageContent`

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.

Use In

- `telegram.Bot.answer_inline_query()`
 - `telegram.Bot.answer_web_app_query()`
-

See also:

[Working with Files and Media](#)

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*, optional) – Recording duration in seconds.
- **reply_markup** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- **input_message_content** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the voice recording.

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.

TypeTuple[*telegram.MessageEntity*]**voice_duration**

Optional. Recording duration in seconds.

Type*int***reply_markup**

Optional. Inline keyboard attached to the message.

Type*telegram.InlineKeyboardMarkup***input_message_content**

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

Type*telegram.InputMessageContent*

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,  
                                       disable_web_page_preview=None, entities=None,  
                                       link_preview_options=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

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.

- **`disable_web_page_preview`** (`bool`, optional) – Disables link previews for links in the sent message. 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.

Deprecated since version 20.8: In future versions, this argument will become keyword only.

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

New in version 20.8.

`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. Mutually exclusive with [disable_web_page_preview](#).

New in version 20.8.

Type`LinkPreviewOptions`

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`
-

property `disable_web_page_preview`

Disables link previews for links in the sent message.

Deprecated since version 20.8.

Type

Optional[bool]

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.

Parameters

- [latitude](#) (float) – Latitude of the location in degrees.
- [longitude](#) (float) – Longitude of the location in degrees.
- [horizontal_accuracy](#) (float, optional) – The radius of uncertainty for the location, measured in meters; 0- 1500.
- [live_period](#) (int, optional) – Period in seconds for which the location will be updated, should be between 60 and 86400.
- [heading](#) (int, optional) – For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.
- [proximity_alert_radius](#) (int, optional) – For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and 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.

Type

int

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`

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`
-

HORIZONTAL_ACCURACY = 1500

`telegram.constants.LocationLimit.HORIZONTAL_ACCURACY`

New in version 20.0.

MAX_HEADING = 360

`telegram.constants.LocationLimit.MAX_HEADING`

New in version 20.0.

MAX_LIVE_PERIOD = 86400

`telegram.constants.LocationLimit.MAX_LIVE_PERIOD`

New in version 20.0.

MAX_PROXIMITY_ALERT_RADIUS = 100000

`telegram.constants.LocationLimit.MAX_PROXIMITY_ALERT_RADIUS`

New in version 20.0.

MIN_HEADING = 1

`telegram.constants.LocationLimit.MIN_HEADING`

New in version 20.0.

MIN_LIVE_PERIOD = 60

`telegram.constants.LocationLimit.MIN_LIVE_PERIOD`

New in version 20.0.

MIN_PROXIMITY_ALERT_RADIUS = 1

`telegram.constants.LocationLimit.MIN_PROXIMITY_ALERT_RADIUS`

New 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 Place details are mutually exclusive. However, this behaviour is undocumented and might be changed by Telegram.

Parameters

- `latitude` (`float`) – Latitude of the location in degrees.
- `longitude` (`float`) – Longitude of the location in degrees.
- `title` (`str`) – Name of the venue.
- `address` (`str`) – Address of the venue.
- `foursquare_id` (`str`, optional) – Foursquare identifier of the venue, if known.
- `foursquare_type` (`str`, optional) – Foursquare type of the venue, if known. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”.)
- `google_place_id` (`str`, optional) – Google Places identifier of the venue.
- `google_place_type` (`str`, optional) – Google Places type of the venue. (See supported types.)

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`

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`
-

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.

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`

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`
-

InputInvoiceMessageContent

class telegram.**InputInvoiceMessageContent**(*title, description, payload, provider_token, currency, prices, max_tip_amount=None, suggested_tip_amounts=None, provider_data=None, photo_url=None, photo_size=None, photo_width=None, photo_height=None, need_name=None, need_phone_number=None, need_email=None, need_shipping_address=None, send_phone_number_to_provider=None, send_email_to_provider=None, is_flexible=None, *, 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*, *provider_token*, *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`
-

New in version 13.5.

Parameters

- **`title`** (`str`) – Product name. 1- 32 characters.
- **`description`** (`str`) – Product description. 1- 255 characters.
- **`payload`** (`str`) – Bot-defined invoice payload. 1- 128 bytes. This will not be displayed to the user, use for your internal processes.
- **`provider_token`** (`str`) – Payment provider token, obtained via [@Botfather](#).
- **`currency`** (`str`) – Three-letter ISO 4217 currency code, see more on [currencies](#)
- **`prices`** (Sequence[[telegram.LabeledPrice](#)]) – Price breakdown, a list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, 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.

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

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

title

Product name. 1- 32 characters.

Type

`str`

description

Product description. 1- 255 characters.

Type

`str`

payload

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

Type

`str`

provider_token

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

Type

`str`

currency

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

Type

`str`

prices

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

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.

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.

Type

`bool`

need_phone_number

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

Type

`bool`

need_email

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

Type

`bool`

need_shipping_address

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

Type

`bool`

send_phone_number_to_provider

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

Type
`bool`

send_email_to_provider

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

Type
`bool`

is_flexible

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

Type
`bool`

classmethod `de_json(data, bot)`

See `telegram.TelegramObject.de_json()`.

Payments

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.

Parameters

- **title** (`str`) – Product name.
- **description** (`str`) – Product description.
- **start_parameter** (`str`) – Unique bot deep-linking parameter that can be used to generate this invoice.
- **currency** (`str`) – Three-letter ISO 4217 currency code.
- **total_amount** (`int`) – Total price in the smallest units of the currency (integer, not float/double). For example, for a price of US\$ 1.45 pass `amount = 145`. See the `exp` parameter in `currencies.json`, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).

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.

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 `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).

Type

`int`

Available In

- `telegram.ExternalReplyInfo.invoice`
 - `telegram.Message.invoice`
-

MAX_DESCRIPTION_LENGTH = 255

`telegram.constants.InvoiceLimit.MAX_DESCRIPTION_LENGTH`

New in version 20.0.

MAX_PAYLOAD_LENGTH = 128

`telegram.constants.InvoiceLimit.MAX_PAYLOAD_LENGTH`

New in version 20.0.

MAX_TIP_AMOUNTS = 4

`telegram.constants.InvoiceLimit.MAX_TIP_AMOUNTS`

New in version 20.0.

MAX_TITLE_LENGTH = 32

`telegram.constants.InvoiceLimit.MAX_TITLE_LENGTH`

New in version 20.0.

MIN_DESCRIPTION_LENGTH = 1

`telegram.constants.InvoiceLimit.MIN_DESCRIPTION_LENGTH`

New in version 20.0.

MIN_PAYLOAD_LENGTH = 1

`telegram.constants.InvoiceLimit.MIN_PAYLOAD_LENGTH`

New in version 20.0.

MIN_TITLE_LENGTH = 1

`telegram.constants.InvoiceLimit.MIN_TITLE_LENGTH`

New in version 20.0.

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

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 `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).

Type

[int](#)

Use In

- [telegram.Bot.create_invoice_link\(\)](#)
 - [telegram.Bot.send_invoice\(\)](#)
-

Available In

- [telegram.InputInvoiceMessageContent.prices](#)
 - [telegram.ShippingOption.prices](#)
-

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.

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](#)

Available In

- [telegram.PreCheckoutQuery.order_info](#)
 - [telegram.SuccessfulPayment.order_info](#)
-

classmethod **de_json**(*data, bot*)

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.

Parameters

- **id** (*str*) – Unique query identifier.
- **from_user** ([telegram.User](#)) – User who sent the query.
- **currency** (*str*) – Three-letter ISO 4217 currency code.
- **total_amount** (*int*) – Total price in the smallest units of the currency (integer, not float/double). For example, for a price of US\$ 1.45 pass `amount = 145`. See the `exp` parameter in [currencies.json](#), it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- **invoice_payload** (*str*) – Bot specified invoice payload.
- **shipping_option_id** (*str*, optional) – Identifier of the shipping option chosen by the user.
- **order_info** ([telegram.OrderInfo](#), optional) – Order info provided by the user.

id

Unique query identifier.

Type

str

from_user

User who sent the query.

Type

[telegram.User](#)

currency

Three-letter ISO 4217 currency code.

Type

str

total_amount

Total price in the smallest units of the currency (integer, not float/double). For example, for a price of US\$ 1.45 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).

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`

Available In

`telegram.Update.pre_checkout_query`

async `answer`(*ok*, *error_message=None*, *, *read_timeout=None*, *write_timeout=None*,
connect_timeout=None, *pool_timeout=None*, *api_kwargs=None*)

Shortcut for:

```
await bot.answer_pre_checkout_query(update.pre_checkout_query.id, *args, kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.answer_pre_checkout_query()`.

classmethod `de_json`(*data*, *bot*)

See `telegram.TelegramObject.de_json()`.

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.

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`

Available In

- `telegram.OrderInfo.shipping_address`
 - `telegram.ShippingQuery.shipping_address`
-

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

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.

TypeTuple[`telegram.LabeledPrice`]

Use In`telegram.Bot.answer_shipping_query()`

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.

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`

Available In

`telegram.Update.shipping_query`

async `answer(ok, shipping_options=None, error_message=None, *, read_timeout=None, write_timeout=None, connect_timeout=None, pool_timeout=None, api_kwargs=None)`

Shortcut for:

```
await bot.answer_shipping_query(update.shipping_query.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.answer_shipping_query()`.

classmethod `de_json(data, bot)`

See `telegram.TelegramObject.de_json()`.

SuccessfulPayment

class `telegram.SuccessfulPayment(currency, total_amount, invoice_payload, telegram_payment_charge_id, provider_payment_charge_id, shipping_option_id=None, order_info=None, *, api_kwargs=None)`

Bases: `telegram.TelegramObject`

This object contains basic information about a successful payment.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `telegram_payment_charge_id` and `provider_payment_charge_id` are equal.

Parameters

- **currency** (`str`) – Three-letter ISO 4217 currency code.
- **total_amount** (`int`) – Total price in the smallest units of the currency (integer, not float/double). For example, for a price of US\$ 1.45 pass `amount = 145`. See the `exp` parameter in `currencies.json`, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- **invoice_payload** (`str`) – Bot specified invoice payload.
- **shipping_option_id** (`str`, optional) – Identifier of the shipping option chosen by the user.
- **order_info** (`telegram.OrderInfo`, optional) – Order info provided by the user.
- **telegram_payment_charge_id** (`str`) – Telegram payment identifier.
- **provider_payment_charge_id** (`str`) – Provider payment identifier.

currency

Three-letter ISO 4217 currency code.

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 `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).

Type

`int`

invoice_payload

Bot specified invoice payload.

Type

`str`

shipping_option_id

Optional. Identifier of the shipping option chosen by the user.

Type

`str`

order_info

Optional. Order info provided by the user.

Type

[`telegram.OrderInfo`](#)

telegram_payment_charge_id

Telegram payment identifier.

Type

`str`

provider_payment_charge_id

Provider payment identifier.

Type

`str`

Available In

[`telegram.Message.successful_payment`](#)

classmethod `de_json(data, bot)`

See [`telegram.TelegramObject.de_json\(\)`](#).

Games

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.

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](#)

Available In

[telegram.Message.game](#)

classmethod de_json(data, bot)

See [telegram.TelegramObject.de_json\(\)](#).

parse_text_entities(types=None)

Returns a [dict](#) that maps [telegram.MessageEntity](#) to [str](#). It contains entities from this message filtered by their [type](#) attribute as the key, and the text that each entity belongs to as the value of the [dict](#).

Note: This method should always be used instead of the [text_entities](#) attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See [parse_text_entity](#) for more info.

Parameters

types (List[[str](#)], optional) – List of [telegram.MessageEntity](#) types as strings. If the [type](#) attribute of an entity is contained in this list, it will be returned. Defaults to [telegram.MessageEntity.ALL_TYPES](#).

Returns

A dictionary of entities mapped to the text that belongs to them, calculated based on UTF-16 codepoints.

Return type

Dict[[telegram.MessageEntity](#), [str](#)]

parse_text_entity(entity)

Returns the text from a given [telegram.MessageEntity](#).

Note: This method is present because Telegram calculates the offset and length in UTF-16 codepoint pairs, which some versions of Python don't handle automatically. (That is, you can't just slice `Message.text` with the offset and length.)

Parameters

entity ([telegram.MessageEntity](#)) – The entity to extract the text from. It must be an entity that belongs to this message.

Returns

The text of the given entity.

Return type`str`**Raises**`RuntimeError` – If this game has no text.

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.

Parameters

- `position` (`int`) – Position in high score table for the game.
- `user` (`telegram.User`) – User.
- `score` (`int`) – Score.

position

Position in high score table for the game.

Type`int`**user**

User.

Type`telegram.User`**score**

Score.

Type`int`

```
classmethod de_json(data, bot)
```

See `telegram.TelegramObject.de_json()`.

Passport

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.data`
 - `telegram.EncryptedCredentials.decrypted_data`
 - `telegram.PassportData.decrypted_credentials`
-

classmethod `de_json(data, bot)`

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`.

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`

Available In

`telegram.SecureValue.data`

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`.

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

Available In

telegram.PassportData.credentials

property decrypted_data**Lazily decrypt and return credentials data. This object**

also contains the user specified nonce as *decrypted_data.nonce*.

Raises

telegram.error.PassportDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type

telegram.Credentials

property decrypted_secret

Lazily decrypt and return secret.

Raises

telegram.error.PassportDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type

str

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, credentials=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](#).

Parameters

- **type** ([str](#)) – Element type. One of “personal_details”, “passport”, “driver_license”, “identity_card”, “internal_passport”, “address”, “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”, “phone_number”, “email”.
- **hash** ([str](#)) – Base64-encoded element hash for using in [telegram.PassportElementErrorUnspecified](#).
- **data** ([telegram.PersonalDetails](#) | [telegram.IdDocumentData](#) | [telegram.ResidentialAddress](#) | [str](#), optional) – Decrypted or encrypted data, available for “personal_details”, “passport”, “driver_license”, “identity_card”, “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 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 for “passport”, “driver_license”, “identity_card” and “internal_passport”.
- **reverse_side** ([telegram.PassportFile](#), optional) – Encrypted/decrypted file with the reverse side of the document, provided by the user. Available for “driver_license” and “identity_card”.
- **selfie** ([telegram.PassportFile](#), optional) – Encrypted/decrypted file with the selfie of the user holding a document, provided by the user; available for “passport”, “driver_license”, “identity_card” and “internal_passport”.
- **translation** (Sequence[[telegram.PassportFile](#)], optional) – Array of encrypted/decrypted files with translated versions of documents provided by the user. Available if requested for “passport”, “driver_license”, “identity_card”, “internal_passport”, “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.

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 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 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 for “passport”, “driver_license”, “identity_card” and “internal_passport”.

Type

`telegram.PassportFile`

reverse_side

Optional. Encrypted/decrypted file with the reverse side of the document, provided by the user. Available for “driver_license” and “identity_card”.

Type

`telegram.PassportFile`

selfie

Optional. Encrypted/decrypted file with the selfie of the user holding a document, provided by the user; available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

Type

telegram.PassportFile

translation

Optional. Array of encrypted/decrypted files with translated versions of documents provided by the user. Available if requested for “passport”, “driver_license”, “identity_card”, “internal_passport”, “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.

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*]

Available In

telegram.PassportData.data

classmethod de_json(data, bot)

See *telegram.TelegramObject.de_json()*.

classmethod de_json_decrypted(data, bot, credentials)

Variant of *telegram.TelegramObject.de_json()* that also takes into account passport credentials.

Parameters

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

Return type

telegram.EncryptedPassportElement

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.

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`

Available In

- `telegram.SecureValue.files`
 - `telegram.SecureValue.front_side`
 - `telegram.SecureValue.reverse_side`
 - `telegram.SecureValue.selfie`
 - `telegram.SecureValue.translation`
-

IdDocumentData

class `telegram.IdDocumentData`(*document_no*, *expiry_date*, *, *api_kwargs=None*)

Bases: `telegram.TelegramObject`

This object represents the data of an identity document.

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`

Available In

`telegram.EncryptedPassportElement.data`

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`.

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](#)

Available In

[telegram.Message.passport_data](#)

classmethod de_json(data, bot)

See [telegram.TelegramObject.de_json\(\)](#).

property decrypted_credentials

Lazily decrypt and return credentials that were used

to decrypt the data. This object also contains the user specified payload as *decrypted_data.payload*.

Raises

[telegram.error.PassportDecryptionError](#) – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type

[telegram.Credentials](#)

property decrypted_data

Lazily decrypt and return information

about documents and other Telegram Passport elements which were shared with the bot.

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.

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

Use In

[telegram.Bot.set_passport_data_errors\(\)](#)

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.

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`

Use In

`telegram.Bot.set_passport_data_errors()`

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.

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`

Use In

`telegram.Bot.set_passport_data_errors()`

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.

Parameters

- **type** (`str`) – The section of the user's Telegram Passport which has the issue, one of "utility_bill", "bank_statement", "rental_agreement", "passport_registration", "temporary_registration".
- **file_hashes** (List[`str`]) – List of base64-encoded file hashes.
- **message** (`str`) – Error message.

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`

message

Error message.

Type
`str`

Use In

`telegram.Bot.set_passport_data_errors()`

property file_hashes

List of base64-encoded file hashes.

Deprecated since version 20.6: This attribute will return a tuple instead of a list in future major versions.

to_dict(*recursive*=True)

See `telegram.TelegramObject.to_dict()` for details.

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.

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](#)

Use In

[telegram.Bot.set_passport_data_errors\(\)](#)

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.

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`

Use In

`telegram.Bot.set_passport_data_errors()`

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.

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`

Use In

`telegram.Bot.set_passport_data_errors()`

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.

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`

Use In

`telegram.Bot.set_passport_data_errors()`

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.

Parameters

- [type](#) ([str](#)) – Type of element of the user's Telegram Passport which has the issue, one of "passport", "driver_license", "identity_card", "internal_passport", "utility_bill", "bank_statement", "rental_agreement", "passport_registration", "temporary_registration".
- [file_hashes](#) ([List\[str\]](#)) – List of base64-encoded file hashes.
- [message](#) ([str](#)) – Error message.

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](#)

message

Error message.

Type

[str](#)

Use In

[telegram.Bot.set_passport_data_errors\(\)](#)

property file_hashes

List of base64-encoded file hashes.

Deprecated since version 20.6: This attribute will return a tuple instead of a list in future major versions.

to_dict(recursive=True)

See [telegram.TelegramObject.to_dict\(\)](#) for details.

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.

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

Use In

telegram.Bot.set_passport_data_errors()

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.

Parameters

- **file_id** (*str*) – Identifier for this file, which can be used to download or reuse the file.
- **file_unique_id** (*str*) – Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the file.
- **file_size** (*int*) – File size in bytes.
- **file_date** (*int*) – Unix time when the file was uploaded.

Deprecated since version 20.6: This argument will only accept a datetime instead of an integer in future major versions.

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`

Available In

- `telegram.EncryptedPassportElement.files`
 - `telegram.EncryptedPassportElement.front_side`
 - `telegram.EncryptedPassportElement.reverse_side`
 - `telegram.EncryptedPassportElement.selfie`
 - `telegram.EncryptedPassportElement.translation`
-

classmethod `de_json_decrypted(data, bot, credentials)`

Variant of `telegram.TelegramObject.de_json()` that also takes into account passport credentials.

Parameters

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

Return type
`telegram.PassportFile`

classmethod `de_list_decrypted(data, bot, credentials)`

Variant of `telegram.TelegramObject.de_list()` that also takes into account passport credentials.

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`) – The bot associated with these objects.
- **`credentials`** (`telegram.FileCredentials`) – The credentials

Return type
`Tuple[telegram.PassportFile]`

property `file_date`

Unix time when the file was uploaded.

Deprecated since version 20.6: This attribute will return a datetime instead of a integer in future major versions.

Type
`int`

```
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` –

`to_dict(recursive=True)`

See `telegram.TelegramObject.to_dict()` for details.

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.

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`

Available In

`telegram.EncryptedPassportElement.data`

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.

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

Available In

[*telegram.EncryptedPassportElement.data*](#)

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.

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*

Available In*telegram.Credentials.secure_data*

classmethod `de_json(data, bot)`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.

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*]

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*
-

classmethod `de_json(data, bot)`

See *telegram.TelegramObject.de_json()*.

10.2 telegram.ext package

Extensions over the Telegram Bot API to facilitate bot making

10.2.1 Application

```
class telegram.ext.Application(*, bot, update_queue, updater, job_queue, update_processor,
                               persistence, context_types, post_init, post_shutdown, post_stop)
```

Bases: `typing.Generic`, `typing.AsyncContextManager`

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()
```

Available In

`telegram.ext.CallbackContext.application`

Returned In

`telegram.ext.ApplicationBuilder.build()`

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

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 inadvisable.
 - 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 inadvisable.
 - 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()`.

New 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

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: Optional[object], context: ↪
↪ CallbackContext)
```

The error that happened will be present in `telegram.ext.CallbackContext.error`.

- **block** (bool, optional) – Determines whether the return value of the callback should be awaited before processing the next error handler in *process_error()*. Defaults to `True`.

add_handler(handler, group=0)

Register a handler.

TL;DR: Order and priority counts. 0 or 1 handlers per group will be used. End handling of update with `telegram.ext.ApplicationHandlerStop`.

A handler must be an instance of a subclass of `telegram.ext.BaseHandler`. All handlers are organized in groups with a numeric value. The default group is 0. All groups will be evaluated for handling an update, but only 0 or 1 handler per group will be used. If `telegram.ext.`

`ApplicationHandlerStop` is raised from one of the handlers, no further handlers (regardless of the group) will be called.

The priority/order of handlers is determined as follows:

- Priority of the group (lower group number == higher priority)
- The first handler in a group which can handle an update (see `telegram.ext.BaseHandler.check_update`) will be used. Other handlers from the group will not be used. The order in which handlers were added to the group defines the priority.

Warning: Adding persistent `telegram.ext.ConversationHandler` after the application has been initialized is discouraged. This is because the persisted conversation states need to be loaded into memory while the application is already processing updates, which might lead to race conditions and undesired behavior. In particular, current conversation states may be overridden by the loaded data.

Parameters

- **handler** (`telegram.ext.BaseHandler`) – A `BaseHandler` instance.
- **group** (`int`, optional) – The group identifier. Default is `0`.

add_handlers(*handlers*, *group=0*)

Registers multiple handlers at once. The order of the handlers in the passed sequence(s) matters. See `add_handler()` for details.

New in version 20.0.

Parameters

- **handlers** (`List[telegram.ext.BaseHandler] | Dict[int, List[telegram.ext.BaseHandler]]`) – Specify a sequence of handlers *or* a dictionary where the keys are groups and values are handlers.
- **group** (`int`, optional) – Specify which group the sequence of `handlers` should be added to. Defaults to `0`.

Example:

```
app.add_handlers(handlers={
    -1: [MessageHandler(...)],
    1: [CallbackQueryHandler(...), CommandHandler(...)]
})
```

static builder()

Convenience method. Returns a new `telegram.ext.ApplicationBuilder`.

New in version 20.0.

property concurrent_updates

The number of concurrent updates that will be processed in parallel. A value of `0` indicates updates are *not* being processed concurrently.

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.

New 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.

New in version 20.0.

Parameters

chat_id (int) – The chat id to delete. The entry will be deleted even if it is not empty.

drop_user_data(*user_id*)

Drops the corresponding entry from the *user_data*. Will also be deleted from the persistence on the next run of `update_persistence()`, if applicable.

Warning: When using `concurrent_updates` or the `job_queue`, `process_update()` or `telegram.ext.Job.run()` may re-create this entry due to the asynchronous nature of these features. Please make sure that your program can avoid or handle such situations.

New in version 20.0.

Parameters

`user_id` (`int`) – The user id to delete. The entry will be deleted even if it is not empty.

`async initialize()`

Initializes the Application by initializing:

- The `bot`, by calling `telegram.Bot.initialize()`.
- The `updater`, by calling `telegram.ext.Updater.initialize()`.
- The `persistence`, by loading persistent conversations and data.
- 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.

New 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.
New in version 20.0.
- **coroutine** (coroutine function, optional) – The coroutine that caused the error.

Returns

`True`, if one of the error handlers raised `telegram.ext.ApplicationHandlerStop`.
`False`, otherwise.

Return type

`bool`

async process_update(update)

Processes a single update and marks the update to be updated by the persistence later. Exceptions raised by handler callbacks will be processed by [process_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.

Parameters

callback (coroutine function) – The error handler to remove.

remove_handler(handler, group=0)

Remove a handler from the specified group.

Parameters

- **handler** ([telegram.ext.BaseHandler](#)) – A [telegram.ext.BaseHandler](#) instance.
- **group** (object, optional) – The group identifier. Default is 0.

run_polling(poll_interval=0.0, timeout=10, bootstrap_retries=-1, read_timeout=None, write_timeout=None, connect_timeout=None, pool_timeout=None, allowed_updates=None, drop_pending_updates=None, close_loop=True, stop_signals=None)

Convenience method that takes care of initializing and starting the app, polling updates from Telegram using [telegram.ext.Updater.start_polling\(\)](#) and a graceful shutdown of the app on exit.

The app will shut down when [KeyboardInterrupt](#) or [SystemExit](#) is raised. On unix, the app will also shut down on receiving the signals specified by [stop_signals](#).

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\(\)](#)

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()`.
-

Parameters

- **`poll_interval`** (`float`, optional) – Time to wait between polling updates from Telegram in seconds. Default is `0.0`.
- **`timeout`** (`int`, optional) – Passed to `telegram.Bot.get_updates.timeout`. Default is `10` seconds.
- **`bootstrap_retries`** (`int`, optional) – Whether the bootstrapping phase of the `telegram.ext.Updater` will retry on failures on the Telegram server.
 - `< 0` - retry indefinitely (default)
 - `0` - no retries
 - `> 0` - retry up to X times

- **`read_timeout`** (`float`, optional) – Value to pass to `telegram.Bot.get_updates.read_timeout`. Defaults to `DEFAULT_NONE`.

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

Deprecated since version 20.7: Deprecated in favor of setting the timeout via `telegram.ext.ApplicationBuilder.get_updates_read_timeout()`.

- **`write_timeout`** (`float | None`, optional) – Value to pass to `telegram.Bot.get_updates.write_timeout`. Defaults to `DEFAULT_NONE`.

Deprecated since version 20.7: Deprecated in favor of setting the timeout via `telegram.ext.ApplicationBuilder.get_updates_write_timeout()`.

- **`connect_timeout`** (`float | None`, optional) – Value to pass to `telegram.Bot.get_updates.connect_timeout`. Defaults to `DEFAULT_NONE`.

Deprecated since version 20.7: Deprecated in favor of setting the timeout via `telegram.ext.ApplicationBuilder.get_updates_connect_timeout()`.

- **`pool_timeout`** (`float | None`, optional) – Value to pass to `telegram.Bot.get_updates.pool_timeout`. Defaults to `DEFAULT_NONE`.

Deprecated since version 20.7: Deprecated in favor of setting the timeout via `telegram.ext.ApplicationBuilder.get_updates_pool_timeout()`.

- **`drop_pending_updates`** (`bool`, optional) – Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is `False`.
- **`allowed_updates`** (`List[str]`, optional) – Passed to `telegram.Bot.get_updates()`.
- **`close_loop`** (`bool`, optional) – If `True`, the current event loop will be closed upon shutdown. 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. 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 of the `telegram.ext.Updater` will retry on failures on the Telegram server.
 - `< 0` - retry indefinitely
 - `0` - no retries (default)
 - `> 0` - retry up to X times
- **webhook_url** (`str`, optional) – Explicitly specify the webhook url. Useful behind NAT, reverse proxy, etc. Default is derived from `listen`, `port`, `url_path`, `cert`, and `key`.
- **allowed_updates** (`List[str]`, optional) – Passed to `telegram.Bot.set_webhook()`.
- **drop_pending_updates** (`bool`, optional) – Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is `False`.
- **ip_address** (`str`, optional) – Passed to `telegram.Bot.set_webhook()`.
- **max_connections** (`int`, optional) – Passed to `telegram.Bot.set_webhook()`. Defaults to `40`.
- **close_loop** (`bool`, optional) – If `True`, the current event loop will be closed upon shutdown. Defaults to `True`.

See also:

`asyncio.loop.close()`

- **stop_signals** (`Sequence[int]` | `None`, optional) – Signals that will shut down the app. Pass `None` to not use stop signals. Defaults to `signal.SIGINT`, `signal.SIGTERM` and `signal.SIGABRT`.

Caution: Not every `asyncio.AbstractEventLoop` implements `asyncio.loop.add_signal_handler()`. Most notably, the standard event loop on Windows, `asyncio.ProactorEventLoop`, does not implement this method. If this method is not available, stop signals can not be set.

- **secret_token** (`str`, optional) – Secret token to ensure webhook requests originate from Telegram. See `telegram.Bot.set_webhook.secret_token` for more details.

When added, the web server started by this call will expect the token to be set in the `X-Telegram-Bot-API-Secret-Token` header of an incoming request and will raise a `http.HTTPStatus.FORBIDDEN` error if either the header isn't set or it is set to a wrong token.

New in version 20.0.

- **unix** (`pathlib.Path` | `str`, optional) – Path to the unix socket file. Path does not need to exist, in which case the file will be created.

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`.

New in version 20.8.

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.

Note: If the application is not running, this method does nothing.

New in version 20.5.

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](#)

New in version 20.4.

Type

telegram.ext.BaseUpdateProcessor

10.2.2 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](#)

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()***Parameters**

base_file_url (*str*) – The URL.

Returns

The same builder with the updated argument.

Return type*ApplicationBuilder***base_url**(*base_url*)

Sets the base URL for `telegram.ext.Application.bot`. If not called, will default to `'https://api.telegram.org/bot'`.

See also:*telegram.Bot.base_url*, *Local Bot API Server*, *base_file_url()***Parameters**

base_url (*str*) – The URL.

Returns

The same builder with the updated argument.

Return type

ApplicationBuilder

bot(*bot*)

Sets a *telegram.Bot* instance for *telegram.ext.Application.bot*. Instances of subclasses like *telegram.ext.ExtBot* are also valid.

Parameters

bot (*telegram.Bot*) – The bot.

Returns

The same builder with the updated argument.

Return type

ApplicationBuilder

build()

Builds a *telegram.ext.Application* with the provided arguments.

Calls *telegram.ext.JobQueue.set_application()* and *telegram.ext.BasePersistence.set_bot()* if appropriate.

Returns

telegram.ext.Application

concurrent_updates(*concurrent_updates*)

Specifies if and how many updates may be processed concurrently instead of one by one. 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]
```

New 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()

New 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_proxy_url`(*get_updates_proxy_url*)

Legacy name for *get_updates_proxy()*, kept for backward compatibility.

See also:

proxy()

Deprecated since version 20.7.

Parameters

`get_updates_proxy_url` (`str` | `httpx.Proxy` | `httpx.URL`) – See `telegram.ext.ApplicationBuilder.get_updates_proxy.get_updates_proxy`.

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()`

New 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”.

New 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`

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()*.

New 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

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\(\)`](#)

New 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`](#)

`proxy_url(proxy_url)`

Legacy name for `proxy()`, kept for backward compatibility.

See also:

[`get_updates_proxy\(\)`](#)

Deprecated since version 20.7.

Parameters

`proxy_url` (`str` | `httpx.Proxy` | `httpx.URL`) – See `telegram.ext.ApplicationBuilder.proxy.proxy`.

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()*

New 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*

10.2.3 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

10.2.4 BaseUpdateProcessor

class telegram.ext.**BaseUpdateProcessor**(max_concurrent_updates)

Bases: [typing.AsyncContextManager](#), [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()
```

Use In

[telegram.ext.ApplicationBuilder.concurrent_updates\(\)](#)

Available In

[telegram.ext.Application.update_processor](#)

See also:

[__aenter__\(\)](#) and [__aexit__\(\)](#).

See also:

[Concurrency](#)

New 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.

`abstract 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.

`abstract 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.

`abstract async shutdown()`

Shutdown the processor so resources can be freed. Must be implemented by a subclass.

See also:

[initialize\(\)](#)

10.2.5 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

Parameters

- **application** (`telegram.ext.Application`) – The application associated with this context.
- **chat_id** (`int`, optional) – The ID of the chat associated with this object. Used to provide `chat_data`.
New in version 20.0.
- **user_id** (`int`, optional) – The ID of the user associated with this object. Used to provide `user_data`.
New in version 20.0.

coroutine

Optional. Only present in error handlers if the error was caused by 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 <code>chat_data</code> 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.

New in version 13.6.

Note: Will *not* raise exceptions in case the data is not found in the cache. Will raise `KeyError` in case the callback query can not be found in the cache.

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.
- **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`.

New in version 20.0.

New 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()`.

New in version 13.6.

update(`data`)

Updates `self.__slots__` with the passed data.

Parameters

data (Dict[str, object]) – The data.

property update_queue

The `asyncio.Queue` instance used by the `telegram.ext.Application` and (usually) the `telegram.ext.Updater` associated with this context.

Type

`asyncio.Queue`

property user_data

Optional. An object that can be used to keep any data in. For each update from the same user it will be the same `ContextTypes.user_data`. Defaults to `dict`.

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`

10.2.6 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`

Use In

`telegram.ext.ApplicationBuilder.context_types()`

Available In

- `telegram.ext.Application.context_types`
 - `telegram.ext.PicklePersistence.context_types`
-

See also:

[Architecture Overview](#), [Storing Bot, User and Chat Related Data](#)

New in version 13.6.

Parameters

- **context** (*type*, optional) – Determines the type of the `context` argument of all (error-)handler callbacks and job callbacks. Must be a subclass of `telegram.ext.CallbackContext`. Defaults to `telegram.ext.CallbackContext`.
- **bot_data** (*type*, optional) – Determines the type of `context.bot_data` of all (error-)handler callbacks and job callbacks. Defaults to `dict`. Must support instantiating without arguments.

- **chat_data** (type, optional) – Determines the type of `context.chat_data` of all (error-)handler callbacks and job callbacks. Defaults to `dict`. Must support instantiating without arguments.
- **user_data** (type, optional) – Determines the type of `context.user_data` of all (error-)handler callbacks and job callbacks. Defaults to `dict`. Must support instantiating without arguments.

DEFAULT_TYPE

Shortcut for the type annotation for the `context` argument that's correct for the default settings, i.e. if `telegram.ext.ContextTypes` is not used.

Example

```
async def callback(update: Update, context: ContextTypes.DEFAULT_TYPE):  
    ...
```

alias of `CallbackContext[ExtBot[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.

10.2.7 Defaults

```
final class telegram.ext.Defaults(parse_mode=None, disable_notification=None,  
                                  disable_web_page_preview=None, quote=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

Use In

`telegram.ext.ApplicationBuilder.defaults()`

See also:

[Architecture Overview, Adding Defaults to Your Bot](#)

Changed in version 20.0: Removed the argument and attribute `timeout`. Specify default timeout behavior for the networking backend directly via `telegram.ext.ApplicationBuilder` instead.

Parameters

- **parse_mode** (str, optional) – 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.

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

Deprecated since version 20.8: Use `link_preview_options` instead. This parameter will be removed in future versions.

- `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`.
- `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.

Deprecated since version 20.8: Use `do_quote` instead. This parameter will be removed in future versions.

- `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`. If the `telegram.ext.JobQueue` is used, this must be a timezone provided by the `pytz` module. Defaults to `pytz.utc`, if available, and `datetime.timezone.utc` otherwise.
- `block` (bool, optional) – Default setting for the `BaseHandler.block` parameter of handlers and error handlers registered through `Application.add_handler()` and `Application.add_error_handler()`. Defaults to `True`.
- `protect_content` (bool, optional) – Protects the contents of the sent message from forwarding and saving.

New in version 20.0.

- `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)
```

(continues on next page)

(continued from previous page)

```

    )

    # 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")
    )

```

New 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.

New 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 `disable_web_page_preview`

Optional. Disables link previews for links in all outgoing messages.

Deprecated since version 20.8: Use `link_preview_options` instead. This attribute will be removed in future versions.

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.

New 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.

New 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.

New in version 20.0.

Type

`bool`

property 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.

Deprecated since version 20.8: Use `do_quote` instead. This attribute will be removed in future versions.

Type

`bool`

property quote_parse_mode

Optional. Alias for `parse_mode`, used for the corresponding parameter of `telegram.ReplyParameters()`.

Type

`str`

property tzinfo

A timezone to be used for all date(time) objects appearing throughout PTB.

Type

`tzinfo`

10.2.8 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

Use In

[telegram.ext.ApplicationBuilder.bot\(\)](#)

Available In

- [telegram.ext.Application.bot](#)
 - [telegram.ext.BasePersistence.bot](#)
 - [telegram.ext.CallbackContext.bot](#)
 - [telegram.ext.CallbackDataCache.bot](#)
 - [telegram.ext.Updater.bot](#)
-

See also:

[Arbitrary callback_data](#)

New 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.

New 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.

New in version 20.0.

async `shutdown()`

See `telegram.Bot.shutdown()`. Also shuts down the `ExtBot.rate_limiter` (if set) by calling `telegram.ext.BaseRateLimiter.shutdown()`.

10.2.9 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.

Available In

`telegram.ext.CallbackContext.job`

See also:

`Job Queue`

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.
New in version 20.0.
- **user_id** (*int*, optional) – User id of the user that this job is associated with.
New in version 20.0.

callback

The callback function that should be executed by the new job.

Type

`coroutine function`

data

Optional. Additional data needed for the `callback` function.

Type

`object`

name

Optional. The name of the new job.

Type

`str`

chat_id

Optional. Chat id of the chat that this job is associated with.

New in version 20.0.

Type

`int`

user_id

Optional. User id of the user that this job is associated with.

New in version 20.0.

Type

`int`

__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`.

New 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.

10.2.10 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

Use In

`telegram.ext.ApplicationBuilder.job_queue()`

Available In

- `telegram.ext.Application.job_queue`
 - `telegram.ext.CallbackContext.job_queue`
-

See also:

[Architecture Overview](#), [Job Queue](#)

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 *pending/scheduled* jobs with the given name that are currently in the *JobQueue*.

Returns

Tuple of all *pending* or *scheduled* jobs matching the name.

Return type

Tuple[*Job*]

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()`.

New in version 20.4.

Parameters

- *job_queue* (*JobQueue*) – The job queue that created the job.
- *job* (*Job*) – The job to run.

jobs()

Returns a tuple of all *scheduled* jobs that are currently in the *JobQueue*.

Returns

Tuple of all *scheduled* jobs.

Return type

Tuple[*Job*]

run_custom(*callback*, *job_kwargs*, *data=None*, *name=None*, *chat_id=None*, *user_id=None*)

Creates a new custom defined *Job*.

Parameters

- **callback** (coroutine function) – The callback function that should be executed by the new job. Callback signature:

```
async def callback(context: CallbackContext)
```

- **job_kwargs** (dict) – Arbitrary keyword arguments. Used as arguments for `apscheduler.schedulers.base.BaseScheduler.add_job()`.
- **data** (object, optional) – Additional data needed for the callback function. Can be accessed through `Job.data` in the callback. Defaults to `None`.

Changed in version 20.0: Renamed the parameter `context` to `data`.

- **name** (str, optional) – The name of the new job. Defaults to `callback.__name__`.
- **chat_id** (int, optional) – Chat id of the chat associated with this job. If passed, the corresponding `chat_data` will be available in the callback.

New in version 20.0.

- **user_id** (int, optional) – User id of the user associated with this job. If passed, the corresponding `user_data` will be available in the callback.

New in version 20.0.

Returns

The new *Job* instance that has been added to the job queue.

Return type

`telegram.ext.Job`

run_daily(*callback*, *time*, *days=(0, 1, 2, 3, 4, 5, 6)*, *data=None*, *name=None*, *chat_id=None*, *user_id=None*, *job_kwargs=None*)

Creates a new *Job* that runs on a daily basis and adds it to the queue.

Note: For a note about DST, please see the documentation of `APScheduler`.

Parameters

- **callback** (coroutine function) – The callback function that should be executed by the new job. Callback signature:

```
async def callback(context: CallbackContext)
```

- **time** (datetime.time) – Time of day at which the job should run. If the time-zone (datetime.time.tzinfo) is `None`, the default timezone of the bot will be used, which is UTC unless `telegram.ext.Defaults.tzinfo` is used.
- **days** (Tuple[int], optional) – Defines on which days of the week the job should run (where 0-6 correspond to sunday - saturday). By default, the job will run every day.

Changed in version 20.0: Changed day of the week mapping of 0-6 from monday-sunday to sunday-saturday.

- **data** (*object*, optional) – Additional data needed for the callback function. Can be accessed through `Job.data` in the callback. Defaults to `None`.

Changed in version 20.0: Renamed the parameter `context` to `data`.

- **name** (*str*, optional) – The name of the new job. Defaults to `callback.__name__`.
- **chat_id** (*int*, optional) – Chat id of the chat associated with this job. If passed, the corresponding `chat_data` will be available in the callback.

New in version 20.0.

- **user_id** (*int*, optional) – User id of the user associated with this job. If passed, the corresponding `user_data` will be available in the callback.

New in version 20.0.

- **job_kwargs** (*dict*, optional) – Arbitrary keyword arguments to pass to the `apscheduler.schedulers.base.BaseScheduler.add_job()`.

Returns

The new `Job` instance that has been added to the job queue.

Return type

`telegram.ext.Job`

run_monthly(*callback*, *when*, *day*, *data=None*, *name=None*, *chat_id=None*, *user_id=None*, *job_kwargs=None*)

Creates a new `Job` that runs on a monthly basis and adds it to the queue.

Changed in version 20.0: The `day_is_strict` argument was removed. Instead one can now pass `-1` to the `day` parameter to have the job run on the last day of the month.

Parameters

- **callback** (*coroutine function*) – The callback function that should be executed by the new job. Callback signature:

```
async def callback(context: CallbackContext)
```

- **when** (*datetime.time*) – Time of day at which the job should run. If the timezone (`when.tzinfo`) is `None`, the default timezone of the bot will be used, which is UTC unless `telegram.ext.Defaults.tzinfo` is used.
- **day** (*int*) – Defines the day of the month whereby the job would run. It should be within the range of 1 and 31, inclusive. If a month has fewer days than this number, the job will not run in this month. Passing `-1` leads to the job running on the last day of the month.
- **data** (*object*, optional) – Additional data needed for the callback function. Can be accessed through `Job.data` in the callback. Defaults to `None`.

Changed in version 20.0: Renamed the parameter `context` to `data`.

- **name** (*str*, optional) – The name of the new job. Defaults to `callback.__name__`.
- **chat_id** (*int*, optional) – Chat id of the chat associated with this job. If passed, the corresponding `chat_data` will be available in the callback.

New in version 20.0.

- **user_id** (*int*, optional) – User id of the user associated with this job. If passed, the corresponding `user_data` will be available in the callback.

New in version 20.0.

- **job_kwargs** (*dict*, optional) – Arbitrary keyword arguments to pass to the `apscheduler.schedulers.base.BaseScheduler.add_job()`.

Returns

The new *Job* instance that has been added to the job queue.

Return type

telegram.ext.Job

run_once(*callback*, *when*, *data=None*, *name=None*, *chat_id=None*, *user_id=None*, *job_kwargs=None*)

Creates a new *Job* instance that runs once and adds it to the queue.

Parameters

- **callback** (coroutine function) – The callback function that should be executed by the new job. Callback signature:

```
async def callback(context: CallbackContext)
```

- **when** (int | float | datetime.timedelta | datetime.datetime | datetime.time) – Time in or at which the job should run. This parameter will be interpreted depending on its type.
 - int or float will be interpreted as “seconds from now” in which the job should run.
 - datetime.timedelta will be interpreted as “time from now” in which the job should run.
 - datetime.datetime will be interpreted as a specific date and time at which the job should run. If the timezone (datetime.datetime.tzinfo) is None, the default timezone of the bot will be used, which is UTC unless telegram.ext.Defaults.tzinfo is used.
 - datetime.time will be interpreted as a specific time of day at which the job should run. This could be either today or, if the time has already passed, tomorrow. If the timezone (datetime.time.tzinfo) is None, the default timezone of the bot will be used, which is UTC unless telegram.ext.Defaults.tzinfo is used.
- **chat_id** (int, optional) – Chat id of the chat associated with this job. If passed, the corresponding *chat_data* will be available in the callback.
New in version 20.0.
- **user_id** (int, optional) – User id of the user associated with this job. If passed, the corresponding *user_data* will be available in the callback.
New in version 20.0.
- **data** (object, optional) – Additional data needed for the callback function. Can be accessed through *Job.data* in the callback. Defaults to None.
Changed in version 20.0: Renamed the parameter context to *data*.
- **name** (str, optional) – The name of the new job. Defaults to *callback.__name__*.
- **job_kwargs** (dict, optional) – Arbitrary keyword arguments to pass to the *apscheduler.schedulers.base.BaseScheduler.add_job()*.

Returns

The new *Job* instance that has been added to the job queue.

Return type

telegram.ext.Job

run_repeating(*callback*, *interval*, *first=None*, *last=None*, *data=None*, *name=None*, *chat_id=None*, *user_id=None*, *job_kwargs=None*)

Creates a new *Job* instance that runs at specified intervals and adds it to the queue.

Note: For a note about DST, please see the documentation of [APScheduler](#).

Parameters

- **callback** (coroutine function) – The callback function that should be executed by the new job. Callback signature:

```
async def callback(context: CallbackContext)
```

- **interval** (int | float | datetime.timedelta) – The interval in which the job will run. If it is an int or a float, it will be interpreted as seconds.
- **first** (int | float | datetime.timedelta | datetime.datetime | datetime.time, optional) – Time in or at which the job should run. This parameter will be interpreted depending on its type.
 - int or float will be interpreted as “seconds from now” in which the job should run.
 - datetime.timedelta will be interpreted as “time from now” in which the job should run.
 - datetime.datetime will be interpreted as a specific date and time at which the job should run. If the timezone (datetime.datetime.tzinfo) is None, the default timezone of the bot will be used.
 - datetime.time will be interpreted as a specific time of day at which the job should run. This could be either today or, if the time has already passed, tomorrow. If the timezone (datetime.time.tzinfo) is None, the default timezone of the bot will be used, which is UTC unless telegram.ext.Defaults.tzinfo is used.

Defaults to *interval*

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.

New in version 20.0.

- **user_id** (*int*, optional) – User id of the user associated with this job. If passed, the corresponding *user_data* will be available in the callback.

New in version 20.0.

- **job_kwargs** (*dict*, optional) – Arbitrary keyword arguments to pass to the `apscheduler.schedulers.base.BaseScheduler.add_job()`.

Returns

The new *Job* instance that has been added to the job queue.

Return type

telegram.ext.Job

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.

New 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`.

10.2.11 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](#)

New 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.

10.2.12 Updater

class telegram.ext.**Updater**(*bot, update_queue*)

Bases: [typing.AsyncContextManager](#)

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()
```

Use In

[telegram.ext.ApplicationBuilder.updater\(\)](#)

Available In

`telegram.ext.Application.updater`

See also:

`__aenter__()` and `__aexit__()`.

See also:

[Architecture Overview](#), [Builder Pattern](#)

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=10, bootstrap_retries=-1, read_timeout=None, write_timeout=None, connect_timeout=None, pool_timeout=None, allowed_updates=None, drop_pending_updates=None, error_callback=None*)

Starts polling updates from Telegram.

Changed in version 20.0: Removed the `clean` argument in favor of `drop_pending_updates`.

Parameters

- **poll_interval** (*float*, optional) – Time to wait between polling updates from Telegram in seconds. Default is `0.0`.
- **timeout** (*int*, optional) – Passed to `telegram.Bot.get_updates.timeout`. Defaults to `10` seconds.
- **bootstrap_retries** (*int*, optional) – Whether the bootstrapping phase of the `telegram.ext.Updater` will retry on failures on the Telegram server.
 - `< 0` - retry indefinitely (default)
 - `0` - no retries
 - `> 0` - retry up to X times

- **read_timeout** (*float*, optional) – Value to pass to `telegram.Bot.get_updates.read_timeout`. Defaults to `DEFAULT_NONE`.

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

Deprecated since version 20.7: Deprecated in favor of setting the timeout via `telegram.ext.ApplicationBuilder.get_updates_read_timeout()` or `telegram.Bot.get_updates_request`.

- **write_timeout** (*float | None*, optional) – Value to pass to `telegram.Bot.get_updates.write_timeout`. Defaults to `DEFAULT_NONE`.

Deprecated since version 20.7: Deprecated in favor of setting the timeout via `telegram.ext.ApplicationBuilder.get_updates_write_timeout()` or `telegram.Bot.get_updates_request`.

- **connect_timeout** (*float | None*, optional) – Value to pass to `telegram.Bot.get_updates.connect_timeout`. Defaults to `DEFAULT_NONE`.

Deprecated since version 20.7: Deprecated in favor of setting the timeout via `telegram.ext.ApplicationBuilder.get_updates_connect_timeout()` or `telegram.Bot.get_updates_request`.

- **pool_timeout** (*float | None*, optional) – Value to pass to `telegram.Bot.get_updates.pool_timeout`. Defaults to `DEFAULT_NONE`.

Deprecated since version 20.7: Deprecated in favor of setting the timeout via `telegram.ext.ApplicationBuilder.get_updates_pool_timeout()` or `telegram.Bot.get_updates_request`.

- **allowed_updates** (*List[str]*, optional) – Passed to `telegram.Bot.get_updates()`.

- **`drop_pending_updates`** (`bool`, optional) – Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is `False`.

New in version 13.4.

- **`error_callback`** (Callable[[`telegram.error.TelegramError`], `None`], optional) – Callback to handle `telegram.error.TelegramError`s that occur while calling `telegram.Bot.get_updates()` during polling. Defaults to `None`, in which case errors will be logged. Callback signature:

```
def callback(error: telegram.error.TelegramError)
```

Note: The `error_callback` must *not* be a `coroutine function`! If asynchronous behavior of the callback is wanted, please schedule a task from within the callback.

Returns

The update queue that can be filled from the main thread.

Return type

`asyncio.Queue`

Raises

`RuntimeError` – If the updater is already running or was not initialized.

```
async start_webhook(listen='127.0.0.1', port=80, url_path="", cert=None, key=None,
                    bootstrap_retries=0, webhook_url=None, allowed_updates=None,
                    drop_pending_updates=None, ip_address=None, max_connections=40,
                    secret_token=None, 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`.

New in version 13.4.

- **bootstrap_retries** (`int`, optional) – Whether the bootstrapping phase of the `telegram.ext.Updater` will retry on failures on the Telegram server.
 - `< 0` - retry indefinitely
 - `0` - no retries (default)
 - `> 0` - retry up to X times
- **webhook_url** (`str`, optional) – Explicitly specify the webhook url. Useful behind NAT, reverse proxy, etc. Default is derived from `listen`, `port`, `url_path`, `cert`, and `key`.
- **ip_address** (`str`, optional) – Passed to `telegram.Bot.set_webhook()`. Defaults to `None`.

New in version 13.4.

- **allowed_updates** (`List[str]`, optional) – Passed to `telegram.Bot.set_webhook()`. Defaults to `None`.
- **max_connections** (`int`, optional) – Passed to `telegram.Bot.set_webhook()`. Defaults to `40`.

New in version 13.6.

- **secret_token** (`str`, optional) – Passed to `telegram.Bot.set_webhook()`. Defaults to `None`.

When added, the web server started by this call will expect the token to be set in the `X-Telegram-Bot-API-Secret-Token` header of an incoming request and will raise a `http.HTTPStatus.FORBIDDEN` error if either the header isn't set or it is set to a wrong token.

New in version 20.0.

- **unix** (`pathlib.Path` | `str`, optional) – Path to the unix socket file. Path does not need to exist, in which case the file will be created.

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`.

New in version 20.8.

Returns

The update queue that can be filled from the main thread.

Return type

`queue.Queue`

Raises

RuntimeError – If the updater is already running or was not initialized.

`async stop()`

Stops the polling/webhook.

See also:

`start_polling()`, `start_webhook()`

Raises

RuntimeError – If the updater is not running.

10.2.13 Handlers

BaseHandler

class telegram.ext.**BaseHandler**(*callback*, *block=True*)

Bases: `typing.Generic`, `ABC`

The base class for all update handlers. Create custom handlers by inheriting from it.

Warning: When setting *block* to `False`, you cannot rely on adding custom attributes to `telegram.ext.CallbackContext`. See its docs for more info.

This class is a `Generic` class and accepts two type variables:

1. The type of the updates that this handler will handle. Must coincide with the type of the first argument of *callback*. *check_update()* must only accept updates of this type.
2. The type of the second argument of *callback*. Must coincide with the type of the parameters *handle_update.context* and *collect_additional_context.context* as well as the second argument of *callback*. Must be either `CallbackContext` or a subclass of that class.

Tip: For this type variable, one should usually provide a `TypeVar` that is also used for the mentioned method arguments. That way, a type checker can check whether this handler fits the definition of the *Application*.

Available In

`telegram.ext.Application.handlers`

See also:

Types of Handlers

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](#)**abstract check_update(update)**

This method is called to determine if an update should be handled by this handler instance. It should always be overridden.

Note: Custom updates types can be handled by the application. Therefore, an implementation of this method should always check the type of [update](#).

Parameters

[update](#) ([object](#) | [telegram.Update](#)) – The update to be tested.

Returns

Either [None](#) or [False](#) if the update should not be handled. Otherwise an object that will be passed to [handle_update\(\)](#) and [collect_additional_context\(\)](#) when the update gets handled.

collect_additional_context(context, update, application, check_result)

Prepares additional arguments for the context. Override if needed.

Parameters

- [context](#) ([telegram.ext.CallbackContext](#)) – The context object.
- [update](#) ([telegram.Update](#)) – The update to gather chat/user id from.
- [application](#) ([telegram.ext.Application](#)) – The calling application.
- [check_result](#) – The result (return value) from [check_update\(\)](#).

async handle_update(update, application, check_result, context)

This method is called if it was determined that an update should indeed be handled by this instance. Calls [callback](#) along with its respectful arguments. To work with the [telegram.ext.ConversationHandler](#), this method returns the value returned from [callback](#). Note that it can be overridden if needed by the subclassing handler.

Parameters

- [update](#) ([str](#) | [telegram.Update](#)) – The update to be handled.
- [application](#) ([telegram.ext.Application](#)) – The calling application.

- `check_result` (object) – The result from `check_update()`.
- `context` (`telegram.ext.CallbackContext`) – The context as provided by the application.

CallbackQueryHandler

`class telegram.ext.CallbackQueryHandler(callback, 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`.

New in version 13.6.

Warning: When setting `block` to `False`, you cannot rely on adding custom attributes to `telegram.ext.CallbackContext`. See its docs for more info.

Parameters

- `callback` (coroutine function) – The callback function for this handler. Will be called when `check_update()` has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

The return value of the callback is usually ignored except for the special case of `telegram.ext.ConversationHandler`.

- `pattern` (`str` | `re.Pattern` | callable | `type`, optional) – Pattern to test `telegram.CallbackQuery.data` against. If a string or a regex pattern is passed, `re.match()` is used on `telegram.CallbackQuery.data` to determine if an update should be handled by this handler. If your bot allows arbitrary objects as `callback_data`, non-strings will be accepted. To filter arbitrary objects you may pass:
 - a callable, accepting exactly one argument, namely the `telegram.CallbackQuery.data`. It must return `True` or `False/None` to indicate, whether the update should be handled.
 - a `type`. If `telegram.CallbackQuery.data` is an instance of that type (or a subclass), the update will be handled.

If `telegram.CallbackQuery.data` is `None`, the `telegram.CallbackQuery` update will not be handled.

See also:

Arbitrary `callback_data`

Changed in version 13.6: Added support for arbitrary callback data.

- **block** (`bool`, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in `telegram.ext.Application.process_update()`. Defaults to `True`.

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`

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`

Available In

`telegram.ext.Application.handlers`

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

New 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.

Available In

telegram.ext.Application.handlers

New 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.

New 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.

Available In

`telegram.ext.Application.handlers`

New in version 13.8.

Parameters

- **callback** (`coroutine function`) – The callback function for this handler. Will be called when `check_update()` has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

The return value of the callback is usually ignored except for the special case of `telegram.ext.ConversationHandler`.

- **chat_id** (`int` | `Collection[int]`, optional) – Filters requests to allow only those which are asking to join the specified chat ID(s).

New in version 20.0.

- **username** (`str` | `Collection[str]`, optional) – Filters requests to allow only those which are asking to join the specified username(s).

New 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)`

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

Available In

`telegram.ext.Application.handlers`

New in version 13.4.

Parameters

- **callback** (coroutine function) – The callback function for this handler. Will be called when `check_update()` has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

The return value of the callback is usually ignored except for the special case of `telegram.ext.ConversationHandler`.

- **`chat_member_types`** (`int`, optional) – Pass one of `MY_CHAT_MEMBER`, `CHAT_MEMBER` or `ANY_CHAT_MEMBER` to specify if this handler should handle only updates with `telegram.Update.my_chat_member`, `telegram.Update.chat_member` or both. Defaults to `MY_CHAT_MEMBER`.
- **`block`** (`bool`, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in `telegram.ext.Application.process_update()`. Defaults to `True`.

See also:

Concurrency

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.

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](#).

New in version 13.6.

callback

The callback function for this handler.

Type

[coroutine function](#)

block

Determines whether the return value of the callback should be awaited before processing the next handler in [telegram.ext.Application.process_update\(\)](#).

Type

[bool](#)

pattern

Optional. Regex pattern to test [telegram.ChosenInlineResult.result_id](#) against.

New in version 13.6.

Type

[Pattern](#)

Available In

[telegram.ext.Application.handlers](#)

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 /, optionally followed by an @ and the bot's name and/or some additional text. The handler will add a *list* to the *CallbackContext* named *CallbackContext.args*. It will contain a list of strings, which is the text following the command split on single or consecutive whitespace characters.

By default, the handler listens to messages as well as edited messages. To change this behavior use *~filters.UpdateType.EDITED_MESSAGE* in the filter argument.

Note: *CommandHandler* does *not* handle (edited) channel posts and does *not* handle commands that are part of a caption. Please use *MessageHandler* with a suitable combination of filters (e.g. *telegram.ext.filters.UpdateType.CHANNEL_POSTS*, *telegram.ext.filters.CAPTION* and *telegram.ext.filters.Regex*) to handle those messages.

Warning: When setting *block* to *False*, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs for more info.

Examples

- *Timer Bot*
 - *Error Handler Bot*
-

Available In

telegram.ext.Application.handlers

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.

New 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.

New 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*
-

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`

Available In

`telegram.ext.Application.handlers`

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(...)`.

`telegram.ext.filters.ALL = filters.ALL`

All Messages.

`telegram.ext.filters.ANIMATION = filters.ANIMATION`

Messages that contain `telegram.Message.animation`.

`telegram.ext.filters.ATTACHMENT = filters.ATTACHMENT`

Messages that contain `telegram.Message.effective_attachment()`.

New in version 13.6.

`telegram.ext.filters.AUDIO = filters.AUDIO`

Messages that contain `telegram.Message.audio`.

`telegram.ext.filters.CAPTION = filters.CAPTION`

Shortcut for `telegram.ext.filters.Caption()`.

Examples

To allow any caption, simply use `MessageHandler(filters.CAPTION, callback_method)`.

`telegram.ext.filters.CHAT = filters.CHAT`

This filter filters *any* message that has a `telegram.Message.chat`.

Deprecated since version 20.8: This filter has no effect since `telegram.Message.chat` is always present.

`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.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`.

New in version 20.0.

`telegram.ext.filters.HAS_PROTECTED_CONTENT = filters.HAS_PROTECTED_CONTENT`

Messages that contain `telegram.Message.has_protected_content`.

New in version 13.9.

`telegram.ext.filters.INVOICE = filters.INVOICE`

Messages that contain `telegram.Message.invoice`.

`telegram.ext.filters.IS_AUTOMATIC_FORWARD = filters.IS_AUTOMATIC_FORWARD`

Messages that contain `telegram.Message.is_automatic_forward`.

New in version 13.9.

`telegram.ext.filters.IS_TOPIC_MESSAGE = filters.IS_TOPIC_MESSAGE`

Messages that contain `telegram.Message.is_topic_message`.

New in version 20.0.

`telegram.ext.filters.LOCATION = filters.LOCATION`

Messages that contain `telegram.Message.location`.

`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`.

New in version 20.0.

`telegram.ext.filters.REPLY = filters.REPLY`

Messages that contain `telegram.Message.reply_to_message`.

`telegram.ext.filters.STORY = filters.STORY`

Messages that contain `telegram.Message.story`.

New in version 20.5.

`telegram.ext.filters.SUCCESSFUL_PAYMENT = filters.SUCCESSFUL_PAYMENT`

Messages that contain `telegram.Message.successful_payment`.

`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`.

New 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.

`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`
-

New in version 20.0: Added the arguments `name` and `data_filter`.

Parameters

- **`name`** (`str`) – Name for this filter. Defaults to the type of filter.
- **`data_filter`** (`bool`) – Whether this filter is a data filter. A data filter should return a dict with lists. The dict will be merged with `telegram.ext.CallbackContext`'s internal dict in most cases (depends on the handler).

`__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.

Parameters

update (*telegram.Update*) – The update to check.

Returns

`True` if the update contains one of *channel_post*, *message*, *edited_channel_post* or *edited_message*, `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 completely 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 completely replace the current set of allowed users.

Returns

`frozenset(str)`

class telegram.ext.filters.ChatType

Bases: `object`

Subset for filtering the type of chat.

Examples

Use these filters like: `filters.ChatType.CHANNEL` or `filters.ChatType.SUPERGROUP` etc.

Caution: `filters.ChatType` itself is *not* a filter, but just a convenience namespace.

CHANNEL = `filters.ChatType.CHANNEL`

Updates from channel.

GROUP = `filters.ChatType.GROUP`

Updates from group.

GROUPS = `filters.ChatType.GROUPS`

Update from group *or* supergroup.

PRIVATE = `filters.ChatType.PRIVATE`

Update from private chats.

SUPERGROUP = `filters.ChatType.SUPERGROUP`

Updates from supergroup.

class `telegram.ext.filters.Command(only_start=True)`

Bases: `telegram.ext.filters.MessageFilter`

Messages with a `telegram.MessageEntity.BOT_COMMAND`. By default, only allows messages *starting* with a bot command. Pass `False` to also allow messages that contain a bot command *anywhere* in the text.

Examples

`MessageHandler(filters.Command(False), command_anywhere_callback)`

See also:

`telegram.ext.filters.COMMAND`.

Note: `telegram.ext.filters.TEXT` also accepts messages containing a command.

Parameters

only_start (`bool`, optional) – Whether to only allow messages that *start* with a bot command. Defaults to `True`.

class `telegram.ext.filters.Dice(values=None, emoji=None)`

Bases: `telegram.ext.filters.MessageFilter`

Dice Messages. If an integer or a list of integers is passed, it filters messages to only allow those whose dice value is appearing in the given list.

New in version 13.4.

Examples

To allow any dice message, simply use `MessageHandler(filters.Dice.ALL, callback_method)`.

To allow any dice message, but with value 3 *or* 4, use `MessageHandler(filters.Dice([3, 4]), callback_method)`

To allow only dice messages with the emoji , but any value, use `MessageHandler(filters.Dice.DICE, callback_method)`.

To allow only dice messages with the emoji and with value 6, use `MessageHandler(filters.Dice.Darts(6), callback_method)`.

To allow only dice messages with the emoji and with value 5 *or* 6, use `MessageHandler(filters.Dice.Football([5, 6]), callback_method)`.

Note: Dice messages don't have text. If you want to filter either text or dice messages, use `filters.TEXT` | `filters.Dice.ALL`.

Parameters

values (`int` | `Collection[int]`, optional) – Which values to allow. If not specified, will allow the specified dice message.

ALL = `filters.Dice.ALL`

Dice messages with any value and any emoji.

class `Basketball(values)`

Bases: `telegram.ext.filters.MessageFilter`

Dice messages with the emoji . Supports passing a list of integers.

Parameters

values (`int` | `Collection[int]`) – Which values to allow.

BASKETBALL = `filters.Dice.BASKETBALL`

Dice messages with the emoji . Matches any dice value.

class `Bowling(values)`

Bases: `telegram.ext.filters.MessageFilter`

Dice messages with the emoji . Supports passing a list of integers.

Parameters

values (`int` | `Collection[int]`) – Which values to allow.

BOWLING = `filters.Dice.BOWLING`

Dice messages with the emoji . Matches any dice value.

class `Darts(values)`

Bases: `telegram.ext.filters.MessageFilter`

Dice messages with the emoji . Supports passing a list of integers.

Parameters

values (`int` | `Collection[int]`) – Which values to allow.

DARTS = `filters.Dice.DARTS`

Dice messages with the emoji . Matches any dice value.

class `Dice(values)`

Bases: `telegram.ext.filters.MessageFilter`

Dice messages with the emoji . Supports passing a list of integers.

Parameters

values (`int` | `Collection[int]`) – Which values to allow.

DICE = `filters.Dice.DICE`

Dice messages with the emoji . Matches any dice value.

class `Football(values)`

Bases: `telegram.ext.filters.MessageFilter`

Dice messages with the emoji . Supports passing a list of integers.

Parameters

values (`int` | `Collection[int]`) – Which values to allow.

FOOTBALL = `filters.Dice.FOOTBALL`

Dice messages with the emoji . Matches any dice value.

class `SlotMachine(values)`

Bases: `telegram.ext.filters.MessageFilter`

Dice messages with the emoji . Supports passing a list of integers.

Parameters

values (`int` | `Collection[int]`) – Which values to allow.

SLOT_MACHINE = `filters.Dice.SLOT_MACHINE`

Dice messages with the emoji . Matches any dice value.

class telegram.ext.filters.Document

Bases: `object`

Subset for messages containing a document/file.

Examples

Use these filters like: `filters.Document.MP3`, `filters.Document.MimeType("text/plain")` etc. Or just use `filters.Document.ALL` for all document messages.

Caution: `filters.Document` itself is *not* a filter, but just a convenience namespace.

ALL = `filters.Document.ALL`

Messages that contain a `telegram.Message.document`.

class Category(*category*)

Bases: `telegram.ext.filters.MessageFilter`

Filters documents by their category in the mime-type attribute.

Parameters

category (`str`) – Category of the media you want to filter.

Example

`filters.Document.Category('audio/')` returns `True` for all types of audio sent as a file, for example 'audio/mpeg' or 'audio/x-wav'.

Note: This Filter only filters by the `mime_type` of the document, it doesn't check the validity of the document. The user can manipulate the mime-type of a message and send media with wrong types that don't fit to this handler.

APPLICATION = `filters.Document.Category('application/')`

Use as `filters.Document.APPLICATION`.

AUDIO = `filters.Document.Category('audio/')`

Use as `filters.Document.AUDIO`.

IMAGE = `filters.Document.Category('image/')`

Use as `filters.Document.IMAGE`.

VIDEO = `filters.Document.Category('video/')`

Use as `filters.Document.VIDEO`.

TEXT = `filters.Document.Category('text/')`

Use as `filters.Document.TEXT`.

class FileExtension(*file_extension*, *case_sensitive=False*)

Bases: `telegram.ext.filters.MessageFilter`

This filter filters documents by their file ending/extension.

Parameters

- **file_extension** (`str` | `None`) – Media file extension you want to filter.
- **case_sensitive** (`bool`, optional) – Pass `True` to make the filter case sensitive. Default: `False`.

Example

- `filters.Document.FileExtension("jpg")` filters files with extension ".jpg".
 - `filters.Document.FileExtension(".jpg")` filters files with extension "..jpg".
 - `filters.Document.FileExtension("Dockerfile", case_sensitive=True)` filters files with extension ".Dockerfile" minding the case.
 - `filters.Document.FileExtension(None)` filters files without a dot in the filename.
-

Note:

- This Filter only filters by the file ending/extension of the document, it doesn't check the validity of document.
 - The user can manipulate the file extension of a document and send media with wrong types that don't fit to this handler.
 - Case insensitive by default, you may change this with the flag `case_sensitive=True`.
 - Extension should be passed without leading dot unless it's a part of the extension.
 - Pass `None` to filter files with no extension, i.e. without a dot in the filename.
-

class `MimeType`(*mimetype*)

Bases: `telegram.ext.filters.MessageFilter`

This Filter filters documents by their mime-type attribute.

Parameters

mimetype (*str*) – The mimetype to filter.

Example

`filters.Document.MimeType('audio/mpeg')` filters all audio in *.mp3* format.

Note: This Filter only filters by the `mime_type` of the document, it doesn't check the validity of document. The user can manipulate the mime-type of a message and send media with wrong types that don't fit to this handler.

APK = `filters.Document.MimeType('application/vnd.android.package-archive')`

Use as `filters.Document.APK`.

DOC = `filters.Document.MimeType('application/msword')`

Use as `filters.Document.DOC`.

DOCX = `filters.Document.MimeType('application/vnd.openxmlformats-officedocument.wordprocessingml.document')`

Use as `filters.Document.DOCX`.

EXE = `filters.Document.MimeType('application/octet-stream')`

Use as `filters.Document.EXE`.

MP4 = `filters.Document.MimeType('video/mp4')`

Use as `filters.Document.MP4`.

GIF = `filters.Document.MimeType('image/gif')`

Use as `filters.Document.GIF`.

JPG = `filters.Document.MimeType('image/jpeg')`

Use as `filters.Document.JPG`.

MP3 = `filters.Document.MimeType('audio/mpeg')`

Use as `filters.Document.MP3`.

PDF = `filters.Document.MimeType('application/pdf')`

Use as `filters.Document.PDF`.

PY = `filters.Document.MimeType('text/x-python')`

Use as `filters.Document.PY`.

SVG = `filters.Document.MimeType('image/svg+xml')`

Use as `filters.Document.SVG`.

TXT = `filters.Document.MimeType('text/plain')`

Use as `filters.Document.TXT`.

TARGZ = `filters.Document.MimeType('application/x-compressed-tar')`

Use as `filters.Document.TARGZ`.

WAV = `filters.Document.MimeType('audio/x-wav')`

Use as `filters.Document.WAV`.

XML = `filters.Document.MimeType('text/xml')`

Use as `filters.Document.XML`.

ZIP = `filters.Document.MimeType('application/zip')`

Use as `filters.Document.ZIP`.

class `telegram.ext.filters.Entity(entity_type)`

Bases: `telegram.ext.filters.MessageFilter`

Filters messages to only allow those which have a `telegram.MessageEntity` where their *type* matches *entity_type*.

Examples

`MessageHandler(filters.Entity("hashtag"), callback_method)`

Parameters

entity_type (*str*) – Entity type to check for. All types can be found as constants in `telegram.MessageEntity`.

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`

New 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 completely 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 completely 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)
```

New 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.

Available In

- [telegram.ext.CommandHandler.filters](#)
 - [telegram.ext.MessageHandler.filters](#)
 - [telegram.ext.PrefixHandler.filters](#)
-

See also:

[Advanced 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](#)

abstract 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 completely 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 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.

ALL = filters.StatusUpdate.ALL

Messages that contain any of the below.

CHAT_CREATED = filters.StatusUpdate.CHAT_CREATED

Messages that contain [telegram.Message.group_chat_created](#), [telegram.Message.supergroup_chat_created](#) or [telegram.Message.channel_chat_created](#).

CHAT_SHARED = filters.StatusUpdate.CHAT_SHARED

Messages that contain [telegram.Message.chat_shared](#).

New in version 20.1.

CONNECTED_WEBSITE = filters.StatusUpdate.CONNECTED_WEBSITE

Messages that contain [telegram.Message.connected_website](#).

DELETE_CHAT_PHOTO = filters.StatusUpdate.DELETE_CHAT_PHOTO

Messages that contain [telegram.Message.delete_chat_photo](#).

FORUM_TOPIC_CLOSED = filters.StatusUpdate.FORUM_TOPIC_CLOSED

Messages that contain [telegram.Message.forum_topic_closed](#).

New in version 20.0.

FORUM_TOPIC_CREATED = filters.StatusUpdate.FORUM_TOPIC_CREATED

Messages that contain [telegram.Message.forum_topic_created](#).

New in version 20.0.

FORUM_TOPIC_EDITED = filters.StatusUpdate.FORUM_TOPIC_EDITED

Messages that contain [telegram.Message.forum_topic_edited](#).

New in version 20.0.

FORUM_TOPIC_REOPENED = filters.StatusUpdate.FORUM_TOPIC_REOPENED

Messages that contain [telegram.Message.forum_topic_reopened](#).

New in version 20.0.

GENERAL_FORUM_TOPIC_HIDDEN = filters.StatusUpdate.GENERAL_FORUM_TOPIC_HIDDEN

Messages that contain [telegram.Message.general_forum_topic_hidden](#).

New in version 20.0.

GENERAL_FORUM_TOPIC_UNHIDDEN = `filters.StatusUpdate.GENERAL_FORUM_TOPIC_UNHIDDEN`

Messages that contain `telegram.Message.general_forum_topic_unhidden`.

New in version 20.0.

GIVEAWAY_CREATED = `filters.StatusUpdate.GIVEAWAY_CREATED`

Messages that contain `telegram.Message.giveaway_created`.

New 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`.

MESSAGE_AUTO_DELETE_TIMER_CHANGED =
`filters.StatusUpdate.MESSAGE_AUTO_DELETE_TIMER_CHANGED`

Messages that contain `telegram.Message.message_auto_delete_timer_changed`

New in version 13.4.

MIGRATE = `filters.StatusUpdate.MIGRATE`

Messages that contain `telegram.Message.migrate_from_chat_id` or `telegram.Message.migrate_to_chat_id`.

NEW_CHAT_MEMBERS = `filters.StatusUpdate.NEW_CHAT_MEMBERS`

Messages that contain `telegram.Message.new_chat_members`.

NEW_CHAT_PHOTO = `filters.StatusUpdate.NEW_CHAT_PHOTO`

Messages that contain `telegram.Message.new_chat_photo`.

NEW_CHAT_TITLE = `filters.StatusUpdate.NEW_CHAT_TITLE`

Messages that contain `telegram.Message.new_chat_title`.

PINNED_MESSAGE = `filters.StatusUpdate.PINNED_MESSAGE`

Messages that contain `telegram.Message.pinned_message`.

PROXIMITY_ALERT_TRIGGERED = `filters.StatusUpdate.PROXIMITY_ALERT_TRIGGERED`

Messages that contain `telegram.Message.proximity_alert_triggered`.

USER_SHARED = `filters.StatusUpdate.USER_SHARED`

Messages that contain `telegram.Message.user_shared`.

Warning: This will only catch the legacy `telegram.Message.user_shared` attribute, not the new `telegram.Message.users_shared` attribute!

New in version 20.1.

Deprecated since version 20.8: Use `USERS_SHARED` instead.

USERS_SHARED = `filters.StatusUpdate.USERS_SHARED`

Messages that contain `telegram.Message.users_shared`.

New in version 20.8.

VIDEO_CHAT_ENDED = `filters.StatusUpdate.VIDEO_CHAT_ENDED`

Messages that contain `telegram.Message.video_chat_ended`.

New in version 13.4.

Changed in version 20.0: This filter was formerly named `VOICE_CHAT_ENDED`

VIDEO_CHAT_SCHEDULED = filters.StatusUpdate.VIDEO_CHAT_SCHEDULED

Messages that contain *telegram.Message.video_chat_scheduled*.

New in version 13.5.

Changed in version 20.0: This filter was formerly named VOICE_CHAT_SCHEDULED

VIDEO_CHAT_STARTED = filters.StatusUpdate.VIDEO_CHAT_STARTED

Messages that contain *telegram.Message.video_chat_started*.

New in version 13.4.

Changed in version 20.0: This filter was formerly named VOICE_CHAT_STARTED

VIDEO_CHAT_PARTICIPANTS_INVITED = filters.StatusUpdate.VIDEO_CHAT_PARTICIPANTS_INVITED

Messages that contain *telegram.Message.video_chat_participants_invited*.

New in version 13.4.

Changed in version 20.0: This filter was formerly named VOICE_CHAT_PARTICIPANTS_INVITED

WEB_APP_DATA = filters.StatusUpdate.WEB_APP_DATA

Messages that contain *telegram.Message.web_app_data*.

New in version 20.0.

WRITE_ACCESS_ALLOWED = filters.StatusUpdate.WRITE_ACCESS_ALLOWED

Messages that contain *telegram.Message.write_access_allowed*.

New 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: <code>filters.Sticker</code> itself is <i>not</i> a filter, but just a convenience namespace.

ALL = filters.Sticker.ALL

Messages that contain *telegram.Message.sticker*.

ANIMATED = filters.Sticker.ANIMATED

Messages that contain *telegram.Message.sticker* and *is animated*.

New in version 20.0.

STATIC = filters.Sticker.STATIC

Messages that contain *telegram.Message.sticker* and is a static sticker, i.e. does not contain *telegram.Sticker.is_animated* or *telegram.Sticker.is_video*.

New in version 20.0.

VIDEO = filters.Sticker.VIDEO

Messages that contain *telegram.Message.sticker* and is a *video sticker*.

New in version 20.0.

PREMIUM = filters.Sticker.PREMIUM

Messages that contain `telegram.Message.sticker` and have a `premium animation`.

New in version 20.0.

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.

New 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`

abstract filter(update)

This method must be overwritten.

Parameters

update (`telegram.Update`) – The update that is tested.

Returns

`dict` or `bool`.

class telegram.ext.filters.UpdateType

Bases: `object`

Subset for filtering the type of update.

Examples

Use these filters like: `filters.UpdateType.MESSAGE` or `filters.UpdateType.CHANNEL_POSTS` etc.

Caution: `filters.UpdateType` itself is *not* a filter, but just a convenience namespace.

CHANNEL_POST = `filters.UpdateType.CHANNEL_POST`

Updates with `telegram.Update.channel_post`.

CHANNEL_POSTS = `filters.UpdateType.CHANNEL_POSTS`

Updates with either `telegram.Update.channel_post` or `telegram.Update.edited_channel_post`.

EDITED = `filters.UpdateType.EDITED`

Updates with either `telegram.Update.edited_message` or `telegram.Update.edited_channel_post`.

New in version 20.0.

EDITED_CHANNEL_POST = `filters.UpdateType.EDITED_CHANNEL_POST`

Updates with `telegram.Update.edited_channel_post`.

EDITED_MESSAGE = `filters.UpdateType.EDITED_MESSAGE`

Updates with `telegram.Update.edited_message`.

MESSAGE = `filters.UpdateType.MESSAGE`

Updates with `telegram.Update.message`.

MESSAGES = `filters.UpdateType.MESSAGES`

Updates with either `telegram.Update.message` or `telegram.Update.edited_message`.

class `telegram.ext.filters.User`(*user_id=None, username=None, allow_empty=False*)

Bases: `telegram.ext.filters.MessageFilter`

Filters messages to allow only those which are from specified user ID(s) or username(s).

Examples

`MessageHandler(filters.User(1234), callback_method)`

Parameters

- **`user_id`** (`int` | `Collection[int]`, optional) – Which user ID(s) to allow through.
- **`username`** (`str` | `Collection[str]`, optional) – Which username(s) to allow through. Leading '@' s in usernames will be discarded.
- **`allow_empty`** (`bool`, optional) – Whether updates should be processed, if no user is specified in `user_ids` and `usernames`. Defaults to `False`.

Raises

`RuntimeError` – If `user_id` and `username` are both present.

`allow_empty`

Whether updates should be processed, if no user is specified in `user_ids` and `usernames`.

Type

`bool`

`add_usernames`(*username*)

Add one or more chats to the allowed usernames.

Parameters

`username` (`str` | `Collection[str]`) – Which username(s) to allow through. Leading '@' s in usernames will be discarded.

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 completely 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 completely 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)`

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 completely 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 completely 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

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`.

New in version 13.5.

callback

The callback function for this handler.

Type

coroutine function

pattern

Optional. Regex pattern to test `telegram.InlineQuery.query` against.

Type

`str` | `re.Pattern`

chat_types

Optional. List of allowed chat types.

New in version 13.5.

Type

List[`str`]

block

Determines whether the return value of the callback should be awaited before processing the next handler in `telegram.ext.Application.process_update()`.

Type

`bool`

Available In

`telegram.ext.Application.handlers`

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.

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.

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

Available In

`telegram.ext.Application.handlers`

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.

Available In

[telegram.ext.Application.handlers](#)

New 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`

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

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](#)

Available In

[telegram.ext.Application.handlers](#)

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

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](#)

Available In

[telegram.ext.Application.handlers](#)

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

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.

New 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.

New in version 20.8.

Type

[str](#) | `re.Pattern`, optional

Available In

`telegram.ext.Application.handlers`

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.

Available In

`telegram.ext.Application.handlers`

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

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`

Available In

`telegram.ext.Application.handlers`

`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.

Parameters

- `command` (`str`) – The command this handler should listen for.
- `callback` (coroutine function) – The callback function for this handler. Will be called when `check_update()` has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

The return value of the callback is usually ignored except for the special case of `telegram.ext.ConversationHandler`.

- `block` (`bool`, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in `telegram.ext.Application.process_update()`. Defaults to `True`.

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`

Available In

`telegram.ext.Application.handlers`

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.

Parameters

- **pattern** (`str` | `re.Pattern`) – The regex pattern.
- **callback** (coroutine function) – The callback function for this handler. Will be called when `check_update()` has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

The return value of the callback is usually ignored except for the special case of `telegram.ext.ConversationHandler`.

- **block** (`bool`, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in `telegram.ext.Application.process_update()`. Defaults to `True`.

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`

Available In

`telegram.ext.Application.handlers`

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.

Parameters

- **type** (`type`) – The `type` of updates this handler should process, as determined by `isinstance`
- **callback** (`coroutine function`) – The callback function for this handler. Will be called when `check_update()` has determined that an update should be processed by this handler. Callback signature:

```
async def callback(update: Update, context: CallbackContext)
```

The return value of the callback is usually ignored except for the special case of `telegram.ext.ConversationHandler`.

- **strict** (`bool`, optional) – Use `type` instead of `isinstance`. Default is `False`.
- **block** (`bool`, optional) – Determines whether the return value of the callback should be awaited before processing the next handler in `telegram.ext.Application.process_update()`. Defaults to `True`.

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`

Available In

`telegram.ext.Application.handlers`

check_update(update)

Determines whether an update should be passed to this handler's `callback`.

Parameters

update (`object`) – Incoming update.

Returns

`bool`

10.2.14 Persistence

BasePersistence

class telegram.ext.**BasePersistence**(*store_data=None, update_interval=60*)

Bases: [typing.Generic](#), [ABC](#)

Interface class for adding persistence to your bot. Subclass this object for different implementations of a persistent bot.

Attention: The interface provided by this class is intended to be accessed exclusively by [Application](#). Calling any of the methods below manually might interfere with the integration of persistence into [Application](#).

All relevant methods must be overwritten. This includes:

- [get_bot_data\(\)](#)
- [update_bot_data\(\)](#)
- [refresh_bot_data\(\)](#)
- [get_chat_data\(\)](#)
- [update_chat_data\(\)](#)
- [refresh_chat_data\(\)](#)
- [drop_chat_data\(\)](#)
- [get_user_data\(\)](#)
- [update_user_data\(\)](#)
- [refresh_user_data\(\)](#)
- [drop_user_data\(\)](#)
- [get_callback_data\(\)](#)
- [update_callback_data\(\)](#)
- [get_conversations\(\)](#)
- [update_conversation\(\)](#)
- [flush\(\)](#)

If you don't actually need one of those methods, a simple `pass` is enough. For example, if you don't store `bot_data`, you don't need [get_bot_data\(\)](#), [update_bot_data\(\)](#) or [refresh_bot_data\(\)](#).

Note: You should avoid saving [telegram.Bot](#) instances. This is because if you change e.g. the bots token, this won't propagate to the serialized instances and may lead to exceptions.

To prevent this, the implementation may use `bot` to replace bot instances with a placeholder before serialization and insert `bot` back when loading the data. Since `bot` will be set when the process starts, this will be the up-to-date bot instance.

If the persistence implementation does not take care of this, you should make sure not to store any bot instances in the data that will be persisted. E.g. in case of [telegram.TelegramObject](#), one may call [set_bot\(\)](#) to ensure that shortcuts like [telegram.Message.reply_text\(\)](#) are available.

This class is a [Generic](#) class and accepts three type variables:

1. The type of the second argument of `update_user_data()`, which must coincide with the type of the second argument of `refresh_user_data()` and the values in the dictionary returned by `get_user_data()`.
2. The type of the second argument of `update_chat_data()`, which must coincide with the type of the second argument of `refresh_chat_data()` and the values in the dictionary returned by `get_chat_data()`.
3. The type of the argument of `update_bot_data()`, which must coincide with the type of the argument of `refresh_bot_data()` and the return value of `get_bot_data()`.

Use In

`telegram.ext.ApplicationBuilder.persistence()`

Available In

`telegram.ext.Application.persistence`

See also:

[Architecture Overview, Making Your Bot Persistent](#)

Changed in version 20.0:

- The parameters and attributes `store_*_data` were replaced by `store_data`.
- `insert/replace_bot` was dropped. Serialization of bot instances now needs to be handled by the specific implementation - see above note.

Parameters

- **`store_data`** (*PersistenceInput*, optional) – Specifies which kinds of data will be saved by this persistence instance. By default, all available kinds of data will be saved.
- **`update_interval`** (*int | float*, optional) – The *Application* will update the persistence in regular intervals. This parameter specifies the time (in seconds) to wait between two consecutive runs of updating the persistence. Defaults to 60 seconds.

New in version 20.0.

store_data

Specifies which kinds of data will be saved by this persistence instance.

Type

PersistenceInput

bot

The bot associated with the persistence.

Type

telegram.Bot

abstract async drop_chat_data(chat_id)

Will be called by the *telegram.ext.Application*, when using `drop_chat_data()`.

New in version 20.0.

Parameters

`chat_id` (*int*) – The chat id to delete from the persistence.

abstract async drop_user_data(*user_id*)

Will be called by the `telegram.ext.Application`, when using `drop_user_data()`.

New in version 20.0.

Parameters

user_id (*int*) – The user id to delete from the persistence.

abstract async flush()

Will be called by `telegram.ext.Application.stop()`. Gives the persistence a chance to finish up saving or close a database connection gracefully.

Changed in version 20.0: Changed this method into an `abstractmethod()`.

abstract async get_bot_data()

Will be called by `telegram.ext.Application` upon creation with a persistence object. It should return the `bot_data` if stored, or an empty `dict`. In the latter case, the `dict` should produce values corresponding to one of the following:

- `dict`
- The type from `telegram.ext.ContextTypes.bot_data` if `telegram.ext.ContextTypes` are used.

Returns

The restored bot data.

Return type

`Dict[int, dict | telegram.ext.ContextTypes.bot_data]`

abstract async get_callback_data()

Will be called by `telegram.ext.Application` upon creation with a persistence object. If callback data was stored, it should be returned.

New in version 13.6.

Changed in version 20.0: Changed this method into an `abstractmethod()`.

Returns

`Tuple[List[Tuple[str, float, Dict[str, object]]], Dict[str, str]] | None`: The restored metadata or `None`, if no data was stored.

abstract async get_chat_data()

Will be called by `telegram.ext.Application` upon creation with a persistence object. It should return the `chat_data` if stored, or an empty `dict`. In the latter case, the dictionary should produce values corresponding to one of the following:

- `dict`
- The type from `telegram.ext.ContextTypes.chat_data` if `telegram.ext.ContextTypes` is used.

Changed in version 20.0: This method may now return a `dict` instead of a `collections.defaultdict`

Returns

The restored chat data.

Return type

`Dict[int, dict | telegram.ext.ContextTypes.chat_data]`

abstract async get_conversations(*name*)

Will be called by `telegram.ext.Application` when a `telegram.ext.ConversationHandler` is added if `telegram.ext.ConversationHandler.persistent` is `True`. It should return the conversations for the handler with `name` or an empty `dict`.

Parameters

`name` (`str`) – The handlers name.

Returns

The restored conversations for the handler.

Return type

`dict`

abstract async get_user_data()

Will be called by `telegram.ext.Application` upon creation with a persistence object. It should return the `user_data` if stored, or an empty `dict`. In the latter case, the dictionary should produce values corresponding to one of the following:

- `dict`
- The type from `telegram.ext.ContextTypes.user_data` if `telegram.ext.ContextTypes` is used.

Changed in version 20.0: This method may now return a `dict` instead of a `collections.defaultdict`

Returns

The restored user data.

Return type

`Dict[int, dict | telegram.ext.ContextTypes.user_data]`

abstract async refresh_bot_data(bot_data)

Will be called by the `telegram.ext.Application` before passing the `bot_data` to a callback. Can be used to update data stored in `bot_data` from an external source.

Warning: When using `concurrent_updates()`, this method may be called while a handler callback is still running. This might lead to race conditions.

New in version 13.6.

Changed in version 20.0: Changed this method into an `abstractmethod()`.

Parameters

`bot_data` (`dict` | `telegram.ext.ContextTypes.bot_data`) – The `bot_data`.

abstract async refresh_chat_data(chat_id, chat_data)

Will be called by the `telegram.ext.Application` before passing the `chat_data` to a callback. Can be used to update data stored in `chat_data` from an external source.

Warning: When using `concurrent_updates()`, this method may be called while a handler callback is still running. This might lead to race conditions.

New in version 13.6.

Changed in version 20.0: Changed this method into an `abstractmethod()`.

Parameters

- `chat_id` (`int`) – The chat ID this `chat_data` is associated with.
- `chat_data` (`dict` | `telegram.ext.ContextTypes.chat_data`) – The `chat_data` of a single chat.

abstract async refresh_user_data(*user_id*, *user_data*)

Will be called by the `telegram.ext.Application` before passing the *user_data* to a callback. Can be used to update data stored in *user_data* from an external source.

Warning: When using `concurrent_updates()`, this method may be called while a handler callback is still running. This might lead to race conditions.

New in version 13.6.

Changed in version 20.0: Changed this method into an `abstractmethod()`.

Parameters

- **user_id** (`int`) – The user ID this *user_data* is associated with.
- **user_data** (`dict` | `telegram.ext.ContextTypes.user_data`) – The *user_data* of a single user.

set_bot(*bot*)

Set the Bot to be used by this persistence instance.

Parameters

bot (`telegram.Bot`) – The bot.

Raises

TypeError – If `PersistenceInput.callback_data` is `True` and the *bot* is not an instance of `telegram.ext.ExtBot`.

abstract async update_bot_data(*data*)

Will be called by the `telegram.ext.Application` after a handler has handled an update.

Parameters

data (`dict` | `telegram.ext.ContextTypes.bot_data`) – The `telegram.ext.Application.bot_data`.

abstract async update_callback_data(*data*)

Will be called by the `telegram.ext.Application` after a handler has handled an update.

New in version 13.6.

Changed in version 20.0: Changed this method into an `abstractmethod()`.

Parameters

data (`Tuple[List[Tuple[str, float, Dict[str, Any]]], Dict[str, str]]` | `None`) – The relevant data to restore `telegram.ext.CallbackDataCache`.

abstract async update_chat_data(*chat_id*, *data*)

Will be called by the `telegram.ext.Application` after a handler has handled an update.

Parameters

- **chat_id** (`int`) – The chat the data might have been changed for.
- **data** (`dict` | `telegram.ext.ContextTypes.chat_data`) – The `telegram.ext.Application.chat_data` [*chat_id*].

abstract async update_conversation(*name*, *key*, *new_state*)

Will be called when a `telegram.ext.ConversationHandler` changes states. This allows the storage of the new state in the persistence.

Parameters

- **name** (`str`) – The handler's name.
- **key** (`tuple`) – The key the state is changed for.

- **new_state** (object) – The new state for the given key.

property `update_interval`

Time (in seconds) that the *Application* will wait between two consecutive runs of updating the persistence.

New in version 20.0.

Type

`float`

abstract `async update_user_data(user_id, data)`

Will be called by the *telegram.ext.Application* after a handler has handled an update.

Parameters

- **user_id** (int) – The user the data might have been changed for.
- **data** (dict | *telegram.ext.ContextTypes.user_data*) – The *telegram.ext.Application.user_data* [user_id].

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()`.
-

Use In

telegram.ext.ApplicationBuilder.persistence()

Available In

telegram.ext.Application.persistence

See also:

[Making Your Bot Persistent](#)

Changed in version 20.0: The parameters and attributes `store_*_data` were replaced by *store_data*.

Parameters

- **store_data** (*PersistenceInput*, optional) – Specifies which kinds of data will be saved by this persistence instance. By default, all available kinds of data will be saved.
- **user_data_json** (*str*, optional) – JSON string that will be used to reconstruct `user_data` on creating this persistence. Default is "".
- **chat_data_json** (*str*, optional) – JSON string that will be used to reconstruct `chat_data` on creating this persistence. Default is "".
- **bot_data_json** (*str*, optional) – JSON string that will be used to reconstruct `bot_data` on creating this persistence. Default is "".
- **conversations_json** (*str*, optional) – JSON string that will be used to reconstruct conversation on creating this persistence. Default is "".
- **callback_data_json** (*str*, optional) – JSON string that will be used to reconstruct `callback_data` on creating this persistence. Default is "".

New in version 13.6.

- **update_interval** (*int* | *float*, optional) – The *Application* will update the persistence in regular intervals. This parameter specifies the time (in seconds) to wait between two consecutive runs of updating the persistence. Defaults to 60 seconds.

New in version 20.0.

store_data

Specifies which kinds of data will be saved by this persistence instance.

Type

PersistenceInput

property bot_data

The `bot_data` as a dict.

Type

dict

property bot_data_json

The `bot_data` serialized as a JSON-string.

Type

str

property callback_data

The metadata on the stored callback data.

New in version 13.6.

Type

Tuple[*List*[*Tuple*[*str*, *float*, *Dict*[*str*, *object*]]], *Dict*[*str*, *str*]]

property callback_data_json

The metadata on the stored callback data as a JSON-string.

New in version 13.6.

Type

str

property chat_data

The `chat_data` as a dict.

Type

dict

property chat_data_json

The chat_data serialized as a JSON-string.

Type

`str`

property conversations

The conversations as a dict.

Type

`dict`

property conversations_json

The conversations serialized as a JSON-string.

Type

`str`

async drop_chat_data(chat_id)

Will delete the specified key from the `chat_data`.

New in version 20.0.

Parameters

`chat_id` (`int`) – The chat id to delete from the persistence.

async drop_user_data(user_id)

Will delete the specified key from the `user_data`.

New in version 20.0.

Parameters

`user_id` (`int`) – The user id to delete from the persistence.

async flush()

Does nothing.

New in version 20.0.

See also:

`telegram.ext.BasePersistence.flush()`

async get_bot_data()

Returns the bot_data created from the bot_data_json or an empty `dict`.

Returns

The restored bot data.

Return type

`dict`

async get_callback_data()

Returns the callback_data created from the callback_data_json or `None`.

New in version 13.6.

Returns

The restored metadata or `None`, if no data was stored.

Return type

`Tuple[List[Tuple[str, float, Dict[str, object]]], Dict[str, str]]`

async get_chat_data()

Returns the chat_data created from the chat_data_json or an empty `dict`.

Returns

The restored chat data.

Return type`dict`**async get_conversations**(*name*)

Returns the conversations created from the `conversations_json` or an empty `dict`.

Returns

The restored conversations data.

Return type`dict`**async get_user_data**()

Returns the `user_data` created from the `user_data_json` or an empty `dict`.

Returns

The restored user data.

Return type`dict`**async refresh_bot_data**(*bot_data*)

Does nothing.

New in version 13.6.

See also:

`telegram.ext.BasePersistence.refresh_bot_data()`

async refresh_chat_data(*chat_id*, *chat_data*)

Does nothing.

New in version 13.6.

See also:

`telegram.ext.BasePersistence.refresh_chat_data()`

async refresh_user_data(*user_id*, *user_data*)

Does nothing.

New in version 13.6.

See also:

`telegram.ext.BasePersistence.refresh_user_data()`

async update_bot_data(*data*)

Will update the `bot_data` (if changed).

Parameters

`data` (`dict`) – The `telegram.ext.Application.bot_data`.

async update_callback_data(*data*)

Will update the `callback_data` (if changed).

New in version 13.6.

Parameters

`data` (`Tuple`[`List`[`Tuple`[`str`, `float`, `Dict`[`str`, `object`]]], `Dict`[`str`, `str`]]) – The relevant data to restore `telegram.ext.CallbackDataCache`.

async update_chat_data(*chat_id*, *data*)

Will update the `chat_data` (if changed).

Parameters

- **`chat_id`** (`int`) – The chat the data might have been changed for.

- **data** (`dict`) – The `telegram.ext.Application.chat_data` [`chat_id`].

async update_conversation(*name*, *key*, *new_state*)

Will update the conversations for the given handler.

Parameters

- **name** (`str`) – The handler’s name.
- **key** (`tuple`) – The key the state is changed for.
- **new_state** (`tuple` | `object`) – The new state for the given key.

async update_user_data(*user_id*, *data*)

Will update the user_data (if changed).

Parameters

- **user_id** (`int`) – The user the data might have been changed for.
- **data** (`dict`) – The `telegram.ext.Application.user_data` [`user_id`].

property user_data

The user_data as a dict.

Type

`dict`

property user_data_json

The user_data serialized as a JSON-string.

Type

`str`

PersistenceInput

class telegram.ext.**PersistenceInput**(*bot_data=True*, *chat_data=True*, *user_data=True*,
 callback_data=True)

Bases: `NamedTuple`

Convenience wrapper to group boolean input for the `store_data` parameter for `BasePersistence`.

Parameters

- **bot_data** (`bool`, optional) – Whether the setting should be applied for bot_data. Defaults to `True`.
- **chat_data** (`bool`, optional) – Whether the setting should be applied for chat_data. Defaults to `True`.
- **user_data** (`bool`, optional) – Whether the setting should be applied for user_data. Defaults to `True`.
- **callback_data** (`bool`, optional) – Whether the setting should be applied for callback_data. Defaults to `True`.

bot_data

Whether the setting should be applied for bot_data.

Type

`bool`

chat_data

Whether the setting should be applied for chat_data.

Type

`bool`

user_data

Whether the setting should be applied for user_data.

Type

`bool`

callback_data

Whether the setting should be applied for callback_data.

Type

`bool`

Available In

- `telegram.ext.BasePersistence.store_data`
 - `telegram.ext.DictPersistence.store_data`
 - `telegram.ext.PicklePersistence.store_data`
-

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

Use In

`telegram.ext.ApplicationBuilder.persistence()`

Available In

`telegram.ext.Application.persistence`

See also:

[Making Your Bot Persistent](#)

Changed in version 20.0:

- The parameters and attributes `store_*_data` were replaced by `store_data`.
- The parameter and attribute `filename` were replaced by `filepath`.
- `filepath` now also accepts `pathlib.Path` as argument.

Parameters

- **`filepath`** (`str` | `pathlib.Path`) – The filepath for storing the pickle files. When `single_file` is `False` this will be used as a prefix.
- **`store_data`** (`PersistenceInput`, optional) – Specifies which kinds of data will be saved by this persistence instance. By default, all available kinds of data will be saved.
- **`single_file`** (`bool`, optional) – When `False` will store 5 separate files of `filename_user_data`, `filename_bot_data`, `filename_chat_data`, `filename_callback_data` and `filename_conversations`. Default is `True`.
- **`on_flush`** (`bool`, optional) – When `True` will only save to file when `flush()` is called and keep data in memory until that happens. When `False` will store data on any transaction *and* on call to `flush()`. Default is `False`.
- **`context_types`** (`telegram.ext.ContextTypes`, optional) – Pass an instance of `telegram.ext.ContextTypes` to customize the types used in the `context` interface. If not passed, the defaults documented in `telegram.ext.ContextTypes` will be used.
New in version 13.6.
- **`update_interval`** (`int` | `float`, optional) – The `Application` will update the persistence in regular intervals. This parameter specifies the time (in seconds) to wait between two consecutive runs of updating the persistence. Defaults to 60 seconds.
New in version 20.0.

filepath

The filepath for storing the pickle files. When `single_file` is `False` this will be used as a prefix.

Type

`str` | `pathlib.Path`

store_data

Specifies which kinds of data will be saved by this persistence instance.

Type

`PersistenceInput`

single_file

Optional. When `False` will store 5 separate files of `filename_user_data`, `filename_bot_data`, `filename_chat_data`, `filename_callback_data` and `filename_conversations`. Default is `True`.

Type

`bool`

on_flush

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.

New in version 13.6.

Type

`telegram.ext.ContextTypes`

async drop_chat_data(chat_id)

Will delete the specified key from the chat_data and depending on [on_flush](#) save the pickle file.

New in version 20.0.

Parameters

[chat_id](#) ([int](#)) – The chat id to delete from the persistence.

async drop_user_data(user_id)

Will delete the specified key from the user_data and depending on [on_flush](#) save the pickle file.

New in version 20.0.

Parameters

[user_id](#) ([int](#)) – The user id to delete from the persistence.

async flush()

Will save all data in memory to pickle file(s).

async get_bot_data()

Returns the bot_data from the pickle file if it exists or an empty object of type [dict](#) | [telegram.ext.ContextTypes.bot_data](#).

Returns

The restored bot data.

Return type

[dict](#) | [telegram.ext.ContextTypes.bot_data](#)

async get_callback_data()

Returns the callback data from the pickle file if it exists or [None](#).

New in version 13.6.

Returns

[Tuple](#)[[List](#)[[Tuple](#)[[str](#), [float](#), [Dict](#)[[str](#), [object](#)]]], [Dict](#)[[str](#), [str](#)]] | [None](#): The restored metadata or [None](#), if no data was stored.

async get_chat_data()

Returns the chat_data from the pickle file if it exists or an empty [dict](#).

Returns

The restored chat data.

Return type

[Dict](#)[[int](#), [dict](#)]

async get_conversations(name)

Returns the conversations from the pickle file if it exists or an empty dict.

Parameters

[name](#) ([str](#)) – The handlers name.

Returns

The restored conversations for the handler.

Return type

[dict](#)

async get_user_data()

Returns the user_data from the pickle file if it exists or an empty [dict](#).

Returns

The restored user data.

Return type

[Dict](#)[[int](#), [dict](#)]

async refresh_bot_data(*bot_data*)

Does nothing.

New in version 13.6.

See also:

`telegram.ext.BasePersistence.refresh_bot_data()`

async refresh_chat_data(*chat_id*, *chat_data*)

Does nothing.

New in version 13.6.

See also:

`telegram.ext.BasePersistence.refresh_chat_data()`

async refresh_user_data(*user_id*, *user_data*)

Does nothing.

New in version 13.6.

See also:

`telegram.ext.BasePersistence.refresh_user_data()`

async update_bot_data(*data*)

Will update the `bot_data` and depending on `on_flush` save the pickle file.

Parameters

data (dict | `telegram.ext.ContextTypes.bot_data`) – The `telegram.ext.Application.bot_data`.

async update_callback_data(*data*)

Will update the `callback_data` (if changed) and depending on `on_flush` save the pickle file.

New in version 13.6.

Parameters

data (Tuple[List[Tuple[str, float, Dict[str, object]]], Dict[str, str]]) – The relevant data to restore `telegram.ext.CallbackDataCache`.

async update_chat_data(*chat_id*, *data*)

Will update the `chat_data` and depending on `on_flush` save the pickle file.

Parameters

- **chat_id** (int) – The chat the data might have been changed for.
- **data** (dict) – The `telegram.ext.Application.chat_data` [`chat_id`].

async update_conversation(*name*, *key*, *new_state*)

Will update the conversations for the given handler and depending on `on_flush` save the pickle file.

Parameters

- **name** (str) – The handler's name.
- **key** (tuple) – The key the state is changed for.
- **new_state** (object) – The new state for the given key.

async update_user_data(*user_id*, *data*)

Will update the `user_data` and depending on `on_flush` save the pickle file.

Parameters

- **user_id** (int) – The user the data might have been changed for.
- **data** (dict) – The `telegram.ext.Application.user_data` [`user_id`].

10.2.15 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

Available In

[telegram.ext.ExtBot.callback_data_cache](#)

See also:

[Architecture Overview](#), [Arbitrary callback_data](#)

New 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.

New 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](#)

New in version 13.6.

Parameters

callback_data (*int*, optional) – The button data of which the callback data could not be found.

callback_data

Optional. The button data of which the callback data could not be found.

Type

int

__reduce__()

Defines how to serialize the exception for pickle. See [object.__reduce__\(\)](#) for more info.

Returns

tuple

10.2.16 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.

Use In

[telegram.ext.ApplicationBuilder.rate_limiter\(\)](#)

See also:

[Architecture Overview](#), [Avoiding Flood Limits](#)

New in version 20.0.

abstract async initialize()

Initialize resources used by this class. Must be implemented by a subclass.

abstract async process_request(*callback, args, kwargs, endpoint, data, rate_limit_args*)

Process a request. Must be implemented by a subclass.

This method must call *callback* and return the result of the call. *When* the callback is called is up to the implementation.

Important: This method must only return once the result of *callback* is known!

If a *RetryAfter* error is raised, this method may try to make a new request by calling the callback again.

Warning: This method *should not* handle any other exception raised by *callback*!

There are basically two different approaches how a rate limiter can be implemented:

1. React only if necessary. In this case, the *callback* is called without any precautions. If a *RetryAfter* error is raised, processing requests is halted for the *retry_after* and finally the *callback* is called again. This approach is often amendable for bots that don't have a large user base and/or don't send more messages than they get updates.
2. Throttle all outgoing requests. In this case the implementation makes sure that the requests are spread out over a longer time interval in order to stay below the rate limits. This approach is often amendable for bots that have a large user base and/or send more messages than they get updates.

An implementation can use the information provided by *data*, *endpoint* and *rate_limit_args* to handle each request differently.

Examples

- It is usually desirable to call *telegram.Bot.answer_inline_query()* as quickly as possible, while delaying *telegram.Bot.send_message()* is acceptable.
 - There are *different* rate limits for group chats and private chats.
 - When sending broadcast messages to a large number of users, these requests can typically be delayed for a longer time than messages that are direct replies to a user input.
-

Parameters

- *callback* (Callable[*...*, *coroutine*]) – The coroutine function that must be called to make the request.
- *args* (Tuple[*object*]) – The positional arguments for the *callback* function.
- *kwargs* (Dict[*str*, *object*]) – The keyword arguments for the *callback* function.
- *endpoint* (*str*) – The endpoint that the request is made for, e.g. "sendMessage".
- *data* (Dict[*str*, *object*]) – The parameters that were passed to the method of *ExtBot*. Any *api_kwargs* are included in this and any *defaults* are already applied.

Example

When calling:

```
await ext_bot.send_message(
    chat_id=1,
    text="Hello world!",
    api_kwargs={"custom": "arg"}
)
```

then *data* will be:

```
{"chat_id": 1, "text": "Hello world!", "custom": "arg"}
```

- ***rate_limit_args*** (*None* | *object*) – Custom arguments passed to the methods of *ExtBot*. Can e.g. be used to specify the priority of the request.

Returns

The result of the callback function.

Return type

bool | *Dict[str, object]* | *None*

abstract async shutdown()

Stop & clear resources used by this class. Must be implemented by a subclass.

AIORateLimiter

```
class telegram.ext.AIORateLimiter(overall_max_rate=30, overall_time_period=1,
                                  group_max_rate=20, group_time_period=60, max_retries=0)
```

Bases: *telegram.ext.BaseRateLimiter*

Implementation of *BaseRateLimiter* using the library *aiolimiter*.

Important: If you want to use this class, you must install PTB with the optional requirement *rate-limiter*, i.e.

```
pip install "python-telegram-bot[rate-limiter]"
```

The rate limiting is applied by combining two levels of throttling and *process_request()* roughly boils down to:

```
async with group_limiter(group_id):
    async with overall_limiter:
        await callback(*args, **kwargs)
```

Here, *group_id* is determined by checking if there is a *chat_id* parameter in the *data*. The *overall_limiter* is applied only if a *chat_id* argument is present at all.

Attention:

- Some bot methods accept a *chat_id* parameter in form of a *@username* for supergroups and channels. As we can't know which *@username* corresponds to which integer *chat_id*, these will be treated as different groups, which may lead to exceeding the rate limit.
- As channels can't be differentiated from supergroups by the *@username* or integer *chat_id*, this also applies the group related rate limits to channels.

- A `RetryAfter` exception will halt *all* requests for `retry_after` + 0.1 seconds. This may be stricter than necessary in some cases, e.g. the bot may hit a rate limit in one group but might still be allowed to send messages in another group.

Note: This class is to be understood as minimal effort reference implementation. If you would like to handle rate limiting in a more sophisticated, fine-tuned way, we welcome you to implement your own subclass of `BaseRateLimiter`. Feel free to check out the source code of this class for inspiration.

Use In

`telegram.ext.ApplicationBuilder.rate_limiter()`

See also:

[Avoiding Flood Limits](#)

New in version 20.0.

Parameters

- `overall_max_rate` (float) – The maximum number of requests allowed for the entire bot per `overall_time_period`. When set to 0, no rate limiting will be applied. Defaults to 30.
- `overall_time_period` (float) – The time period (in seconds) during which the `overall_max_rate` is enforced. When set to 0, no rate limiting will be applied. Defaults to 1.
- `group_max_rate` (float) – The maximum number of requests allowed for requests related to groups and channels per `group_time_period`. When set to 0, no rate limiting will be applied. Defaults to 20.
- `group_time_period` (float) – The time period (in seconds) during which the `group_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.

10.3 Auxiliary modules

10.3.1 telegram.constants Module

This module contains several constants that are relevant for working with the Bot API.

Unless noted otherwise, all constants in this module were extracted from the [Telegram Bots FAQ](#) and [Telegram Bots API](#).

Most of the following constants are related to specific classes or topics and are grouped into enums. If they are related to a specific class, then they are also available as attributes of those classes.

Changed in version 20.0:

- Most of the constants in this module are grouped into enums.

class telegram.constants.**AccentColor**(value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None)

Bases: [Enum](#)

This enum contains the available accent colors for `telegram.Chat.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.

New 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 = '7.0'`

Telegram Bot API version supported by this version of *python-telegram-bot*. Also available as `telegram.__bot_api_version__`.

New in version 13.4.

Type

`str`

`telegram.constants.BOT_API_VERSION_INFO = (7, 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__`.

New in version 20.0.

class `telegram.constants.BotCommandLimit`(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

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.

New 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`

__abs__()

`abs(self)`

__add__(*value, /*)

Return self+value.

__and__(*value, /*)

Return self&value.

`__bool__()`
True if self else False

`__ceil__()`
Ceiling of an Integral returns itself.

`__divmod__(value, /)`
Return divmod(self, value).

`__eq__(value, /)`
Return self==value.

`__float__()`
float(self)

`__floor__()`
Flooring an Integral returns itself.

`__floordiv__(value, /)`
Return self//value.

`__format__(format_spec, /)`
Convert to a string according to format_spec.

`__ge__(value, /)`
Return self>=value.

`__getattr__(name, /)`
Return getattr(self, name).

`__gt__(value, /)`
Return self>value.

`__hash__()`
Return hash(self).

`__index__()`
Return self converted to an integer, if self is suitable for use as an index into a list.

`__int__()`
int(self)

`__invert__()`
~self

`__le__(value, /)`
Return self<=value.

`__lshift__(value, /)`
Return self<<value.

`__lt__(value, /)`
Return self<value.

`__mod__(value, /)`
Return self%value.

`__mul__(value, /)`
Return self*value.

`__ne__(value, /)`
Return self!=value.

`__neg__()`
-self

`__new__(value)`

`__or__(value, /)`
Return self|value.

`__pos__()`
+self

`__pow__(value, mod=None, /)`
Return pow(self, value, mod).

`__radd__(value, /)`
Return value+self.

`__rand__(value, /)`
Return value&self.

`__rdivmod__(value, /)`
Return divmod(value, self).

`__repr__()`
Return repr(self).

`__rfloordiv__(value, /)`
Return value//self.

`__rlshift__(value, /)`
Return value<<self.

`__rmod__(value, /)`
Return value%self.

`__rmul__(value, /)`
Return value*self.

`__ror__(value, /)`
Return value|self.

`__round__()`
Rounding an Integral returns itself.
Rounding with an ndigits argument also returns an integer.

`__rpow__(value, mod=None, /)`
Return pow(value, self, mod).

`__rrshift__(value, /)`
Return value>>self.

`__rshift__(value, /)`
Return self>>value.

`__rsub__(value, /)`
Return value-self.

`__rtruediv__(value, /)`
Return value/self.

`__rxor__(value, /)`
Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If `byteorder` is `'big'`, the most significant byte is at the beginning of the byte array. If `byteorder` is `'little'`, the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use `'big'`.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If `byteorder` is `'big'`, the most significant byte is at the beginning of the byte array. If `byteorder` is `'little'`, the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use `'big'`.

signed

Determines whether two's complement is used to represent the integer. If `signed` is `False` and a negative integer is given, an `OverflowError` is raised.

```
class telegram.constants.BotCommandScopeType(value, names=None, *values, module=None,  
                                             qualname=None, type=None, start=1,  
                                             boundary=None)
```

Bases: `str`, `enum.Enum`

This enum contains the available types of `telegram.BotCommandScope`. The enum members of this enumeration are instances of `str` and can be treated as such.

New in version 20.0.

```
ALL_CHAT_ADMINISTRATORS = 'all_chat_administrators'
```

The type of `telegram.BotCommandScopeAllChatAdministrators`.

Type

`str`

```
ALL_GROUP_CHATS = 'all_group_chats'
```

The type of `telegram.BotCommandScopeAllGroupChats`.

Type

`str`

```
ALL_PRIVATE_CHATS = 'all_private_chats'
```

The type of `telegram.BotCommandScopeAllPrivateChats`.

Type

`str`

CHAT = 'chat'

The type of *telegram.BotCommandScopeChat*.

Type

str

CHAT_ADMINISTRATORS = 'chat_administrators'

The type of *telegram.BotCommandScopeChatAdministrators*.

Type

str

CHAT_MEMBER = 'chat_member'

The type of *telegram.BotCommandScopeChatMember*.

Type

str

DEFAULT = 'default'

The type of *telegram.BotCommandScopeDefault*.

Type

str

__add__(value, /)

Return self+value.

__contains__(key, /)

Return bool(key in self).

__eq__(value, /)

Return self==value.

__format__(format_spec)

Return a formatted version of the string as described by format_spec.

__ge__(value, /)

Return self>=value.

__getattr__(name, /)

Return getattr(self, name).

__getitem__(key, /)

Return self[key].

__gt__(value, /)

Return self>value.

__hash__()

Return hash(self).

__iter__()

Implement iter(self).

__le__(value, /)

Return self<=value.

__len__()

Return len(self).

__lt__(value, /)

Return self<value.

__mod__(value, /)

Return self%value.

__mul__(value, /)

Return self*value.

__ne__(value, /)

Return self!=value.

__new__(value)

__repr__()

Return repr(self).

__rmod__(value, /)

Return value%self.

__rmul__(value, /)

Return value*self.

__sizeof__()

Return the size of the string in memory, in bytes.

__str__()

Return str(self).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(width, fillchar=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(sub[, start[, end]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

encode(encoding='utf-8', errors='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlchar-refreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith(suffix[, start[, end]]) → bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

expandtabs(tabsize=8)

Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → str

Return a formatted version of *S*, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → str

Return a formatted version of *S*, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call `keyword.iskeyword(s)` to test whether string *s* is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in `repr()` or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(prefix, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(*suffix*, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return string[:-len(suffix)]. Otherwise, return a copy of the original string.

replace(*old*, *new*, *count=-1*, /)

Return a copy with all occurrences of substring *old* replaced by *new*.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises *ValueError* when the substring is not found.

rjust(*width*, *fillchar=' '*, /)

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(*sep=None*, *maxsplit=-1*)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to *None* (the default value), will split on any whitespace character (including *newline* and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars=None*, /)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not *None*, remove characters in *chars* instead.

split(*sep=None*, *maxsplit=-1*)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to *None* (the default value), will split on any whitespace character (including *newline* and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless keepends is given and true.

```
startswith(prefix[, start[, end]]) → bool
```

Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

strip(*chars=None, /*)

Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table*, /)

Replace each character in the string using the given translation table.

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to `None` are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(*width*, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

```
class telegram.constants.BotDescriptionLimit(value, names=None, *values, module=None,
                                             qualname=None, type=None, start=1,
                                             boundary=None)
```

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.

New 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`

__abs__()

`abs(self)`

__add__(*value*, /)

Return self+value.

__and__(*value*, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

__divmod__(*value*, /)

Return divmod(self, value).

__eq__(*value*, /)

Return self==value.

__float__()

`float(self)`

__floor__()

Flooring an Integral returns itself.

__floordiv__(*value*, /)

Return self//value.

__format__(*format_spec*, /)

Convert to a string according to `format_spec`.

__ge__(*value*, /)

Return self>=value.

__getattr__(*name*, /)

Return `getattr(self, name)`.

__gt__(*value*, /)

Return self>value.

__hash__()

Return `hash(self)`.

__index__()

Return self converted to an integer, if self is suitable for use as an index into a list.

__int__()

`int(self)`

__invert__()

`~self`

__le__(*value*, /)

Return self<=value.

__lshift__(value, /)
Return self<<value.

__lt__(value, /)
Return self<value.

__mod__(value, /)
Return self%value.

__mul__(value, /)
Return self*value.

__ne__(value, /)
Return self!=value.

__neg__()
-self

__new__(value)

__or__(value, /)
Return self|value.

__pos__()
+self

__pow__(value, mod=None, /)
Return pow(self, value, mod).

__radd__(value, /)
Return value+self.

__rand__(value, /)
Return value&self.

__rdivmod__(value, /)
Return divmod(value, self).

__repr__()
Return repr(self).

__rfloordiv__(value, /)
Return value//self.

__rlshift__(value, /)
Return value<<self.

__rmod__(value, /)
Return value%self.

__rmul__(value, /)
Return value*self.

__ror__(value, /)
Return value|self.

__round__()
Rounding an Integral returns itself.
Rounding with an ndigits argument also returns an integer.

__rpow__(value, mod=None, /)
Return pow(value, self, mod).

__rrshift__(value, /)

Return value>>self.

__rshift__(value, /)

Return self>>value.

__rsub__(value, /)

Return value-self.

__rtruediv__(value, /)

Return value/self.

__rxor__(value, /)

Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```


conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an `OverflowError` is raised.

class telegram.constants.**BotNameLimit**(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

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.

New in version 20.3.

MAX_NAME_LENGTH = 64

Maximum length for the parameter *name* of `telegram.Bot.set_my_name()`

Type`int`**`__abs__()`**`abs(self)`**`__add__(value, /)`**`Return self+value.`**`__and__(value, /)`**`Return self&value.`**`__bool__()`**`True if self else False`**`__ceil__()`**`Ceiling of an Integral returns itself.`**`__divmod__(value, /)`**`Return divmod(self, value).`**`__eq__(value, /)`**`Return self==value.`**`__float__()`**`float(self)`**`__floor__()`**`Flooring an Integral returns itself.`**`__floordiv__(value, /)`**`Return self//value.`**`__format__(format_spec, /)`**`Convert to a string according to format_spec.`**`__ge__(value, /)`**`Return self>=value.`**`__getattr__(name, /)`**`Return getattr(self, name).`**`__gt__(value, /)`**`Return self>value.`**`__hash__()`**`Return hash(self).`**`__index__()`**`Return self converted to an integer, if self is suitable for use as an index into a list.`**`__int__()`**`int(self)`**`__invert__()`**`~self`**`__le__(value, /)`**`Return self<=value.`**`__lshift__(value, /)`**`Return self<<value.`

`__lt__(value, /)`
Return self<value.

`__mod__(value, /)`
Return self%value.

`__mul__(value, /)`
Return self*value.

`__ne__(value, /)`
Return self!=value.

`__neg__()`
-self

`__new__(value)`

`__or__(value, /)`
Return self|value.

`__pos__()`
+self

`__pow__(value, mod=None, /)`
Return pow(self, value, mod).

`__radd__(value, /)`
Return value+self.

`__rand__(value, /)`
Return value&self.

`__rdivmod__(value, /)`
Return divmod(value, self).

`__repr__()`
Return repr(self).

`__rfloordiv__(value, /)`
Return value//self.

`__rlshift__(value, /)`
Return value<<self.

`__rmod__(value, /)`
Return value%self.

`__rmul__(value, /)`
Return value*self.

`__ror__(value, /)`
Return value|self.

`__round__()`
Rounding an Integral returns itself.
Rounding with an ndigits argument also returns an integer.

`__rpow__(value, mod=None, /)`
Return pow(value, self, mod).

`__rrshift__(value, /)`
Return value>>self.

__rshift__(value, /)

Return self>>value.

__rsub__(value, /)

Return value-self.

__rtruediv__(value, /)

Return value/self.

__rxor__(value, /)

Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an `OverflowError` is raised.

class telegram.constants.**BulkRequestLimit**(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

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.

New 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`

__abs__()

`abs(self)`

__add__(value, /)

Return self+value.

__and__(value, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

__divmod__(value, /)

Return divmod(self, value).

__eq__(value, /)

Return self==value.

__float__()

`float(self)`

__floor__()

Flooring an Integral returns itself.

__floordiv__(value, /)

Return self//value.

__format__(format_spec, /)

Convert to a string according to format_spec.

__ge__(value, /)

Return self>=value.

__getattr__(name, /)

Return getattr(self, name).

__gt__(value, /)

Return self>value.

__hash__()

Return hash(self).

__index__()

Return self converted to an integer, if self is suitable for use as an index into a list.

__int__()

`int(self)`

`__invert__()`
~self

`__le__(value, /)`
Return self<=value.

`__lshift__(value, /)`
Return self<<value.

`__lt__(value, /)`
Return self<value.

`__mod__(value, /)`
Return self%value.

`__mul__(value, /)`
Return self*value.

`__ne__(value, /)`
Return self!=value.

`__neg__()`
-self

`__new__(value)`

`__or__(value, /)`
Return self|value.

`__pos__()`
+self

`__pow__(value, mod=None, /)`
Return pow(self, value, mod).

`__radd__(value, /)`
Return value+self.

`__rand__(value, /)`
Return value&self.

`__rdivmod__(value, /)`
Return divmod(value, self).

`__repr__()`
Return repr(self).

`__rfloordiv__(value, /)`
Return value//self.

`__rlshift__(value, /)`
Return value<<self.

`__rmod__(value, /)`
Return value%self.

`__rmul__(value, /)`
Return value*self.

`__ror__(value, /)`
Return value|self.

__round__()

Rounding an Integral returns itself.

Rounding with an ndigits argument also returns an integer.

__rpow__(value, mod=None, /)

Return pow(value, self, mod).

__rrshift__(value, /)

Return value>>self.

__rshift__(value, /)

Return self>>value.

__rsub__(value, /)

Return value-self.

__rtruediv__(value, /)

Return value/self.

__rxor__(value, /)

Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```


bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with float.is_integer.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An OverflowError is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

```
class telegram.constants.CallbackQueryLimit(value, names=None, *values, module=None,  
                                           qualname=None, type=None, start=1,  
                                           boundary=None)
```

Bases: `enum.IntEnum`

This enum contains limitations for `telegram.CallbackQuery/ telegram.Bot.answer_callback_query()`. The enum members of this enumeration are instances of `int` and can be treated as such.

New in version 20.0.

ANSWER_CALLBACK_QUERY_TEXT_LENGTH = 200

Maximum number of characters in a `str` passed as the `text` parameter of `telegram.Bot.answer_callback_query()`.

Type
`int`

__abs__()

abs(self)

__add__(value, /)

Return self+value.

__and__(value, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

__divmod__(value, /)

Return divmod(self, value).

__eq__(value, /)

Return self==value.

__float__()

float(self)

__floor__()

Flooring an Integral returns itself.

__floordiv__(value, /)

Return self//value.

__format__(format_spec, /)

Convert to a string according to format_spec.

__ge__(value, /)

Return self>=value.

__getattr__(name, /)

Return getattr(self, name).

__gt__(value, /)

Return self>value.

__hash__()

Return hash(self).

__index__()

Return self converted to an integer, if self is suitable for use as an index into a list.

`__int__()`
int(self)

`__invert__()`
~self

`__le__(value, /)`
Return self<=value.

`__lshift__(value, /)`
Return self<<value.

`__lt__(value, /)`
Return self<value.

`__mod__(value, /)`
Return self%value.

`__mul__(value, /)`
Return self*value.

`__ne__(value, /)`
Return self!=value.

`__neg__()`
-self

`__new__(value)`

`__or__(value, /)`
Return self|value.

`__pos__()`
+self

`__pow__(value, mod=None, /)`
Return pow(self, value, mod).

`__radd__(value, /)`
Return value+self.

`__rand__(value, /)`
Return value&self.

`__rdivmod__(value, /)`
Return divmod(value, self).

`__repr__()`
Return repr(self).

`__rfloordiv__(value, /)`
Return value//self.

`__rlshift__(value, /)`
Return value<<self.

`__rmod__(value, /)`
Return value%self.

`__rmul__(value, /)`
Return value*self.

__ror__(value, /)

Return value|self.

__round__()

Rounding an Integral returns itself.

Rounding with an ndigits argument also returns an integer.

__rpow__(value, mod=None, /)

Return pow(value, self, mod).

__rrshift__(value, /)

Return value>>self.

__rshift__(value, /)

Return self>>value.

__rsub__(value, /)

Return value-self.

__rtruediv__(value, /)

Return value/self.

__rxor__(value, /)

Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

```
class telegram.constants.ChatAction(value, names=None, *values, module=None, qualname=None,  
                                     type=None, start=1, boundary=None)
```

Bases: `str`, `enum.Enum`

This enum contains the available chat actions for `telegram.Bot.send_chat_action()`. The enum members of this enumeration are instances of `str` and can be treated as such.

New in version 20.0.

```
CHOOSE_STICKER = 'choose_sticker'
```

Chat action indicating that the bot is selecting a sticker.

Type

`str`

```
FIND_LOCATION = 'find_location'
```

Chat action indicating that the bot is selecting a location.

Type

`str`

```
RECORD_VIDEO = 'record_video'
```

Chat action indicating that the bot is recording a video.

Type

`str`

```
RECORD_VIDEO_NOTE = 'record_video_note'
```

Chat action indicating that the bot is recording a video note.

Type

`str`

```
RECORD_VOICE = 'record_voice'
```

Chat action indicating that the bot is recording a voice message.

Type

`str`

```
TYPING = 'typing'
```

A chat indicating the bot is typing.

Type

`str`

```
UPLOAD_DOCUMENT = 'upload_document'
```

Chat action indicating that the bot is uploading a document.

Type

`str`

```
UPLOAD_PHOTO = 'upload_photo'
```

Chat action indicating that the bot is uploading a photo.

Type

`str`

```
UPLOAD_VIDEO = 'upload_video'
```

Chat action indicating that the bot is uploading a video.

Type

`str`

UPLOAD_VIDEO_NOTE = 'upload_video_note'

Chat action indicating that the bot is uploading a video note.

Type

`str`

UPLOAD_VOICE = 'upload_voice'

Chat action indicating that the bot is uploading a voice message.

Type

`str`

__add__(*value*, /)

Return self+value.

__contains__(*key*, /)

Return bool(key in self).

__eq__(*value*, /)

Return self==value.

__format__(*format_spec*)

Return a formatted version of the string as described by *format_spec*.

__ge__(*value*, /)

Return self>=value.

__getattr__(*name*, /)

Return getattr(self, name).

__getitem__(*key*, /)

Return self[key].

__gt__(*value*, /)

Return self>value.

__hash__()

Return hash(self).

__iter__()

Implement iter(self).

__le__(*value*, /)

Return self<=value.

__len__()

Return len(self).

__lt__(*value*, /)

Return self<value.

__mod__(*value*, /)

Return self%value.

__mul__(*value*, /)

Return self*value.

__ne__(*value*, /)

Return self!=value.

__new__(*value*)

__repr__()

Return repr(self).

__rmod__(*value*, /)

Return *value*%*self*.

__rmul__(*value*, /)

Return *value***self*.

__sizeof__()

Return the size of the string in memory, in bytes.

__str__()

Return *str*(*self*).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(*width*, *fillchar*=' ', /)

Return a centered string of length *width*.

Padding is done using the specified fill character (default is a space).

count(*sub*[, *start*[, *end*]]) → *int*

Return the number of non-overlapping occurrences of substring *sub* in string *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

encode(*encoding*='utf-8', *errors*='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a `UnicodeEncodeError`. Other possible values are 'ignore', 'replace' and 'xmlchar-refreplace' as well as any other name registered with `codecs.register_error` that can handle `UnicodeEncodeErrors`.

endswith(*suffix*[, *start*[, *end*]]) → *bool*

Return `True` if *S* ends with the specified *suffix*, `False` otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *suffix* can also be a tuple of strings to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → *int*

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → *str*

Return a formatted version of *S*, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → *str*

Return a formatted version of *S*, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

isalnum()

Return `True` if the string is an alpha-numeric string, `False` otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return `True` if the string is an alphabetic string, `False` otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return `True` if all characters in the string are ASCII, `False` otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return `True` if the string is a decimal string, `False` otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return `True` if the string is a digit string, `False` otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return `True` if the string is a valid Python identifier, `False` otherwise.

Call `keyword.iskeyword(s)` to test whether string *s* is a reserved identifier, such as “`def`” or “`class`”.

islower()

Return `True` if the string is a lowercase string, `False` otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return `True` if the string is a numeric string, `False` otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return `True` if the string is printable, `False` otherwise.

A string is printable if all of its characters are considered printable in `repr()` or if it is empty.

isspace()

Return `True` if the string is a whitespace string, `False` otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(prefix, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(suffix, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(old, new, count=-1, /)

Return a copy with all occurrences of substring old replaced by new.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(*width*, *fillchar*=' ', /)

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars*=None, /)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

split(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if *S* starts with the specified prefix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *prefix* can also be a tuple of strings to try.

strip(*chars=None*, /)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table*, /)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(*width*, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.ChatBoostSources(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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.

New 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`

__add__(*value*, /)

Return self+value.

__contains__(*key*, /)

Return bool(key in self).

__eq__(*value*, /)

Return self==value.

__format__(*format_spec*)

Return a formatted version of the string as described by *format_spec*.

__ge__(*value*, /)

Return self>=value.

__getattr__(*name*, /)

Return getattr(self, name).

__getitem__(*key*, /)

Return self[key].

__gt__(*value*, /)

Return self>value.

__hash__()

Return hash(self).

__iter__()

Implement iter(self).

__le__(*value*, /)

Return self<=value.

__len__()

Return len(self).

__lt__(*value*, /)

Return self<value.

__mod__(*value*, /)

Return self%value.

__mul__(*value*, /)

Return self*value.

__ne__(*value*, /)

Return self!=value.

__new__(*value*)

__repr__()

Return repr(self).

__rmod__(*value*, /)

Return value%self.

__rmul__(value, /)

Return value*self.

__sizeof__()

Return the size of the string in memory, in bytes.

__str__()

Return str(self).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(width, fillchar=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(sub[, start[, end]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

encode(encoding='utf-8', errors='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlchar-refreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith(suffix[, start[, end]]) → bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

expandtabs(tabsize=8)

Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

find(sub[, start[, end]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

format(*args, **kwargs) → str

Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

format_map(mapping) → str

Return a formatted version of S, using substitutions from mapping. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

isalnum()

Return `True` if the string is an alpha-numeric string, `False` otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return `True` if the string is an alphabetic string, `False` otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return `True` if all characters in the string are ASCII, `False` otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return `True` if the string is a decimal string, `False` otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return `True` if the string is a digit string, `False` otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return `True` if the string is a valid Python identifier, `False` otherwise.

Call `keyword.iskeyword(s)` to test whether string *s* is a reserved identifier, such as “`def`” or “`class`”.

islower()

Return `True` if the string is a lowercase string, `False` otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return `True` if the string is a numeric string, `False` otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return `True` if the string is printable, `False` otherwise.

A string is printable if all of its characters are considered printable in `repr()` or if it is empty.

isspace()

Return `True` if the string is a whitespace string, `False` otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs'])` -> `'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(prefix, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(suffix, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(old, new, count=-1, /)

Return a copy with all occurrences of substring old replaced by new.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind(sub[, start[, end]]) → int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

rindex(sub[, start[, end]]) → int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

rjust(width, fillchar=' ', /)

Return a right-justified string of length width.

Padding is done using the specified fill character (default is a space).

rpartition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(chars=None, /)

Return a copy of the string with trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

split(sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, str.split() is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if *S* starts with the specified prefix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *prefix* can also be a tuple of strings to try.

strip(*chars=None, /*)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table, /*)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(*width, /*)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.ChatID(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `enum.IntEnum`

This enum contains some special chat IDs. The enum members of this enumeration are instances of `int` and can be treated as such.

New in version 20.0.

ANONYMOUS_ADMIN = 1087968824

User ID in groups for messages sent by anonymous admins. 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`

__abs__()

abs(self)

__add__(value, /)

Return self+value.

__and__(value, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

__divmod__(value, /)

Return divmod(self, value).

__eq__(value, /)

Return self==value.

__float__()

float(self)

__floor__()

Flooring an Integral returns itself.

__floordiv__(value, /)

Return self//value.

__format__(format_spec, /)

Convert to a string according to format_spec.

__ge__(value, /)
Return self>=value.

__getattr__(name, /)
Return getattr(self, name).

__gt__(value, /)
Return self>value.

__hash__()
Return hash(self).

__index__()
Return self converted to an integer, if self is suitable for use as an index into a list.

__int__()
int(self)

__invert__()
~self

__le__(value, /)
Return self<=value.

__lshift__(value, /)
Return self<<value.

__lt__(value, /)
Return self<value.

__mod__(value, /)
Return self%value.

__mul__(value, /)
Return self*value.

__ne__(value, /)
Return self!=value.

__neg__()
-self

__new__(value)

__or__(value, /)
Return self|value.

__pos__()
+self

__pow__(value, mod=None, /)
Return pow(self, value, mod).

__radd__(value, /)
Return value+self.

__rand__(value, /)
Return value&self.

__rdivmod__(value, /)
Return divmod(value, self).

__repr__()
Return repr(self).

__rfloordiv__(value, /)
Return value//self.

__rlshift__(value, /)
Return value<<self.

__rmod__(value, /)
Return value%self.

__rmul__(value, /)
Return value*self.

__ror__(value, /)
Return value|self.

__round__()
Rounding an Integral returns itself.
Rounding with an ndigits argument also returns an integer.

__rpow__(value, mod=None, /)
Return pow(value, self, mod).

__rrshift__(value, /)
Return value>>self.

__rshift__(value, /)
Return self>>value.

__rsub__(value, /)
Return value-self.

__rtruediv__(value, /)
Return value/self.

__rxor__(value, /)
Return value^self.

__sizeof__()
Returns size in memory, in bytes.

__str__()
Return repr(self).

__sub__(value, /)
Return self-value.

__truediv__(value, /)
Return self/value.

__trunc__()
Truncating an Integral returns itself.

__xor__(value, /)
Return self^value.

as_integer_ratio()
Return a pair of integers, whose ratio is equal to the original int.
The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1*, *byteorder='big'*, *, *signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If *byteorder* is 'big', the most significant byte is at the beginning of the byte array. If *byteorder* is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If *signed* is `False` and a negative integer is given, an `OverflowError` is raised.

class telegram.constants.**ChatInviteLinkLimit**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `enum.IntEnum`

This enum contains limitations for `telegram.ChatInviteLink/telegram.Bot.create_chat_invite_link()/telegram.Bot.edit_chat_invite_link()`. The enum members of this enumeration are instances of `int` and can be treated as such.

New in version 20.0.

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`

__abs__()

`abs(self)`

__add__(*value*, /)

Return self+value.

__and__(*value*, /)

Return self&value.

__bool__()

True if self else False

__ceil__()
Ceiling of an Integral returns itself.

__divmod__(value, /)
Return divmod(self, value).

__eq__(value, /)
Return self==value.

__float__()
float(self)

__floor__()
Flooring an Integral returns itself.

__floordiv__(value, /)
Return self//value.

__format__(format_spec, /)
Convert to a string according to format_spec.

__ge__(value, /)
Return self>=value.

__getattr__(name, /)
Return getattr(self, name).

__gt__(value, /)
Return self>value.

__hash__()
Return hash(self).

__index__()
Return self converted to an integer, if self is suitable for use as an index into a list.

__int__()
int(self)

__invert__()
~self

__le__(value, /)
Return self<=value.

__lshift__(value, /)
Return self<<value.

__lt__(value, /)
Return self<value.

__mod__(value, /)
Return self%value.

__mul__(value, /)
Return self*value.

__ne__(value, /)
Return self!=value.

__neg__()
-self

`__new__(value)`
`__or__(value, /)`
Return self|value.
`__pos__()`
+self
`__pow__(value, mod=None, /)`
Return pow(self, value, mod).
`__radd__(value, /)`
Return value+self.
`__rand__(value, /)`
Return value&self.
`__rdivmod__(value, /)`
Return divmod(value, self).
`__repr__()`
Return repr(self).
`__rfloordiv__(value, /)`
Return value//self.
`__rlshift__(value, /)`
Return value<<self.
`__rmod__(value, /)`
Return value%self.
`__rmul__(value, /)`
Return value*self.
`__ror__(value, /)`
Return value|self.
`__round__()`
Rounding an Integral returns itself.
Rounding with an ndigits argument also returns an integer.
`__rpow__(value, mod=None, /)`
Return pow(value, self, mod).
`__rrshift__(value, /)`
Return value>>self.
`__rshift__(value, /)`
Return self>>value.
`__rsub__(value, /)`
Return value-self.
`__rtruediv__(value, /)`
Return value/self.
`__rxor__(value, /)`
Return value^self.
`__sizeof__()`
Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of

the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use `'big'`.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If `byteorder` is `'big'`, the most significant byte is at the beginning of the byte array. If `byteorder` is `'little'`, the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use `'big'`.

signed

Determines whether two's complement is used to represent the integer. If `signed` is `False` and a negative integer is given, an `OverflowError` is raised.

class telegram.constants.ChatLimit(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

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.

New 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`

__abs__()

`abs(self)`

__add__(*value*, /)

Return self+value.

__and__(*value*, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

__divmod__(*value*, /)

Return divmod(self, value).

__eq__(*value*, /)

Return self==value.

__float__()

`float(self)`

__floor__()

Flooring an Integral returns itself.

__floordiv__(*value*, /)

Return self//value.

__format__(*format_spec*, /)

Convert to a string according to `format_spec`.

__ge__(*value*, /)

Return self>=value.

__getattr__(*name*, /)

Return `getattr(self, name)`.

__gt__(*value*, /)

Return self>value.

__hash__()

Return `hash(self)`.

__index__()

Return self converted to an integer, if self is suitable for use as an index into a list.

__int__()

`int(self)`

__invert__()

`~self`

__le__(*value*, /)

Return self<=value.

`__lshift__(value, /)`
Return `self<<value`.

`__lt__(value, /)`
Return `self<value`.

`__mod__(value, /)`
Return `self%value`.

`__mul__(value, /)`
Return `self*value`.

`__ne__(value, /)`
Return `self!=value`.

`__neg__()`
`-self`

`__new__(value)`

`__or__(value, /)`
Return `self|value`.

`__pos__()`
`+self`

`__pow__(value, mod=None, /)`
Return `pow(self, value, mod)`.

`__radd__(value, /)`
Return `value+self`.

`__rand__(value, /)`
Return `value&self`.

`__rdivmod__(value, /)`
Return `divmod(value, self)`.

`__repr__()`
Return `repr(self)`.

`__rfloordiv__(value, /)`
Return `value//self`.

`__rlshift__(value, /)`
Return `value<<self`.

`__rmod__(value, /)`
Return `value%self`.

`__rmul__(value, /)`
Return `value*self`.

`__ror__(value, /)`
Return `value|self`.

`__round__()`
Rounding an Integral returns itself.
Rounding with an `ndigits` argument also returns an integer.

`__rpow__(value, mod=None, /)`
Return `pow(value, self, mod)`.

__rrshift__(value, /)

Return value>>self.

__rshift__(value, /)

Return self>>value.

__rsub__(value, /)

Return value-self.

__rtruediv__(value, /)

Return value/self.

__rxor__(value, /)

Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an `OverflowError` is raised.

```
class telegram.constants.ChatMemberStatus(value, names=None, *values, module=None,  
                                           qualname=None, type=None, start=1, boundary=None)
```

Bases: `str`, `enum.Enum`

This enum contains the available states for `telegram.ChatMember`. The enum members of this enumeration are instances of `str` and can be treated as such.

New in version 20.0.

```
ADMINISTRATOR = 'administrator'
```

A `telegram.ChatMember` who is administrator of the chat.

Type

str

BANNED = 'kicked'

A *telegram.ChatMember* who was banned in the chat.

Type

str

LEFT = 'left'

A *telegram.ChatMember* who has left the chat.

Type

str

MEMBER = 'member'

A *telegram.ChatMember* who is a member of the chat.

Type

str

OWNER = 'creator'

A *telegram.ChatMember* who is the owner of the chat.

Type

str

RESTRICTED = 'restricted'

A *telegram.ChatMember* who was restricted in this chat.

Type

str

__add__(value, /)

Return self+value.

__contains__(key, /)

Return bool(key in self).

__eq__(value, /)

Return self==value.

__format__(format_spec)

Return a formatted version of the string as described by format_spec.

__ge__(value, /)

Return self>=value.

__getattr__(name, /)

Return getattr(self, name).

__getitem__(key, /)

Return self[key].

__gt__(value, /)

Return self>value.

__hash__()

Return hash(self).

__iter__()

Implement iter(self).

__le__(value, /)

Return self<=value.

__len__()

Return len(self).

__lt__(value, /)

Return self<value.

__mod__(value, /)

Return self%value.

__mul__(value, /)

Return self*value.

__ne__(value, /)

Return self!=value.

__new__(value)

__repr__()

Return repr(self).

__rmod__(value, /)

Return value%self.

__rmul__(value, /)

Return value*self.

__sizeof__()

Return the size of the string in memory, in bytes.

__str__()

Return str(self).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(width, fillchar=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(sub[, start[, end]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

encode(encoding='utf-8', errors='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith(*suffix*[, *start*[, *end*]]) → bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. *suffix* can also be a tuple of strings to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int

Return the lowest index in S where substring *sub* is found, such that *sub* is contained within S[start:end]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → str

Return a formatted version of S, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → str

Return a formatted version of S, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in S where substring *sub* is found, such that *sub* is contained within S[start:end]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string *s* is a reserved identifier, such as "def" or "class".

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans()

Return a translation table usable for str.translate().

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(*prefix*, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return string[len(prefix):]. Otherwise, return a copy of the original string.

removesuffix(*suffix*, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return string[:-len(suffix)]. Otherwise, return a copy of the original string.

replace(*old*, *new*, *count=-1*, /)

Return a copy with all occurrences of substring *old* replaced by *new*.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(*width*, *fillchar=' '*, /)

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(*sep=None*, *maxsplit=-1*)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to `None` (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(chars=None, /)

Return a copy of the string with trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

split(sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\r` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(keepends=False)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless `keepends` is given and true.

startswith(prefix[, start[, end]]) → bool

Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

strip(chars=None, /)

Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(table, /)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(width, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.ChatPhotoSize(value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None)

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.

New 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`

`__abs__()`

abs(self)

`__add__(value, /)`

Return self+value.

`__and__(value, /)`

Return self&value.

`__bool__()`

True if self else False

`__ceil__()`

Ceiling of an Integral returns itself.

`__divmod__(value, /)`

Return divmod(self, value).

`__eq__(value, /)`

Return self==value.

`__float__()`

float(self)

`__floor__()`

Flooring an Integral returns itself.

`__floordiv__(value, /)`

Return self//value.

`__format__(format_spec, /)`

Convert to a string according to format_spec.

`__ge__(value, /)`

Return self>=value.

`__getattr__(name, /)`

Return getattr(self, name).

`__gt__(value, /)`

Return self>value.

__hash__()

Return hash(self).

__index__()

Return self converted to an integer, if self is suitable for use as an index into a list.

__int__()

int(self)

__invert__()

~self

__le__(value, /)

Return self<=value.

__lshift__(value, /)

Return self<<value.

__lt__(value, /)

Return self<value.

__mod__(value, /)

Return self%value.

__mul__(value, /)

Return self*value.

__ne__(value, /)

Return self!=value.

__neg__()

-self

__new__(value)

__or__(value, /)

Return self|value.

__pos__()

+self

__pow__(value, mod=None, /)

Return pow(self, value, mod).

__radd__(value, /)

Return value+self.

__rand__(value, /)

Return value&self.

__rdivmod__(value, /)

Return divmod(value, self).

__repr__()

Return repr(self).

__rfloordiv__(value, /)

Return value//self.

__rlshift__(value, /)

Return value<<self.

`__rmod__(value, /)`
Return `value%self`.

`__rmul__(value, /)`
Return `value*self`.

`__ror__(value, /)`
Return `value|self`.

`__round__()`
Rounding an Integral returns itself.
Rounding with an `ndigits` argument also returns an integer.

`__rpow__(value, mod=None, /)`
Return `pow(value, self, mod)`.

`__rrshift__(value, /)`
Return `value>>self`.

`__rshift__(value, /)`
Return `self>>value`.

`__rsub__(value, /)`
Return `value-self`.

`__rtruediv__(value, /)`
Return `value/self`.

`__rxor__(value, /)`
Return `value^self`.

`__sizeof__()`
Returns size in memory, in bytes.

`__str__()`
Return `repr(self)`.

`__sub__(value, /)`
Return `self-value`.

`__truediv__(value, /)`
Return `self/value`.

`__trunc__()`
Truncating an Integral returns itself.

`__xor__(value, /)`
Return `self^value`.

`as_integer_ratio()`
Return a pair of integers, whose ratio is equal to the original int.
The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```


bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with float.is_integer.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An OverflowError is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of

the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use `'big'`.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an `OverflowError` is raised.

class telegram.constants.**ChatType**(value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None)

Bases: `str`, `enum.Enum`

This enum contains the available types of `telegram.Chat`. The enum members of this enumeration are instances of `str` and can be treated as such.

New in version 20.0.

CHANNEL = `'channel'`

A `telegram.Chat` that is a channel.

Type

`str`

GROUP = `'group'`

A `telegram.Chat` that is a group.

Type

`str`

PRIVATE = `'private'`

A `telegram.Chat` that is private.

Type

`str`

SENDER = `'sender'`

A `telegram.Chat` that represents the chat of a `telegram.User` sending an `telegram.InlineQuery`.

Type

`str`

SUPERGROUP = `'supergroup'`

A `telegram.Chat` that is a supergroup.

Type

`str`

__add__(value, /)

Return self+value.

__contains__(key, /)

Return bool(key in self).

__eq__(value, /)

Return self==value.

__format__(format_spec)

Return a formatted version of the string as described by format_spec.

__ge__(value, /)

Return self>=value.

__getattr__(name, /)

Return getattr(self, name).

__getitem__(*key*, /)

Return self[key].

__gt__(*value*, /)

Return self>value.

__hash__()

Return hash(self).

__iter__()

Implement iter(self).

__le__(*value*, /)

Return self<=value.

__len__()

Return len(self).

__lt__(*value*, /)

Return self<value.

__mod__(*value*, /)

Return self%value.

__mul__(*value*, /)

Return self*value.

__ne__(*value*, /)

Return self!=value.

__new__(*value*)

__repr__()

Return repr(self).

__rmod__(*value*, /)

Return value%self.

__rmul__(*value*, /)

Return value*self.

__sizeof__()

Return the size of the string in memory, in bytes.

__str__()

Return str(self).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(*width*, *fillchar*=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(*sub*[, *start*[, *end*]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

encode(*encoding*='utf-8', *errors*='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a `UnicodeEncodeError`. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with `codecs.register_error` that can handle `UnicodeEncodeErrors`.

endswith(*suffix*[, *start*[, *end*]]) → bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. *suffix* can also be a tuple of strings to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int

Return the lowest index in S where substring *sub* is found, such that *sub* is contained within S[start:end]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → str

Return a formatted version of S, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → str

Return a formatted version of S, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in S where substring *sub* is found, such that *sub* is contained within S[start:end]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in `x` will be mapped to the character at the same position in `y`. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(*prefix*, /)

Return a `str` with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(*suffix*, /)

Return a `str` with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(*old*, *new*, *count=-1*, /)

Return a copy with all occurrences of substring `old` replaced by `new`.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument `count` is given, only the first `count` occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in `S` where substring `sub` is found, such that `sub` is contained within `S[start:end]`. Optional arguments `start` and `end` are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in `S` where substring `sub` is found, such that `sub` is contained within `S[start:end]`. Optional arguments `start` and `end` are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(*width*, *fillchar=' '*, /)

Return a right-justified string of length `width`.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(*sep=None*, *maxsplit=-1*)

Return a list of the substrings in the string, using `sep` as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars=None, /*)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

split(*sep=None, maxsplit=-1*)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix[, start[, end]]*) → bool

Return True if *S* starts with the specified prefix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *prefix* can also be a tuple of strings to try.

strip(*chars=None, /*)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table, /*)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(width, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.**ContactLimit**(value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None)

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.

New 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`

__abs__()

abs(self)

__add__(value, /)

Return self+value.

__and__(value, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

__divmod__(value, /)

Return divmod(self, value).

__eq__(value, /)

Return self==value.

__float__()

float(self)

__floor__()

Flooring an Integral returns itself.

__floordiv__(value, /)

Return self//value.

__format__(format_spec, /)

Convert to a string according to format_spec.

__ge__(value, /)

Return self>=value.

__getattribute__(*name*, /)
Return getattr(self, name).

__gt__(*value*, /)
Return self>value.

__hash__()
Return hash(self).

__index__()
Return self converted to an integer, if self is suitable for use as an index into a list.

__int__()
int(self)

__invert__()
~self

__le__(*value*, /)
Return self<=value.

__lshift__(*value*, /)
Return self<<value.

__lt__(*value*, /)
Return self<value.

__mod__(*value*, /)
Return self%value.

__mul__(*value*, /)
Return self*value.

__ne__(*value*, /)
Return self!=value.

__neg__()
-self

__new__(*value*)

__or__(*value*, /)
Return self|value.

__pos__()
+self

__pow__(*value*, *mod*=None, /)
Return pow(self, value, mod).

__radd__(*value*, /)
Return value+self.

__rand__(*value*, /)
Return value&self.

__rdivmod__(*value*, /)
Return divmod(value, self).

__repr__()
Return repr(self).

`__rfloordiv__`(*value*, /)

Return `value//self`.

`__rlshift__`(*value*, /)

Return `value<<self`.

`__rmod__`(*value*, /)

Return `value%self`.

`__rmul__`(*value*, /)

Return `value*self`.

`__ror__`(*value*, /)

Return `value|self`.

`__round__`()

Rounding an Integral returns itself.

Rounding with an `ndigits` argument also returns an integer.

`__rpow__`(*value*, *mod*=None, /)

Return `pow(value, self, mod)`.

`__rrshift__`(*value*, /)

Return `value>>self`.

`__rshift__`(*value*, /)

Return `self>>value`.

`__rsub__`(*value*, /)

Return `value-self`.

`__rtruediv__`(*value*, /)

Return `value/self`.

`__rxor__`(*value*, /)

Return `value^self`.

`__sizeof__`()

Returns size in memory, in bytes.

`__str__`()

Return `repr(self)`.

`__sub__`(*value*, /)

Return `self-value`.

`__truediv__`(*value*, /)

Return `self/value`.

`__trunc__`()

Truncating an Integral returns itself.

`__xor__`(*value*, /)

Return `self^value`.

`as_integer_ratio`()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(length=1, byteorder='big', *, signed=False)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an `OverflowError` is raised.

class telegram.constants.**CustomEmojiStickerLimit**(value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None)

Bases: `enum.IntEnum`

This enum contains limitations for `telegram.Bot.get_custom_emoji_stickers()`. The enum members of this enumeration are instances of `int` and can be treated as such.

New in version 20.0.

CUSTOM_EMOJI_IDENTIFIER_LIMIT = 200

Maximum amount of custom emoji identifiers which can be specified for the `custom_emoji_ids` parameter of `telegram.Bot.get_custom_emoji_stickers()`.

Type

`int`

__abs__()

abs(self)

__add__(value, /)

Return self+value.

__and__(value, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

__divmod__(value, /)

Return divmod(self, value).

__eq__(value, /)

Return self==value.

__float__()

float(self)

__floor__()

Flooring an Integral returns itself.

__floordiv__(value, /)

Return self//value.

`__format__`(*format_spec*, /)
Convert to a string according to *format_spec*.

`__ge__`(*value*, /)
Return *self* >= *value*.

`__getattr__`(*name*, /)
Return `getattr(self, name)`.

`__gt__`(*value*, /)
Return *self* > *value*.

`__hash__`()
Return `hash(self)`.

`__index__`()
Return *self* converted to an integer, if *self* is suitable for use as an index into a list.

`__int__`()
`int(self)`

`__invert__`()
~*self*

`__le__`(*value*, /)
Return *self* <= *value*.

`__lshift__`(*value*, /)
Return *self* << *value*.

`__lt__`(*value*, /)
Return *self* < *value*.

`__mod__`(*value*, /)
Return *self* % *value*.

`__mul__`(*value*, /)
Return *self* * *value*.

`__ne__`(*value*, /)
Return *self* != *value*.

`__neg__`()
-*self*

`__new__`(*value*)

`__or__`(*value*, /)
Return *self* | *value*.

`__pos__`()
+*self*

`__pow__`(*value*, *mod*=None, /)
Return `pow(self, value, mod)`.

`__radd__`(*value*, /)
Return *value* + *self*.

`__rand__`(*value*, /)
Return *value* & *self*.

`__rdivmod__`(*value*, /)
Return divmod(*value*, self).

`__repr__`()
Return repr(self).

`__rfloordiv__`(*value*, /)
Return *value*//self.

`__rlshift__`(*value*, /)
Return *value*<<self.

`__rmod__`(*value*, /)
Return *value*%self.

`__rmul__`(*value*, /)
Return *value**self.

`__ror__`(*value*, /)
Return *value*|self.

`__round__`()
Rounding an Integral returns itself.
Rounding with an *ndigits* argument also returns an integer.

`__rpow__`(*value*, *mod*=None, /)
Return pow(*value*, self, *mod*).

`__rrshift__`(*value*, /)
Return *value*>>self.

`__rshift__`(*value*, /)
Return self>>*value*.

`__rsub__`(*value*, /)
Return *value*-self.

`__rtruediv__`(*value*, /)
Return *value*/self.

`__rxor__`(*value*, /)
Return *value*^self.

`__sizeof__`()
Returns size in memory, in bytes.

`__str__`()
Return repr(self).

`__sub__`(*value*, /)
Return self-*value*.

`__truediv__`(*value*, /)
Return self/*value*.

`__trunc__`()
Truncating an Integral returns itself.

`__xor__`(*value*, /)
Return self^*value*.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with float.is_integer.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If `byteorder` is 'big', the most significant byte is at the beginning of the byte array. If `byteorder` is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If `signed` is `False` and a negative integer is given, an `OverflowError` is raised.

class telegram.constants.**DiceEmoji**(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `str`, `enum.Enum`

This enum contains the available emoji for `telegram.Dice/ telegram.Bot.send_dice()`. The enum members of this enumeration are instances of `str` and can be treated as such.

New in version 20.0.

BASKETBALL = ''

A `telegram.Dice` with the emoji .

Type

`str`

BOWLING = ''

A `telegram.Dice` with the emoji .

Type

`str`

DARTS = ''

A `telegram.Dice` with the emoji .

Type

`str`

DICE = ''

A `telegram.Dice` with the emoji .

Type

`str`

FOOTBALL = ''

A `telegram.Dice` with the emoji .

Type

`str`

SLOT_MACHINE = ''

A `telegram.Dice` with the emoji .

Type

`str`

__add__(*value*, /)
Return self+value.

__contains__(*key*, /)
Return bool(key in self).

__eq__(*value*, /)
Return self==value.

__format__(*format_spec*)
Return a formatted version of the string as described by format_spec.

__ge__(*value*, /)
Return self>=value.

__getattr__(*name*, /)
Return getattr(self, name).

__getitem__(*key*, /)
Return self[key].

__gt__(*value*, /)
Return self>value.

__hash__()
Return hash(self).

__iter__()
Implement iter(self).

__le__(*value*, /)
Return self<=value.

__len__()
Return len(self).

__lt__(*value*, /)
Return self<value.

__mod__(*value*, /)
Return self%value.

__mul__(*value*, /)
Return self*value.

__ne__(*value*, /)
Return self!=value.

__new__(*value*)

__repr__()
Return repr(self).

__rmod__(*value*, /)
Return value%self.

__rmul__(*value*, /)
Return value*self.

__sizeof__()
Return the size of the string in memory, in bytes.

__str__()

Return str(self).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(width, fillchar=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(sub[, start[, end]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

encode(encoding='utf-8', errors='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith(suffix[, start[, end]]) → bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

expandtabs(tabsize=8)

Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

find(sub[, start[, end]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

format(*args, **kwargs) → str

Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

format_map(mapping) → str

Return a formatted version of S, using substitutions from mapping. The substitutions are identified by braces ('{' and '}').

index(sub[, start[, end]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(prefix, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(suffix, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(old, new, count=-1, /)

Return a copy with all occurrences of substring old replaced by new.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(*width*, *fillchar*=' ', /)

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars*=None, /)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

split(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends*=False)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if *S* starts with the specified prefix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *prefix* can also be a tuple of strings to try.

strip(*chars=None*, /)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not *None*, remove characters in *chars* instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table*, /)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or *None*.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to *None* are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(*width*, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.DiceLimit(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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.

New 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

__abs__()

abs(self)

__add__(value, /)

Return self+value.

__and__(value, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

__divmod__(value, /)

Return divmod(self, value).

__eq__(value, /)

Return self==value.

__float__()

float(self)

__floor__()

Flooring an Integral returns itself.

__floordiv__(value, /)

Return self//value.

__format__(format_spec, /)

Convert to a string according to format_spec.

__ge__(value, /)

Return self>=value.

__getattr__(name, /)

Return getattr(self, name).

__gt__(value, /)

Return self>value.

__hash__()

Return hash(self).

`__index__()`

Return self converted to an integer, if self is suitable for use as an index into a list.

`__int__()`

`int(self)`

`__invert__()`

`~self`

`__le__(value, /)`

Return `self <= value`.

`__lshift__(value, /)`

Return `self << value`.

`__lt__(value, /)`

Return `self < value`.

`__mod__(value, /)`

Return `self % value`.

`__mul__(value, /)`

Return `self * value`.

`__ne__(value, /)`

Return `self != value`.

`__neg__()`

`-self`

`__new__(value)`

`__or__(value, /)`

Return `self | value`.

`__pos__()`

`+self`

`__pow__(value, mod=None, /)`

Return `pow(self, value, mod)`.

`__radd__(value, /)`

Return `value + self`.

`__rand__(value, /)`

Return `value & self`.

`__rdivmod__(value, /)`

Return `divmod(value, self)`.

`__repr__()`

Return `repr(self)`.

`__rfloordiv__(value, /)`

Return `value // self`.

`__rlshift__(value, /)`

Return `value << self`.

`__rmod__(value, /)`

Return `value % self`.

__rmul__(value, /)

Return value*self.

__ror__(value, /)

Return value|self.

__round__()

Rounding an Integral returns itself.

Rounding with an ndigits argument also returns an integer.

__rpow__(value, mod=None, /)

Return pow(value, self, mod).

__rrshift__(value, /)

Return value>>self.

__rshift__(value, /)

Return self>>value.

__rsub__(value, /)

Return value-self.

__rtruediv__(value, /)

Return value/self.

__rxor__(value, /)

Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

```
class telegram.constants.FileSizeLimit(value, names=None, *values, module=None,
                                       qualname=None, type=None, start=1, boundary=None)
```

Bases: `enum.IntEnum`

This enum contains limitations regarding the upload and download of files. The enum members of this enumeration are instances of `int` and can be treated as such.

New in version 20.0.

FILESIZE_DOWNLOAD = 200000000

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 = 500000000

Bots can upload non-photo files of up to 50MB in size.

Type

`int`

FILESIZE_UPLOAD_LOCAL_MODE = 20000000000

Bots can upload non-photo files of up to 2000MB in size when using a local bot API server.

Type

`int`

PHOTOSIZE_UPLOAD = 100000000

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`

__abs__()

abs(self)

__add__(value, /)

Return self+value.

__and__(value, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

`__divmod__`(*value*, /)
Return divmod(self, value).

`__eq__`(*value*, /)
Return self==value.

`__float__`()
float(self)

`__floor__`()
Flooring an Integral returns itself.

`__floordiv__`(*value*, /)
Return self//value.

`__format__`(*format_spec*, /)
Convert to a string according to format_spec.

`__ge__`(*value*, /)
Return self>=value.

`__getattr__`(*name*, /)
Return getattr(self, name).

`__gt__`(*value*, /)
Return self>value.

`__hash__`()
Return hash(self).

`__index__`()
Return self converted to an integer, if self is suitable for use as an index into a list.

`__int__`()
int(self)

`__invert__`()
~self

`__le__`(*value*, /)
Return self<=value.

`__lshift__`(*value*, /)
Return self<<value.

`__lt__`(*value*, /)
Return self<value.

`__mod__`(*value*, /)
Return self%value.

`__mul__`(*value*, /)
Return self*value.

`__ne__`(*value*, /)
Return self!=value.

`__neg__`()
-self

`__new__`(*value*)

__or__(*value*, /)
Return self|value.

__pos__()
+self

__pow__(*value*, *mod*=None, /)
Return pow(self, value, mod).

__radd__(*value*, /)
Return value+self.

__rand__(*value*, /)
Return value&self.

__rdivmod__(*value*, /)
Return divmod(value, self).

__repr__()
Return repr(self).

__rfloordiv__(*value*, /)
Return value//self.

__rlshift__(*value*, /)
Return value<<self.

__rmod__(*value*, /)
Return value%self.

__rmul__(*value*, /)
Return value*self.

__ror__(*value*, /)
Return value|self.

__round__()
Rounding an Integral returns itself.
Rounding with an ndigits argument also returns an integer.

__rpow__(*value*, *mod*=None, /)
Return pow(value, self, mod).

__rrshift__(*value*, /)
Return value>>self.

__rshift__(*value*, /)
Return self>>value.

__rsub__(*value*, /)
Return value-self.

__rtruediv__(*value*, /)
Return value/self.

__rxor__(*value*, /)
Return value^self.

__sizeof__()
Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of

the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use `'big'`.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If `byteorder` is `'big'`, the most significant byte is at the beginning of the byte array. If `byteorder` is `'little'`, the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use `'big'`.

signed

Determines whether two's complement is used to represent the integer. If `signed` is `False` and a negative integer is given, an `OverflowError` is raised.

class telegram.constants.FloodLimit(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `enum.IntEnum`

This enum contains limitations regarding flood limits. The enum members of this enumeration are instances of `int` and can be treated as such.

New in version 20.0.

MESSAGES_PER_MINUTE_PER_GROUP = 20

The number of messages that can roughly be sent to a particular group within one minute.

Type

`int`

MESSAGES_PER_SECOND = 30

The number of messages that can roughly be sent in an interval of 30 seconds across all chats.

Type

`int`

MESSAGES_PER_SECOND_PER_CHAT = 1

The number of messages that can be sent per second in a particular chat. Telegram may allow short bursts that go over this limit, but eventually you'll begin receiving 429 errors.

Type

`int`

__abs__()

`abs(self)`

__add__(*value*, /)
Return self+value.

__and__(*value*, /)
Return self&value.

__bool__()
True if self else False

__ceil__()
Ceiling of an Integral returns itself.

__divmod__(*value*, /)
Return divmod(self, value).

__eq__(*value*, /)
Return self==value.

__float__()
float(self)

__floor__()
Flooring an Integral returns itself.

__floordiv__(*value*, /)
Return self//value.

__format__(*format_spec*, /)
Convert to a string according to format_spec.

__ge__(*value*, /)
Return self>=value.

__getattr__(*name*, /)
Return getattr(self, name).

__gt__(*value*, /)
Return self>value.

__hash__()
Return hash(self).

__index__()
Return self converted to an integer, if self is suitable for use as an index into a list.

__int__()
int(self)

__invert__()
~self

__le__(*value*, /)
Return self<=value.

__lshift__(*value*, /)
Return self<<value.

__lt__(*value*, /)
Return self<value.

__mod__(*value*, /)
Return self%value.

`__mul__(value, /)`
Return `self*value`.

`__ne__(value, /)`
Return `self!=value`.

`__neg__()`
-self

`__new__(value)`

`__or__(value, /)`
Return `self|value`.

`__pos__()`
+self

`__pow__(value, mod=None, /)`
Return `pow(self, value, mod)`.

`__radd__(value, /)`
Return `value+self`.

`__rand__(value, /)`
Return `value&self`.

`__rdivmod__(value, /)`
Return `divmod(value, self)`.

`__repr__()`
Return `repr(self)`.

`__rfloordiv__(value, /)`
Return `value//self`.

`__rlshift__(value, /)`
Return `value<<self`.

`__rmod__(value, /)`
Return `value%self`.

`__rmul__(value, /)`
Return `value*self`.

`__ror__(value, /)`
Return `value|self`.

`__round__()`
Rounding an Integral returns itself.
Rounding with an `ndigits` argument also returns an integer.

`__rpow__(value, mod=None, /)`
Return `pow(value, self, mod)`.

`__rrshift__(value, /)`
Return `value>>self`.

`__rshift__(value, /)`
Return `self>>value`.

`__rsub__(value, /)`
Return `value-self`.

__rtruediv__(value, /)

Return value/self.

__rxor__(value, /)

Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an `OverflowError` is raised.

class telegram.constants.**ForumIconColor**(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

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.

New 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`

`__abs__()`

`abs(self)`

`__add__(value, /)`

Return self+value.

`__and__(value, /)`

Return self&value.

`__bool__()`

True if self else False

`__ceil__()`

Ceiling of an Integral returns itself.

`__divmod__(value, /)`

Return divmod(self, value).

`__eq__(value, /)`

Return self==value.

`__float__()`

`float(self)`

`__floor__()`

Flooring an Integral returns itself.

`__floordiv__(value, /)`

Return self//value.

`__format__`(*format_spec*, /)
Convert to a string according to *format_spec*.

`__ge__`(*value*, /)
Return *self* >= *value*.

`__getattr__`(*name*, /)
Return `getattr(self, name)`.

`__gt__`(*value*, /)
Return *self* > *value*.

`__hash__`()
Return `hash(self)`.

`__index__`()
Return *self* converted to an integer, if *self* is suitable for use as an index into a list.

`__int__`()
`int(self)`

`__invert__`()
~*self*

`__le__`(*value*, /)
Return *self* <= *value*.

`__lshift__`(*value*, /)
Return *self* << *value*.

`__lt__`(*value*, /)
Return *self* < *value*.

`__mod__`(*value*, /)
Return *self* % *value*.

`__mul__`(*value*, /)
Return *self* * *value*.

`__ne__`(*value*, /)
Return *self* != *value*.

`__neg__`()
-*self*

`__new__`(*value*)

`__or__`(*value*, /)
Return *self* | *value*.

`__pos__`()
+*self*

`__pow__`(*value*, *mod*=None, /)
Return `pow(self, value, mod)`.

`__radd__`(*value*, /)
Return *value* + *self*.

`__rand__`(*value*, /)
Return *value* & *self*.

`__rdivmod__`(*value*, /)
Return divmod(*value*, self).

`__repr__`()
Return repr(self).

`__rfloordiv__`(*value*, /)
Return *value*//self.

`__rlshift__`(*value*, /)
Return *value*<<self.

`__rmod__`(*value*, /)
Return *value*%self.

`__rmul__`(*value*, /)
Return *value**self.

`__ror__`(*value*, /)
Return *value*|self.

`__round__`()
Rounding an Integral returns itself.
Rounding with an *ndigits* argument also returns an integer.

`__rpow__`(*value*, *mod*=None, /)
Return pow(*value*, self, *mod*).

`__rrshift__`(*value*, /)
Return *value*>>self.

`__rshift__`(*value*, /)
Return self>>*value*.

`__rsub__`(*value*, /)
Return *value*-self.

`__rtruediv__`(*value*, /)
Return *value*/self.

`__rxor__`(*value*, /)
Return *value*^self.

`__sizeof__`()
Returns size in memory, in bytes.

`__str__`()
Return repr(self).

`__sub__`(*value*, /)
Return self-*value*.

`__truediv__`(*value*, /)
Return self/*value*.

`__trunc__`()
Truncating an Integral returns itself.

`__xor__`(*value*, /)
Return self^*value*.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with float.is_integer.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If `byteorder` is `'big'`, the most significant byte is at the beginning of the byte array. If `byteorder` is `'little'`, the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use `'big'`.

signed

Determines whether two's complement is used to represent the integer. If `signed` is `False` and a negative integer is given, an `OverflowError` is raised.

class telegram.constants.**ForumTopicLimit**(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

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.

New 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`

__abs__()

`abs(self)`

__add__(*value, /*)

Return `self+value`.

__and__(*value, /*)

Return `self&value`.

`__bool__()`
True if self else False

`__ceil__()`
Ceiling of an Integral returns itself.

`__divmod__(value, /)`
Return divmod(self, value).

`__eq__(value, /)`
Return self==value.

`__float__()`
float(self)

`__floor__()`
Flooring an Integral returns itself.

`__floordiv__(value, /)`
Return self//value.

`__format__(format_spec, /)`
Convert to a string according to format_spec.

`__ge__(value, /)`
Return self>=value.

`__getattr__(name, /)`
Return getattr(self, name).

`__gt__(value, /)`
Return self>value.

`__hash__()`
Return hash(self).

`__index__()`
Return self converted to an integer, if self is suitable for use as an index into a list.

`__int__()`
int(self)

`__invert__()`
~self

`__le__(value, /)`
Return self<=value.

`__lshift__(value, /)`
Return self<<value.

`__lt__(value, /)`
Return self<value.

`__mod__(value, /)`
Return self%value.

`__mul__(value, /)`
Return self*value.

`__ne__(value, /)`
Return self!=value.

__neg__()
-self

__new__(value)

__or__(value, /)
Return self|value.

__pos__()
+self

__pow__(value, mod=None, /)
Return pow(self, value, mod).

__radd__(value, /)
Return value+self.

__rand__(value, /)
Return value&self.

__rdivmod__(value, /)
Return divmod(value, self).

__repr__()
Return repr(self).

__rfloordiv__(value, /)
Return value//self.

__rlshift__(value, /)
Return value<<self.

__rmod__(value, /)
Return value%self.

__rmul__(value, /)
Return value*self.

__ror__(value, /)
Return value|self.

__round__()
Rounding an Integral returns itself.
Rounding with an ndigits argument also returns an integer.

__rpow__(value, mod=None, /)
Return pow(value, self, mod).

__rrshift__(value, /)
Return value>>self.

__rshift__(value, /)
Return self>>value.

__rsub__(value, /)
Return value-self.

__rtruediv__(value, /)
Return value/self.

__rxor__(value, /)
Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an `OverflowError` is raised.

class telegram.constants.**GiveawayLimit**(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

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.

New in version 20.8.

MAX_WINNERS = 100

Maximum number of winners allowed for `telegram.GiveawayWinners.winners`.

Type

`int`

__abs__()

`abs(self)`

__add__(*value, /*)

Return self+value.

__and__(*value, /*)

Return self&value.

__bool__()

True if self else False

`__ceil__()`
Ceiling of an Integral returns itself.

`__divmod__(value, /)`
Return `divmod(self, value)`.

`__eq__(value, /)`
Return `self==value`.

`__float__()`
`float(self)`

`__floor__()`
Flooring an Integral returns itself.

`__floordiv__(value, /)`
Return `self//value`.

`__format__(format_spec, /)`
Convert to a string according to `format_spec`.

`__ge__(value, /)`
Return `self>=value`.

`__getattr__(name, /)`
Return `getattr(self, name)`.

`__gt__(value, /)`
Return `self>value`.

`__hash__()`
Return `hash(self)`.

`__index__()`
Return `self` converted to an integer, if `self` is suitable for use as an index into a list.

`__int__()`
`int(self)`

`__invert__()`
`~self`

`__le__(value, /)`
Return `self<=value`.

`__lshift__(value, /)`
Return `self<<value`.

`__lt__(value, /)`
Return `self<value`.

`__mod__(value, /)`
Return `self%value`.

`__mul__(value, /)`
Return `self*value`.

`__ne__(value, /)`
Return `self!=value`.

`__neg__()`
`-self`

`__new__(value)`
Return self|value.

`__or__(value, /)`
Return self|value.

`__pos__()`
+self

`__pow__(value, mod=None, /)`
Return pow(self, value, mod).

`__radd__(value, /)`
Return value+self.

`__rand__(value, /)`
Return value&self.

`__rdivmod__(value, /)`
Return divmod(value, self).

`__repr__()`
Return repr(self).

`__rfloordiv__(value, /)`
Return value//self.

`__rlshift__(value, /)`
Return value<<self.

`__rmod__(value, /)`
Return value%self.

`__rmul__(value, /)`
Return value*self.

`__ror__(value, /)`
Return value|self.

`__round__()`
Rounding an Integral returns itself.
Rounding with an ndigits argument also returns an integer.

`__rpow__(value, mod=None, /)`
Return pow(value, self, mod).

`__rrshift__(value, /)`
Return value>>self.

`__rshift__(value, /)`
Return self>>value.

`__rsub__(value, /)`
Return value-self.

`__rtruediv__(value, /)`
Return value/self.

`__rxor__(value, /)`
Return value^self.

`__sizeof__()`
Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of

the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use `'big'`.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(length=1, byteorder='big', *, signed=False)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If `byteorder` is `'big'`, the most significant byte is at the beginning of the byte array. If `byteorder` is `'little'`, the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use `'big'`.

signed

Determines whether two's complement is used to represent the integer. If `signed` is `False` and a negative integer is given, an `OverflowError` is raised.

class telegram.constants.**InlineKeyboardButtonLimit**(value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None)

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.

New in version 20.0.

MAX_CALLBACK_DATA = 64

Maximum value allowed for `callback_data` parameter of `telegram.InlineKeyboardButton`

Type

`int`

MIN_CALLBACK_DATA = 1

Minimum value allowed for `callback_data` parameter of `telegram.InlineKeyboardButton`

Type

`int`

__abs__()

`abs(self)`

__add__(value, /)

Return self+value.

__and__(value, /)

Return self&value.

`__bool__()`
True if self else False

`__ceil__()`
Ceiling of an Integral returns itself.

`__divmod__(value, /)`
Return divmod(self, value).

`__eq__(value, /)`
Return self==value.

`__float__()`
float(self)

`__floor__()`
Flooring an Integral returns itself.

`__floordiv__(value, /)`
Return self//value.

`__format__(format_spec, /)`
Convert to a string according to format_spec.

`__ge__(value, /)`
Return self>=value.

`__getattr__(name, /)`
Return getattr(self, name).

`__gt__(value, /)`
Return self>value.

`__hash__()`
Return hash(self).

`__index__()`
Return self converted to an integer, if self is suitable for use as an index into a list.

`__int__()`
int(self)

`__invert__()`
~self

`__le__(value, /)`
Return self<=value.

`__lshift__(value, /)`
Return self<<value.

`__lt__(value, /)`
Return self<value.

`__mod__(value, /)`
Return self%value.

`__mul__(value, /)`
Return self*value.

`__ne__(value, /)`
Return self!=value.

__neg__()
-self

__new__(value)

__or__(value, /)
Return self|value.

__pos__()
+self

__pow__(value, mod=None, /)
Return pow(self, value, mod).

__radd__(value, /)
Return value+self.

__rand__(value, /)
Return value&self.

__rdivmod__(value, /)
Return divmod(value, self).

__repr__()
Return repr(self).

__rfloordiv__(value, /)
Return value//self.

__rlshift__(value, /)
Return value<<self.

__rmod__(value, /)
Return value%self.

__rmul__(value, /)
Return value*self.

__ror__(value, /)
Return value|self.

__round__()
Rounding an Integral returns itself.
Rounding with an ndigits argument also returns an integer.

__rpow__(value, mod=None, /)
Return pow(value, self, mod).

__rrshift__(value, /)
Return value>>self.

__rshift__(value, /)
Return self>>value.

__rsub__(value, /)
Return value-self.

__rtruediv__(value, /)
Return value/self.

__rxor__(value, /)
Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with float.is_integer.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(length=1, byteorder='big', *, signed=False)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An OverflowError is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

class telegram.constants.**InlineKeyboardMarkupLimit**(value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None)

Bases: `enum.IntEnum`

This enum contains limitations for `telegram.InlineKeyboardMarkup`/ `telegram.Bot.send_message()` & friends. The enum members of this enumeration are instances of `int` and can be treated as such.

New in version 20.0.

BUTTONS_PER_ROW = 8

Maximum number of buttons that can be attached to a message per row.

Note: This value is undocumented and might be changed by Telegram.

Type

`int`

TOTAL_BUTTON_NUMBER = 100

Maximum number of buttons that can be attached to a message.

Note: This value is undocumented and might be changed by Telegram.

Type`int`**`__abs__()`**`abs(self)`**`__add__(value, /)`**`Return self+value.`**`__and__(value, /)`**`Return self&value.`**`__bool__()`**`True if self else False`**`__ceil__()`**`Ceiling of an Integral returns itself.`**`__divmod__(value, /)`**`Return divmod(self, value).`**`__eq__(value, /)`**`Return self==value.`**`__float__()`**`float(self)`**`__floor__()`**`Flooring an Integral returns itself.`**`__floordiv__(value, /)`**`Return self//value.`**`__format__(format_spec, /)`**`Convert to a string according to format_spec.`**`__ge__(value, /)`**`Return self>=value.`**`__getattr__(name, /)`**`Return getattr(self, name).`**`__gt__(value, /)`**`Return self>value.`**`__hash__()`**`Return hash(self).`**`__index__()`**`Return self converted to an integer, if self is suitable for use as an index into a list.`**`__int__()`**`int(self)`**`__invert__()`**`~self`

`__le__(value, /)`
Return `self <= value`.

`__lshift__(value, /)`
Return `self << value`.

`__lt__(value, /)`
Return `self < value`.

`__mod__(value, /)`
Return `self % value`.

`__mul__(value, /)`
Return `self * value`.

`__ne__(value, /)`
Return `self != value`.

`__neg__()`
-self

`__new__(value)`

`__or__(value, /)`
Return `self | value`.

`__pos__()`
+self

`__pow__(value, mod=None, /)`
Return `pow(self, value, mod)`.

`__radd__(value, /)`
Return `value + self`.

`__rand__(value, /)`
Return `value & self`.

`__rdivmod__(value, /)`
Return `divmod(value, self)`.

`__repr__()`
Return `repr(self)`.

`__rfloordiv__(value, /)`
Return `value // self`.

`__rlshift__(value, /)`
Return `value << self`.

`__rmod__(value, /)`
Return `value % self`.

`__rmul__(value, /)`
Return `value * self`.

`__ror__(value, /)`
Return `value | self`.

`__round__()`
Rounding an Integral returns itself.
Rounding with an `ndigits` argument also returns an integer.

__rpow__(*value*, *mod=None*, /)
Return pow(*value*, self, *mod*).

__rrshift__(*value*, /)
Return value>>self.

__rshift__(*value*, /)
Return self>>value.

__rsub__(*value*, /)
Return value-self.

__rtruediv__(*value*, /)
Return value/self.

__rxor__(*value*, /)
Return value^self.

__sizeof__()
Returns size in memory, in bytes.

__str__()
Return repr(self).

__sub__(*value*, /)
Return self-value.

__truediv__(*value*, /)
Return self/value.

__trunc__()
Truncating an Integral returns itself.

__xor__(*value*, /)
Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an `OverflowError` is raised.

class telegram.constants.**InlineQueryLimit**(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `enum.IntEnum`

This enum contains limitations for `telegram.InlineQuery/telegram.Bot.answer_inline_query()`. The enum members of this enumeration are instances of `int` and can be treated as such.

New in version 20.0.

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`

MAX_SWITCH_PM_TEXT_LENGTH = 64

Maximum number of characters in a `str` passed as the `switch_pm_parameter` parameter of `telegram.Bot.answer_inline_query()`.

Deprecated since version 20.3: Deprecated in favor of `InlineQueryResultsButtonLimit.MAX_START_PARAMETER_LENGTH`.

Type

`int`

MIN_SWITCH_PM_TEXT_LENGTH = 1

Minimum number of characters in a `str` passed as the `switch_pm_parameter` parameter of `telegram.Bot.answer_inline_query()`.

Deprecated since version 20.3: Deprecated in favor of `InlineQueryResultsButtonLimit.MIN_START_PARAMETER_LENGTH`.

Type

`int`

RESULTS = 50

Maximum number of results that can be passed to `telegram.Bot.answer_inline_query()`.

Type

`int`

__abs__()

`abs(self)`

__add__(value, /)

Return self+value.

__and__(value, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

__divmod__(value, /)

Return divmod(self, value).

__eq__(value, /)

Return self==value.

__float__()

`float(self)`

`__floor__()`

Flooring an Integral returns itself.

`__floordiv__(value, /)`

Return self//value.

`__format__(format_spec, /)`

Convert to a string according to format_spec.

`__ge__(value, /)`

Return self>=value.

`__getattr__(name, /)`

Return getattr(self, name).

`__gt__(value, /)`

Return self>value.

`__hash__()`

Return hash(self).

`__index__()`

Return self converted to an integer, if self is suitable for use as an index into a list.

`__int__()`

int(self)

`__invert__()`

~self

`__le__(value, /)`

Return self<=value.

`__lshift__(value, /)`

Return self<<value.

`__lt__(value, /)`

Return self<value.

`__mod__(value, /)`

Return self%value.

`__mul__(value, /)`

Return self*value.

`__ne__(value, /)`

Return self!=value.

`__neg__()`

-self

`__new__(value)`

`__or__(value, /)`

Return self|value.

`__pos__()`

+self

`__pow__(value, mod=None, /)`

Return pow(self, value, mod).

`__radd__(value, /)`
Return value+self.

`__rand__(value, /)`
Return value&self.

`__rdivmod__(value, /)`
Return divmod(value, self).

`__repr__()`
Return repr(self).

`__rfloordiv__(value, /)`
Return value//self.

`__rlshift__(value, /)`
Return value<<self.

`__rmod__(value, /)`
Return value%self.

`__rmul__(value, /)`
Return value*self.

`__ror__(value, /)`
Return value|self.

`__round__()`
Rounding an Integral returns itself.
Rounding with an ndigits argument also returns an integer.

`__rpow__(value, mod=None, /)`
Return pow(value, self, mod).

`__rrshift__(value, /)`
Return value>>self.

`__rshift__(value, /)`
Return self>>value.

`__rsub__(value, /)`
Return value-self.

`__rtruediv__(value, /)`
Return value/self.

`__rxor__(value, /)`
Return value^self.

`__sizeof__()`
Returns size in memory, in bytes.

`__str__()`
Return repr(self).

`__sub__(value, /)`
Return self-value.

`__truediv__(value, /)`
Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return $\text{self} \wedge \text{value}$.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with float.is_integer.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(length=1, byteorder='big', *, signed=False)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An OverflowError is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

class telegram.constants.**InlineQueryResultLimit**(value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None)

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.

New 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`

__abs__()

abs(self)

__add__(value, /)

Return self+value.

__and__(value, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

`__divmod__`(*value*, /)
Return divmod(self, value).

`__eq__`(*value*, /)
Return self==value.

`__float__`()
float(self)

`__floor__`()
Flooring an Integral returns itself.

`__floordiv__`(*value*, /)
Return self//value.

`__format__`(*format_spec*, /)
Convert to a string according to format_spec.

`__ge__`(*value*, /)
Return self>=value.

`__getattr__`(*name*, /)
Return getattr(self, name).

`__gt__`(*value*, /)
Return self>value.

`__hash__`()
Return hash(self).

`__index__`()
Return self converted to an integer, if self is suitable for use as an index into a list.

`__int__`()
int(self)

`__invert__`()
~self

`__le__`(*value*, /)
Return self<=value.

`__lshift__`(*value*, /)
Return self<<value.

`__lt__`(*value*, /)
Return self<value.

`__mod__`(*value*, /)
Return self%value.

`__mul__`(*value*, /)
Return self*value.

`__ne__`(*value*, /)
Return self!=value.

`__neg__`()
-self

`__new__`(*value*)

__or__(*value*, /)
Return self|value.

__pos__()
+self

__pow__(*value*, *mod*=None, /)
Return pow(self, value, mod).

__radd__(*value*, /)
Return value+self.

__rand__(*value*, /)
Return value&self.

__rdivmod__(*value*, /)
Return divmod(value, self).

__repr__()
Return repr(self).

__rfloordiv__(*value*, /)
Return value//self.

__rlshift__(*value*, /)
Return value<<self.

__rmod__(*value*, /)
Return value%self.

__rmul__(*value*, /)
Return value*self.

__ror__(*value*, /)
Return value|self.

__round__()
Rounding an Integral returns itself.
Rounding with an ndigits argument also returns an integer.

__rpow__(*value*, *mod*=None, /)
Return pow(value, self, mod).

__rrshift__(*value*, /)
Return value>>self.

__rshift__(*value*, /)
Return self>>value.

__rsub__(*value*, /)
Return value-self.

__rtruediv__(*value*, /)
Return value/self.

__rxor__(*value*, /)
Return value^self.

__sizeof__()
Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of

the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use `'big'`.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If `byteorder` is `'big'`, the most significant byte is at the beginning of the byte array. If `byteorder` is `'little'`, the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use `'big'`.

signed

Determines whether two's complement is used to represent the integer. If `signed` is `False` and a negative integer is given, an `OverflowError` is raised.

```
class telegram.constants.InlineQueryResultType(value, names=None, *values, module=None,
                                             qualname=None, type=None, start=1,
                                             boundary=None)
```

Bases: `str`, `enum.Enum`

This enum contains the available types of `telegram.InlineQueryResult`. The enum members of this enumeration are instances of `str` and can be treated as such.

New in version 20.0.

ARTICLE = 'article'

Type of `telegram.InlineQueryResultArticle`.

Type

`str`

AUDIO = 'audio'

Type of `telegram.InlineQueryResultAudio` and `telegram.InlineQueryResultCachedAudio`.

Type

`str`

CONTACT = 'contact'

Type of `telegram.InlineQueryResultContact`.

Type

`str`

DOCUMENT = 'document'

Type of `telegram.InlineQueryResultDocument` and `telegram.InlineQueryResultCachedDocument`.

Type
`str`

GAME = 'game'

Type of `telegram.InlineQueryResultGame`.

Type
`str`

GIF = 'gif'

Type of `telegram.InlineQueryResultGif` and `telegram.InlineQueryResultCachedGif`.

Type
`str`

LOCATION = 'location'

Type of `telegram.InlineQueryResultLocation`.

Type
`str`

MPEG4GIF = 'mpeg4_gif'

Type of `telegram.InlineQueryResultMpeg4Gif` and `telegram.InlineQueryResultCachedMpeg4Gif`.

Type
`str`

PHOTO = 'photo'

Type of `telegram.InlineQueryResultPhoto` and `telegram.InlineQueryResultCachedPhoto`.

Type
`str`

STICKER = 'sticker'

Type of and `telegram.InlineQueryResultCachedSticker`.

Type
`str`

VENUE = 'venue'

Type of `telegram.InlineQueryResultVenue`.

Type
`str`

VIDEO = 'video'

Type of `telegram.InlineQueryResultVideo` and `telegram.InlineQueryResultCachedVideo`.

Type
`str`

VOICE = 'voice'

Type of `telegram.InlineQueryResultVoice` and `telegram.InlineQueryResultCachedVoice`.

Type
`str`

`__add__(value, /)`
Return self+value.

`__contains__(key, /)`
Return bool(key in self).

`__eq__(value, /)`
Return self==value.

`__format__(format_spec)`
Return a formatted version of the string as described by format_spec.

`__ge__(value, /)`
Return self>=value.

`__getattr__(name, /)`
Return getattr(self, name).

`__getitem__(key, /)`
Return self[key].

`__gt__(value, /)`
Return self>value.

`__hash__()`
Return hash(self).

`__iter__()`
Implement iter(self).

`__le__(value, /)`
Return self<=value.

`__len__()`
Return len(self).

`__lt__(value, /)`
Return self<value.

`__mod__(value, /)`
Return self%value.

`__mul__(value, /)`
Return self*value.

`__ne__(value, /)`
Return self!=value.

`__new__(value)`

`__repr__()`
Return repr(self).

`__rmod__(value, /)`
Return value%self.

`__rmul__(value, /)`
Return value*self.

`__sizeof__()`
Return the size of the string in memory, in bytes.

__str__()

Return str(self).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(width, fillchar=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(sub[, start[, end]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

encode(encoding='utf-8', errors='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith(suffix[, start[, end]]) → bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

expandtabs(tabsize=8)

Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

find(sub[, start[, end]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

format(*args, **kwargs) → str

Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

format_map(mapping) → str

Return a formatted version of S, using substitutions from mapping. The substitutions are identified by braces ('{' and '}').

index(sub[, start[, end]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs'])` -> `'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(prefix, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(suffix, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(old, new, count=-1, /)

Return a copy with all occurrences of substring old replaced by new.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(*width*, *fillchar*=' ', /)

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars*=None, /)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

split(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends*=False)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if *S* starts with the specified prefix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *prefix* can also be a tuple of strings to try.

strip(chars=None, /)

Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(table, /)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(width, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.**InlineQueryResultsButtonLimit**(value, names=None, *values,
module=None, qualname=None,
type=None, start=1, boundary=None)

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.

New 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`

__abs__()

abs(self)

__add__(value, /)

Return self+value.

__and__(value, /)

Return self&value.

`__bool__()`
True if self else False

`__ceil__()`
Ceiling of an Integral returns itself.

`__divmod__(value, /)`
Return divmod(self, value).

`__eq__(value, /)`
Return self==value.

`__float__()`
float(self)

`__floor__()`
Flooring an Integral returns itself.

`__floordiv__(value, /)`
Return self//value.

`__format__(format_spec, /)`
Convert to a string according to format_spec.

`__ge__(value, /)`
Return self>=value.

`__getattr__(name, /)`
Return getattr(self, name).

`__gt__(value, /)`
Return self>value.

`__hash__()`
Return hash(self).

`__index__()`
Return self converted to an integer, if self is suitable for use as an index into a list.

`__int__()`
int(self)

`__invert__()`
~self

`__le__(value, /)`
Return self<=value.

`__lshift__(value, /)`
Return self<<value.

`__lt__(value, /)`
Return self<value.

`__mod__(value, /)`
Return self%value.

`__mul__(value, /)`
Return self*value.

`__ne__(value, /)`
Return self!=value.

`__neg__()`
-self

`__new__(value)`

`__or__(value, /)`
Return self|value.

`__pos__()`
+self

`__pow__(value, mod=None, /)`
Return pow(self, value, mod).

`__radd__(value, /)`
Return value+self.

`__rand__(value, /)`
Return value&self.

`__rdivmod__(value, /)`
Return divmod(value, self).

`__repr__()`
Return repr(self).

`__rfloordiv__(value, /)`
Return value//self.

`__rlshift__(value, /)`
Return value<<self.

`__rmod__(value, /)`
Return value%self.

`__rmul__(value, /)`
Return value*self.

`__ror__(value, /)`
Return value|self.

`__round__()`
Rounding an Integral returns itself.
Rounding with an ndigits argument also returns an integer.

`__rpow__(value, mod=None, /)`
Return pow(value, self, mod).

`__rrshift__(value, /)`
Return value>>self.

`__rshift__(value, /)`
Return self>>value.

`__rsub__(value, /)`
Return value-self.

`__rtruediv__(value, /)`
Return value/self.

`__rxor__(value, /)`
Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an `OverflowError` is raised.

class telegram.constants.**InputMediaType**(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `str, enum.Enum`

This enum contains the available types of `telegram.InputMedia`. The enum members of this enumeration are instances of `str` and can be treated as such.

New in version 20.0.

ANIMATION = 'animation'

Type of `telegram.InputMediaAnimation`.

Type

`str`

AUDIO = 'audio'

Type of `telegram.InputMediaAudio`.

Type

`str`

DOCUMENT = 'document'

Type of `telegram.InputMediaDocument`.

Type

`str`

PHOTO = 'photo'

Type of *telegram.InputMediaPhoto*.

Type

str

VIDEO = 'video'

Type of *telegram.InputMediaVideo*.

Type

str

__add__(*value*, /)

Return self+value.

__contains__(*key*, /)

Return bool(key in self).

__eq__(*value*, /)

Return self==value.

__format__(*format_spec*)

Return a formatted version of the string as described by *format_spec*.

__ge__(*value*, /)

Return self>=value.

__getattr__(*name*, /)

Return getattr(self, name).

__getitem__(*key*, /)

Return self[key].

__gt__(*value*, /)

Return self>value.

__hash__()

Return hash(self).

__iter__()

Implement iter(self).

__le__(*value*, /)

Return self<=value.

__len__()

Return len(self).

__lt__(*value*, /)

Return self<value.

__mod__(*value*, /)

Return self%value.

__mul__(*value*, /)

Return self*value.

__ne__(*value*, /)

Return self!=value.

__new__(*value*)

__repr__()

Return repr(self).

__rmod__(*value*, /)

Return *value*%*self*.

__rmul__(*value*, /)

Return *value***self*.

__sizeof__()

Return the size of the string in memory, in bytes.

__str__()

Return *str*(*self*).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(*width*, *fillchar*=' ', /)

Return a centered string of length *width*.

Padding is done using the specified fill character (default is a space).

count(*sub*[, *start*[, *end*]]) → *int*

Return the number of non-overlapping occurrences of substring *sub* in string *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

encode(*encoding*='utf-8', *errors*='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a `UnicodeEncodeError`. Other possible values are 'ignore', 'replace' and 'xmlchar-refreplace' as well as any other name registered with `codecs.register_error` that can handle `UnicodeEncodeErrors`.

endswith(*suffix*[, *start*[, *end*]]) → *bool*

Return `True` if *S* ends with the specified *suffix*, `False` otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *suffix* can also be a tuple of strings to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → *int*

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → *str*

Return a formatted version of *S*, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → *str*

Return a formatted version of *S*, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call `keyword.iskeyword(s)` to test whether string *s* is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in `repr()` or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(prefix, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(suffix, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(old, new, count=-1, /)

Return a copy with all occurrences of substring old replaced by new.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(*width*, *fillchar*=' ', /)

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars*=None, /)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

split(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if *S* starts with the specified prefix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *prefix* can also be a tuple of strings to try.

strip(*chars=None, /*)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table, /*)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(*width, /*)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.**InvoiceLimit**(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

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.

New 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()`.

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_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`**`__abs__()`**`abs(self)`**`__add__(value, /)`**`Return self+value.`**`__and__(value, /)`**`Return self&value.`**`__bool__()`**`True if self else False`**`__ceil__()`**`Ceiling of an Integral returns itself.`**`__divmod__(value, /)`**`Return divmod(self, value).`**`__eq__(value, /)`**`Return self==value.`**`__float__()`**`float(self)`**`__floor__()`**`Flooring an Integral returns itself.`**`__floordiv__(value, /)`**`Return self//value.`**`__format__(format_spec, /)`**`Convert to a string according to format_spec.`**`__ge__(value, /)`**`Return self>=value.`**`__getattr__(name, /)`**`Return getattr(self, name).`**`__gt__(value, /)`**`Return self>value.`**`__hash__()`**`Return hash(self).`**`__index__()`**`Return self converted to an integer, if self is suitable for use as an index into a list.`**`__int__()`**`int(self)`**`__invert__()`**`~self`**`__le__(value, /)`**`Return self<=value.`

__lshift__(value, /)
Return self<<value.

__lt__(value, /)
Return self<value.

__mod__(value, /)
Return self%value.

__mul__(value, /)
Return self*value.

__ne__(value, /)
Return self!=value.

__neg__()
-self

__new__(value)

__or__(value, /)
Return self|value.

__pos__()
+self

__pow__(value, mod=None, /)
Return pow(self, value, mod).

__radd__(value, /)
Return value+self.

__rand__(value, /)
Return value&self.

__rdivmod__(value, /)
Return divmod(value, self).

__repr__()
Return repr(self).

__rfloordiv__(value, /)
Return value//self.

__rlshift__(value, /)
Return value<<self.

__rmod__(value, /)
Return value%self.

__rmul__(value, /)
Return value*self.

__ror__(value, /)
Return value|self.

__round__()
Rounding an Integral returns itself.
Rounding with an ndigits argument also returns an integer.

__rpow__(value, mod=None, /)
Return pow(value, self, mod).

__rrshift__(value, /)

Return value>>self.

__rshift__(value, /)

Return self>>value.

__rsub__(value, /)

Return value-self.

__rtruediv__(value, /)

Return value/self.

__rxor__(value, /)

Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an `OverflowError` is raised.

```
class telegram.constants.KeyboardButtonRequestUsersLimit(value, names=None, *values,  
                                                         module=None, qualname=None,  
                                                         type=None, start=1,  
                                                         boundary=None)
```

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.

New 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`

__abs__()

`abs(self)`

__add__(value, /)

Return self+value.

__and__(value, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

__divmod__(value, /)

Return divmod(self, value).

__eq__(value, /)

Return self==value.

__float__()

`float(self)`

__floor__()

Flooring an Integral returns itself.

__floordiv__(value, /)

Return self//value.

__format__(format_spec, /)

Convert to a string according to format_spec.

__ge__(value, /)

Return self>=value.

__getattr__(name, /)

Return getattr(self, name).

__gt__(value, /)

Return self>value.

__hash__()

Return hash(self).

__index__()

Return self converted to an integer, if self is suitable for use as an index into a list.

`__int__()`
int(self)

`__invert__()`
~self

`__le__(value, /)`
Return self<=value.

`__lshift__(value, /)`
Return self<<value.

`__lt__(value, /)`
Return self<value.

`__mod__(value, /)`
Return self%value.

`__mul__(value, /)`
Return self*value.

`__ne__(value, /)`
Return self!=value.

`__neg__()`
-self

`__new__(value)`

`__or__(value, /)`
Return self|value.

`__pos__()`
+self

`__pow__(value, mod=None, /)`
Return pow(self, value, mod).

`__radd__(value, /)`
Return value+self.

`__rand__(value, /)`
Return value&self.

`__rdivmod__(value, /)`
Return divmod(value, self).

`__repr__()`
Return repr(self).

`__rfloordiv__(value, /)`
Return value//self.

`__rlshift__(value, /)`
Return value<<self.

`__rmod__(value, /)`
Return value%self.

`__rmul__(value, /)`
Return value*self.

__ror__(value, /)

Return value|self.

__round__()

Rounding an Integral returns itself.

Rounding with an ndigits argument also returns an integer.

__rpow__(value, mod=None, /)

Return pow(value, self, mod).

__rrshift__(value, /)

Return value>>self.

__rshift__(value, /)

Return self>>value.

__rsub__(value, /)

Return value-self.

__rtruediv__(value, /)

Return value/self.

__rxor__(value, /)

Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

```
class telegram.constants.LocationLimit(value, names=None, *values, module=None,
                                     qualname=None, type=None, start=1, boundary=None)
```

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.

New 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()`

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`

__abs__()

`abs(self)`

__add__(*value*, /)
Return self+value.

__and__(*value*, /)
Return self&value.

__bool__()
True if self else False

__ceil__()
Ceiling of an Integral returns itself.

__divmod__(*value*, /)
Return divmod(self, value).

__eq__(*value*, /)
Return self==value.

__float__()
float(self)

__floor__()
Flooring an Integral returns itself.

__floordiv__(*value*, /)
Return self//value.

__format__(*format_spec*, /)
Convert to a string according to format_spec.

__ge__(*value*, /)
Return self>=value.

__getattr__(*name*, /)
Return getattr(self, name).

__gt__(*value*, /)
Return self>value.

__hash__()
Return hash(self).

__index__()
Return self converted to an integer, if self is suitable for use as an index into a list.

__int__()
int(self)

__invert__()
~self

__le__(*value*, /)
Return self<=value.

__lshift__(*value*, /)
Return self<<value.

__lt__(*value*, /)
Return self<value.

__mod__(*value*, /)
Return self%value.

`__mul__(value, /)`
Return self*value.

`__ne__(value, /)`
Return self!=value.

`__neg__()`
-self

`__new__(value)`

`__or__(value, /)`
Return self|value.

`__pos__()`
+self

`__pow__(value, mod=None, /)`
Return pow(self, value, mod).

`__radd__(value, /)`
Return value+self.

`__rand__(value, /)`
Return value&self.

`__rdivmod__(value, /)`
Return divmod(value, self).

`__repr__()`
Return repr(self).

`__rfloordiv__(value, /)`
Return value//self.

`__rlshift__(value, /)`
Return value<<self.

`__rmod__(value, /)`
Return value%self.

`__rmul__(value, /)`
Return value*self.

`__ror__(value, /)`
Return value|self.

`__round__()`
Rounding an Integral returns itself.
Rounding with an ndigits argument also returns an integer.

`__rpow__(value, mod=None, /)`
Return pow(value, self, mod).

`__rrshift__(value, /)`
Return value>>self.

`__rshift__(value, /)`
Return self>>value.

`__rsub__(value, /)`
Return value-self.

__rtruediv__(value, /)

Return value/self.

__rxor__(value, /)

Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If *byteorder* is 'big', the most significant byte is at the beginning of the byte array. If *byteorder* is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If *byteorder* is 'big', the most significant byte is at the beginning of the byte array. If *byteorder* is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If *signed* is False and a negative integer is given, an `OverflowError` is raised.

class telegram.constants.MaskPosition(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `str`, `enum.Enum`

This enum contains the available positions for `telegram.MaskPosition`. The enum members of this enumeration are instances of `str` and can be treated as such.

New in version 20.0.

CHIN = 'chin'

Mask position for a sticker on the chin.

Type

`str`

EYES = 'eyes'

Mask position for a sticker on the eyes.

Type

`str`

FOREHEAD = 'forehead'

Mask position for a sticker on the forehead.

Type

`str`

MOUTH = 'mouth'

Mask position for a sticker on the mouth.

Type

`str`

__add__(value, /)

Return self+value.

__contains__(key, /)

Return bool(key in self).

__eq__(value, /)

Return self==value.

__format__(format_spec)

Return a formatted version of the string as described by format_spec.

__ge__(value, /)

Return self>=value.

__getattr__(name, /)

Return getattr(self, name).

__getitem__(key, /)

Return self[key].

__gt__(value, /)

Return self>value.

__hash__()

Return hash(self).

__iter__()

Implement iter(self).

__le__(value, /)

Return self<=value.

__len__()

Return len(self).

__lt__(value, /)

Return self<value.

__mod__(value, /)

Return self%value.

__mul__(value, /)

Return self*value.

__ne__(value, /)

Return self!=value.

__new__(value)

__repr__()

Return repr(self).

__rmod__(value, /)

Return value%self.

__rmul__(value, /)

Return value*self.

__sizeof__()

Return the size of the string in memory, in bytes.

__str__()

Return str(self).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(width, fillchar=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(sub[, start[, end]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

encode(encoding='utf-8', errors='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlchar-refreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith(suffix[, start[, end]]) → bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

expandtabs(tabsize=8)

Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

find(sub[, start[, end]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

format(*args, **kwargs) → str

Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → *str*

Return a formatted version of S, using substitutions from mapping. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start* [, *end*]]) → *int*

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier, such as "def" or "class".

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs'])` -> `'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(prefix, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(suffix, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(old, new, count=-1, /)

Return a copy with all occurrences of substring old replaced by new.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind(sub[, start[, end]]) → int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

rindex(sub[, start[, end]]) → int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

rjust(width, fillchar=' ', /)

Return a right-justified string of length width.

Padding is done using the specified fill character (default is a space).

rpartition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(chars=None, /)

Return a copy of the string with trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

split(sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, str.split() is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if *S* starts with the specified prefix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *prefix* can also be a tuple of strings to try.

strip(*chars=None, /*)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table, /*)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(*width, /*)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.**MediaGroupLimit**(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

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.

New 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`

`__abs__()`
abs(self)

`__add__(value, /)`
Return self+value.

`__and__(value, /)`
Return self&value.

`__bool__()`
True if self else False

`__ceil__()`
Ceiling of an Integral returns itself.

`__divmod__(value, /)`
Return divmod(self, value).

`__eq__(value, /)`
Return self==value.

`__float__()`
float(self)

`__floor__()`
Flooring an Integral returns itself.

`__floordiv__(value, /)`
Return self//value.

`__format__(format_spec, /)`
Convert to a string according to format_spec.

`__ge__(value, /)`
Return self>=value.

`__getattr__(name, /)`
Return getattr(self, name).

`__gt__(value, /)`
Return self>value.

`__hash__()`
Return hash(self).

`__index__()`
Return self converted to an integer, if self is suitable for use as an index into a list.

`__int__()`
int(self)

`__invert__()`
~self

`__le__(value, /)`
Return self<=value.

`__lshift__(value, /)`
Return self<<value.

`__lt__(value, /)`
Return self<value.

`__mod__(value, /)`
Return `self%value`.

`__mul__(value, /)`
Return `self*value`.

`__ne__(value, /)`
Return `self!=value`.

`__neg__()`
`-self`

`__new__(value)`

`__or__(value, /)`
Return `self|value`.

`__pos__()`
`+self`

`__pow__(value, mod=None, /)`
Return `pow(self, value, mod)`.

`__radd__(value, /)`
Return `value+self`.

`__rand__(value, /)`
Return `value&self`.

`__rdivmod__(value, /)`
Return `divmod(value, self)`.

`__repr__()`
Return `repr(self)`.

`__rfloordiv__(value, /)`
Return `value//self`.

`__rlshift__(value, /)`
Return `value<<self`.

`__rmod__(value, /)`
Return `value%self`.

`__rmul__(value, /)`
Return `value*self`.

`__ror__(value, /)`
Return `value|self`.

`__round__()`
Rounding an Integral returns itself.
Rounding with an `ndigits` argument also returns an integer.

`__rpow__(value, mod=None, /)`
Return `pow(value, self, mod)`.

`__rrshift__(value, /)`
Return `value>>self`.

`__rshift__(value, /)`
Return `self>>value`.

__rsub__(value, /)

Return value-self.

__rtruediv__(value, /)

Return value/self.

__rxor__(value, /)

Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(length=1, byteorder='big', *, signed=False)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an `OverflowError` is raised.

```
class telegram.constants.MenuButtonType(value, names=None, *values, module=None,
                                         qualname=None, type=None, start=1, boundary=None)
```

Bases: `str`, `enum.Enum`

This enum contains the available types of `telegram.MenuButton`. The enum members of this enumeration are instances of `str` and can be treated as such.

New in version 20.0.

```
COMMANDS = 'commands'
```

The type of `telegram.MenuButtonCommands`.

Type

`str`

DEFAULT = 'default'

The type of `telegram.MenuButtonDefault`.

Type

`str`

WEB_APP = 'web_app'

The type of `telegram.MenuButtonWebApp`.

Type

`str`

__add__(value, /)

Return self+value.

__contains__(key, /)

Return bool(key in self).

__eq__(value, /)

Return self==value.

__format__(format_spec)

Return a formatted version of the string as described by format_spec.

__ge__(value, /)

Return self>=value.

__getattr__(name, /)

Return getattr(self, name).

__getitem__(key, /)

Return self[key].

__gt__(value, /)

Return self>value.

__hash__()

Return hash(self).

__iter__()

Implement iter(self).

__le__(value, /)

Return self<=value.

__len__()

Return len(self).

__lt__(value, /)

Return self<value.

__mod__(value, /)

Return self%value.

__mul__(value, /)

Return self*value.

__ne__(value, /)

Return self!=value.

__new__(value)

__repr__()

Return repr(self).

__rmod__(*value*, /)

Return *value*%*self*.

__rmul__(*value*, /)

Return *value***self*.

__sizeof__()

Return the size of the string in memory, in bytes.

__str__()

Return *str*(*self*).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(*width*, *fillchar*=' ', /)

Return a centered string of length *width*.

Padding is done using the specified fill character (default is a space).

count(*sub*[, *start*[, *end*]]) → *int*

Return the number of non-overlapping occurrences of substring *sub* in string *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

encode(*encoding*='utf-8', *errors*='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a `UnicodeEncodeError`. Other possible values are 'ignore', 'replace' and 'xmlchar-refreplace' as well as any other name registered with `codecs.register_error` that can handle `UnicodeEncodeErrors`.

endswith(*suffix*[, *start*[, *end*]]) → *bool*

Return `True` if *S* ends with the specified *suffix*, `False` otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *suffix* can also be a tuple of strings to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → *int*

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → *str*

Return a formatted version of *S*, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → *str*

Return a formatted version of *S*, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → *int*

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

isalnum()

Return `True` if the string is an alpha-numeric string, `False` otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return `True` if the string is an alphabetic string, `False` otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return `True` if all characters in the string are ASCII, `False` otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return `True` if the string is a decimal string, `False` otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return `True` if the string is a digit string, `False` otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return `True` if the string is a valid Python identifier, `False` otherwise.

Call `keyword.iskeyword(s)` to test whether string *s* is a reserved identifier, such as “`def`” or “`class`”.

islower()

Return `True` if the string is a lowercase string, `False` otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return `True` if the string is a numeric string, `False` otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return `True` if the string is printable, `False` otherwise.

A string is printable if all of its characters are considered printable in `repr()` or if it is empty.

isspace()

Return `True` if the string is a whitespace string, `False` otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs'])` -> `'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(prefix, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(suffix, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(old, new, count=-1, /)

Return a copy with all occurrences of substring old replaced by new.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(*width*, *fillchar*=' ', /)

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars*=None, /)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

split(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if *S* starts with the specified prefix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *prefix* can also be a tuple of strings to try.

strip(*chars=None, /*)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table, /*)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(*width, /*)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.**MessageType**(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `str`, `enum.Enum`

This enum contains the available types of `telegram.Message` that can be seen as attachment. The enum members of this enumeration are instances of `str` and can be treated as such.

New in version 20.0.

ANIMATION = 'animation'

Messages with `telegram.Message.animation`.

Type

`str`

AUDIO = 'audio'

Messages with `telegram.Message.audio`.

Type

`str`

CONTACT = 'contact'

Messages with *telegram.Message.contact*.

Type

str

DICE = 'dice'

Messages with *telegram.Message.dice*.

Type

str

DOCUMENT = 'document'

Messages with *telegram.Message.document*.

Type

str

GAME = 'game'

Messages with *telegram.Message.game*.

Type

str

INVOICE = 'invoice'

Messages with *telegram.Message.invoice*.

Type

str

LOCATION = 'location'

Messages with *telegram.Message.location*.

Type

str

PASSPORT_DATA = 'passport_data'

Messages with *telegram.Message.passport_data*.

Type

str

PHOTO = 'photo'

Messages with *telegram.Message.photo*.

Type

str

POLL = 'poll'

Messages with *telegram.Message.poll*.

Type

str

STICKER = 'sticker'

Messages with *telegram.Message.sticker*.

Type

str

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`

__add__(value, /)

Return self+value.

__contains__(key, /)

Return bool(key in self).

__eq__(value, /)

Return self==value.

__format__(format_spec)

Return a formatted version of the string as described by format_spec.

__ge__(value, /)

Return self>=value.

__getattr__(name, /)

Return getattr(self, name).

__getitem__(key, /)

Return self[key].

__gt__(value, /)

Return self>value.

__hash__()

Return hash(self).

__iter__()

Implement iter(self).

__le__(value, /)

Return self<=value.

__len__()

Return len(self).

__lt__(value, /)

Return self<value.

__mod__(value, /)

Return self%value.

__mul__(value, /)

Return self*value.

__ne__(value, /)

Return self!=value.

__new__(value)

__repr__()

Return repr(self).

__rmod__(value, /)

Return value%self.

__rmul__(value, /)

Return value*self.

__sizeof__()

Return the size of the string in memory, in bytes.

__str__()

Return str(self).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(width, fillchar=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(sub[, start[, end]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

encode(encoding='utf-8', errors='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith(*suffix*[, *start*[, *end*]]) → bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. *suffix* can also be a tuple of strings to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int

Return the lowest index in S where substring *sub* is found, such that *sub* is contained within S[start:end]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → str

Return a formatted version of S, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → str

Return a formatted version of S, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in S where substring *sub* is found, such that *sub* is contained within S[start:end]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string *s* is a reserved identifier, such as "def" or "class".

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in `repr()` or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(*prefix*, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return string[len(prefix):]. Otherwise, return a copy of the original string.

removesuffix(*suffix*, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return string[:-len(suffix)]. Otherwise, return a copy of the original string.

replace(*old*, *new*, *count=-1*, /)

Return a copy with all occurrences of substring *old* replaced by *new*.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(*width*, *fillchar=' '*, /)

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(*sep=None*, *maxsplit=-1*)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to `None` (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(chars=None, /)

Return a copy of the string with trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

split(sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(keepends=False)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless `keepends` is given and true.

startswith(prefix[, start[, end]]) → bool

Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

strip(chars=None, /)

Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(table, /)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(width, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.MessageEntityType(value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None)

Bases: `str`, `enum.Enum`

This enum contains the available types of `telegram.MessageEntity`. The enum members of this enumeration are instances of `str` and can be treated as such.

New in version 20.0.

BLOCKQUOTE = 'blockquote'

Message entities representing a block quotation.

New 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.

New in version 20.0.

Type

`str`

EMAIL = 'email'

Message entities representing a email.

Type

`str`

HASHTAG = 'hashtag'

Message entities representing a hashtag.

Type

`str`

ITALIC = 'italic'

Message entities representing italic text.

Type

`str`

MENTION = 'mention'

Message entities representing a mention.

Type

`str`

PHONE_NUMBER = 'phone_number'

Message entities representing a phone number.

Type

`str`

PRE = 'pre'

Message entities representing monowidth block.

Type

`str`

SPOILER = 'spoiler'

Message entities representing spoiler text.

Type

`str`

STRIKETHROUGH = 'strikethrough'

Message entities representing strikethrough text.

Type

`str`

TEXT_LINK = 'text_link'

Message entities representing clickable text URLs.

Type

`str`

TEXT_MENTION = 'text_mention'

Message entities representing text mention for users without usernames.

Type

`str`

UNDERLINE = 'underline'

Message entities representing underline text.

Type

`str`

URL = 'url'

Message entities representing a url.

Type

`str`

__add__(value, /)

Return self+value.

__contains__(key, /)

Return bool(key in self).

__eq__(value, /)

Return self==value.

__format__(format_spec)

Return a formatted version of the string as described by format_spec.

__ge__(*value*, /)
Return self>=value.

__getattr__(*name*, /)
Return getattr(self, name).

__getitem__(*key*, /)
Return self[key].

__gt__(*value*, /)
Return self>value.

__hash__()
Return hash(self).

__iter__()
Implement iter(self).

__le__(*value*, /)
Return self<=value.

__len__()
Return len(self).

__lt__(*value*, /)
Return self<value.

__mod__(*value*, /)
Return self%value.

__mul__(*value*, /)
Return self*value.

__ne__(*value*, /)
Return self!=value.

__new__(*value*)

__repr__()
Return repr(self).

__rmod__(*value*, /)
Return value%self.

__rmul__(*value*, /)
Return value*self.

__sizeof__()
Return the size of the string in memory, in bytes.

__str__()
Return str(self).

capitalize()
Return a capitalized version of the string.
More specifically, make the first character have upper case and the rest lower case.

casefold()
Return a version of the string suitable for caseless comparisons.

center(width, fillchar=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(sub[, start[, end]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

encode(encoding='utf-8', errors='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlchar-refreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith(suffix[, start[, end]]) → bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

expandtabs(tabsize=8)

Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

find(sub[, start[, end]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

format(*args, **kwargs) → str

Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

format_map(mapping) → str

Return a formatted version of S, using substitutions from mapping. The substitutions are identified by braces ('{' and '}').

index(sub[, start[, end]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `‘.’.join([‘ab’, ‘pq’, ‘rs’]) -> ‘ab.pq.rs’`

ljust(*width*, *fillchar*=' ', /)

Return a left-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(*chars*=None, /)

Return a copy of the string with leading whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in *x* will be mapped to the character at the same position in *y*. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(*prefix*, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(*suffix*, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(*old*, *new*, *count*=-1, /)

Return a copy with all occurrences of substring *old* replaced by *new*.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(width, fillchar=' ', /)

Return a right-justified string of length width.

Padding is done using the specified fill character (default is a space).

rpartition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\r`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(chars=None, /)

Return a copy of the string with trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

split(sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\r`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(keepends=False)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless `keepends` is given and true.

startswith(prefix[, start[, end]]) → bool

Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

strip(chars=None, /)

Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppcased characters and all remaining cased characters have lower case.

translate(*table*, /)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(*width*, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.**MessageLimit**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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.

New 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()`

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 the `message_text` parameter of `telegram.InputTextMessageContent` and the `text` parameter of `telegram.Bot.edit_message_text()`.

Type`int`**__abs__()**

`abs(self)`

__add__(value, /)

Return self+value.

__and__(value, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

__divmod__(value, /)

Return divmod(self, value).

__eq__(value, /)

Return self==value.

__float__()

`float(self)`

__floor__()

Flooring an Integral returns itself.

__floordiv__(value, /)

Return self//value.

__format__(format_spec, /)

Convert to a string according to `format_spec`.

__ge__(value, /)

Return self>=value.

__getattr__(name, /)

Return `getattr(self, name)`.

`__gt__(value, /)`
Return self>value.

`__hash__()`
Return hash(self).

`__index__()`
Return self converted to an integer, if self is suitable for use as an index into a list.

`__int__()`
int(self)

`__invert__()`
~self

`__le__(value, /)`
Return self<=value.

`__lshift__(value, /)`
Return self<<value.

`__lt__(value, /)`
Return self<value.

`__mod__(value, /)`
Return self%value.

`__mul__(value, /)`
Return self*value.

`__ne__(value, /)`
Return self!=value.

`__neg__()`
-self

`__new__(value)`

`__or__(value, /)`
Return self|value.

`__pos__()`
+self

`__pow__(value, mod=None, /)`
Return pow(self, value, mod).

`__radd__(value, /)`
Return value+self.

`__rand__(value, /)`
Return value&self.

`__rdivmod__(value, /)`
Return divmod(value, self).

`__repr__()`
Return repr(self).

`__rfloordiv__(value, /)`
Return value//self.

`__rlshift__(value, /)`
Return `value<<self`.

`__rmod__(value, /)`
Return `value%self`.

`__rmul__(value, /)`
Return `value*self`.

`__ror__(value, /)`
Return `value|self`.

`__round__()`
Rounding an Integral returns itself.
Rounding with an `ndigits` argument also returns an integer.

`__rpow__(value, mod=None, /)`
Return `pow(value, self, mod)`.

`__rrshift__(value, /)`
Return `value>>self`.

`__rshift__(value, /)`
Return `self>>value`.

`__rsub__(value, /)`
Return `value-self`.

`__rtruediv__(value, /)`
Return `value/self`.

`__rxor__(value, /)`
Return `value^self`.

`__sizeof__()`
Returns size in memory, in bytes.

`__str__()`
Return `repr(self)`.

`__sub__(value, /)`
Return `self-value`.

`__truediv__(value, /)`
Return `self/value`.

`__trunc__()`
Truncating an Integral returns itself.

`__xor__(value, /)`
Return `self^value`.

`as_integer_ratio()`
Return a pair of integers, whose ratio is equal to the original int.
The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with float.is_integer.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An OverflowError is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of

the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use `'big'`.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an `OverflowError` is raised.

```
class telegram.constants.MessageOriginType(value, names=None, *values, module=None,
                                           qualname=None, type=None, start=1, boundary=None)
```

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.

New 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`

__add__(value, /)

Return self+value.

__contains__(key, /)

Return bool(key in self).

__eq__(value, /)

Return self==value.

__format__(format_spec)

Return a formatted version of the string as described by format_spec.

__ge__(value, /)

Return self>=value.

__getattr__(name, /)

Return getattr(self, name).

__getitem__(key, /)

Return self[key].

__gt__(value, /)

Return self>value.

__hash__()

Return hash(self).

__iter__()

Implement iter(self).

__le__(value, /)

Return self<=value.

__len__()

Return len(self).

__lt__(value, /)

Return self<value.

__mod__(value, /)

Return self%value.

__mul__(value, /)

Return self*value.

__ne__(value, /)

Return self!=value.

__new__(value)

__repr__()

Return repr(self).

__rmod__(value, /)

Return value%self.

__rmul__(value, /)

Return value*self.

__sizeof__()

Return the size of the string in memory, in bytes.

__str__()

Return str(self).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(width, fillchar=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(sub[, start[, end]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

encode(encoding='utf-8', errors='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a `UnicodeEncodeError`. Other possible values are 'ignore', 'replace' and 'xmlchar-refreplace' as well as any other name registered with `codecs.register_error` that can handle `UnicodeEncodeErrors`.

endswith(*suffix*[, *start*[, *end*]]) → bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. *suffix* can also be a tuple of strings to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int

Return the lowest index in S where substring *sub* is found, such that *sub* is contained within S[start:end]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → str

Return a formatted version of S, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → str

Return a formatted version of S, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in S where substring *sub* is found, such that *sub* is contained within S[start:end]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call `keyword.iskeyword(s)` to test whether string `s` is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in `repr()` or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `‘.’.join([‘ab’, ‘pq’, ‘rs’]) -> ‘ab.pq.rs’`

ljust(width, fillchar=‘ ’, /)

Return a left-justified string of length `width`.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If `chars` is given and not None, remove characters in `chars` instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in

x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(prefix, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return string[len(prefix):]. Otherwise, return a copy of the original string.

removesuffix(suffix, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return string[:-len(suffix)]. Otherwise, return a copy of the original string.

replace(old, new, count=-1, /)

Return a copy with all occurrences of substring old replaced by new.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind(sub[, start[, end]]) → int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

rindex(sub[, start[, end]]) → int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

rjust(width, fillchar=' ', /)

Return a right-justified string of length width.

Padding is done using the specified fill character (default is a space).

rpartition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars=None, /*)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not *None*, remove characters in *chars* instead.

split(*sep=None, maxsplit=-1*)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to *None* (the default value), will split on any whitespace character (including `n`, `r`, `t`, `f` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix[, start[, end]]*) → *bool*

Return True if *S* starts with the specified prefix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *prefix* can also be a tuple of strings to try.

strip(*chars=None, /*)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not *None*, remove characters in *chars* instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table, /*)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or *None*.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to *None* are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(*width, /*)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

```
class telegram.constants.MessageType(value, names=None, *values, module=None, qualname=None,
                                     type=None, start=1, boundary=None)
```

Bases: `str`, `enum.Enum`

This enum contains the available types of `telegram.Message`. The enum members of this enumeration are instances of `str` and can be treated as such.

New in version 20.0.

ANIMATION = 'animation'

Messages with `telegram.Message.animation`.

Type
`str`

AUDIO = 'audio'

Messages with `telegram.Message.audio`.

Type
`str`

CHANNEL_CHAT_CREATED = 'channel_chat_created'

Messages with `telegram.Message.channel_chat_created`.

Type
`str`

CHAT_SHARED = 'chat_shared'

Messages with `telegram.Message.chat_shared`.

New in version 20.8.

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`

DOCUMENT = 'document'

Messages with `telegram.Message.document`.

Type
`str`

FORUM_TOPIC_CLOSED = 'forum_topic_closed'

Messages with *telegram.Message.forum_topic_closed*.

New in version 20.8.

Type

str

FORUM_TOPIC_CREATED = 'forum_topic_created'

Messages with *telegram.Message.forum_topic_created*.

New in version 20.8.

Type

str

FORUM_TOPIC_EDITED = 'forum_topic_edited'

Messages with *telegram.Message.forum_topic_edited*.

New in version 20.8.

Type

str

FORUM_TOPIC_REOPENED = 'forum_topic_reopened'

Messages with *telegram.Message.forum_topic_reopened*.

New 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*.

New in version 20.8.

Type

str

GENERAL_FORUM_TOPIC_UNHIDDEN = 'general_forum_topic_unhidden'

Messages with *telegram.Message.general_forum_topic_unhidden*.

New in version 20.8.

Type

str

GIVEAWAY = 'giveaway'

Messages with *telegram.Message.giveaway*.

New in version 20.8.

Type

str

GIVEAWAY_COMPLETED = 'giveaway_completed'

Messages with *telegram.Message.giveaway_completed*.

New in version 20.8.

Type

str

GIVEAWAY_CREATED = 'giveaway_created'

Messages with `telegram.Message.giveaway_created`.

New in version 20.8.

Type

`str`

GIVEAWAY_WINNERS = 'giveaway_winners'

Messages with `telegram.Message.giveaway_winners`.

New 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`

LOCATION = 'location'

Messages with `telegram.Message.location`.

Type

`str`

MESSAGE_AUTO_DELETE_TIMER_CHANGED = 'message_auto_delete_timer_changed'

Messages with `telegram.Message.message_auto_delete_timer_changed`.

Type

`str`

MIGRATE_TO_CHAT_ID = 'migrate_to_chat_id'

Messages with `telegram.Message.migrate_to_chat_id`.

Type

`str`

NEW_CHAT_MEMBERS = 'new_chat_members'

Messages with `telegram.Message.new_chat_members`.

Type

`str`

NEW_CHAT_PHOTO = 'new_chat_photo'

Messages with `telegram.Message.new_chat_photo`.

Type

`str`

NEW_CHAT_TITLE = 'new_chat_title'

Messages with `telegram.Message.new_chat_title`.

Type

`str`

PASSPORT_DATA = 'passport_data'

Messages with `telegram.Message.passport_data`.

Type

`str`

PHOTO = 'photo'

Messages with `telegram.Message.photo`.

Type

`str`

PINNED_MESSAGE = 'pinned_message'

Messages with `telegram.Message.pinned_message`.

Type

`str`

POLL = 'poll'

Messages with `telegram.Message.poll`.

Type

`str`

PROXIMITY_ALERT_TRIGGERED = 'proximity_alert_triggered'

Messages with `telegram.Message.proximity_alert_triggered`.

Type

`str`

STICKER = 'sticker'

Messages with `telegram.Message.sticker`.

Type

`str`

STORY = 'story'

Messages with `telegram.Message.story`.

Type

`str`

SUCCESSFUL_PAYMENT = 'successful_payment'

Messages with `telegram.Message.successful_payment`.

Type

`str`

SUPERGROUP_CHAT_CREATED = 'supergroup_chat_created'

Messages with `telegram.Message.supergroup_chat_created`.

Type

`str`

TEXT = 'text'

Messages with `telegram.Message.text`.

Type

`str`

USERS_SHARED = 'users_shared'

Messages with *telegram.Message.users_shared*.

New 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*.

New in version 20.8.

Type

str

WRITE_ACCESS_ALLOWED = 'write_access_allowed'

Messages with `telegram.Message.write_access_allowed`.

New in version 20.8.

Type

`str`

__add__(*value*, /)

Return self+value.

__contains__(*key*, /)

Return bool(key in self).

__eq__(*value*, /)

Return self==value.

__format__(*format_spec*)

Return a formatted version of the string as described by *format_spec*.

__ge__(*value*, /)

Return self>=value.

__getattr__(*name*, /)

Return getattr(self, name).

__getitem__(*key*, /)

Return self[key].

__gt__(*value*, /)

Return self>value.

__hash__()

Return hash(self).

__iter__()

Implement iter(self).

__le__(*value*, /)

Return self<=value.

__len__()

Return len(self).

__lt__(*value*, /)

Return self<value.

__mod__(*value*, /)

Return self%value.

__mul__(*value*, /)

Return self*value.

__ne__(*value*, /)

Return self!=value.

__new__(*value*)

__repr__()

Return repr(self).

__rmod__(*value*, /)

Return value%self.

__rmul__(value, /)

Return value*self.

__sizeof__()

Return the size of the string in memory, in bytes.

__str__()

Return str(self).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(width, fillchar=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(sub[, start[, end]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

encode(encoding='utf-8', errors='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlchar-refreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith(suffix[, start[, end]]) → bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

expandtabs(tabsize=8)

Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

find(sub[, start[, end]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

format(*args, **kwargs) → str

Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

format_map(mapping) → str

Return a formatted version of S, using substitutions from mapping. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

isalnum()

Return `True` if the string is an alpha-numeric string, `False` otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return `True` if the string is an alphabetic string, `False` otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return `True` if all characters in the string are ASCII, `False` otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return `True` if the string is a decimal string, `False` otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return `True` if the string is a digit string, `False` otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return `True` if the string is a valid Python identifier, `False` otherwise.

Call `keyword.iskeyword(s)` to test whether string *s* is a reserved identifier, such as “`def`” or “`class`”.

islower()

Return `True` if the string is a lowercase string, `False` otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return `True` if the string is a numeric string, `False` otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return `True` if the string is printable, `False` otherwise.

A string is printable if all of its characters are considered printable in `repr()` or if it is empty.

isspace()

Return `True` if the string is a whitespace string, `False` otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs'])` -> `'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(prefix, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(suffix, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(old, new, count=-1, /)

Return a copy with all occurrences of substring old replaced by new.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → *int*

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → *int*

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises *ValueError* when the substring is not found.

rjust(*width*, *fillchar*=' ', /)

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including *n*, *r*, *t*, *f* and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars*=None, /)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

split(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including *n*, *r*, *t*, *f* and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, *str.split()* is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. *prefix* can also be a tuple of strings to try.

strip(*chars=None, /*)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table, /*)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(*width, /*)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.ParseMode(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `str`, `enum.Enum`

This enum contains the available parse modes. The enum members of this enumeration are instances of `str` and can be treated as such.

New in version 20.0.

HTML = 'HTML'

HTML parse mode.

Type

`str`

MARKDOWN = 'Markdown'

Markdown parse mode.

Note: `MARKDOWN` is a legacy mode, retained by Telegram for backward compatibility. You should use `MARKDOWN_V2` instead.

Type`str`**MARKDOWN_V2** = 'MarkdownV2'

Markdown parse mode version 2.

Type`str`**__add__**(*value*, /)

Return self+value.

__contains__(*key*, /)

Return bool(key in self).

__eq__(*value*, /)

Return self==value.

__format__(*format_spec*)Return a formatted version of the string as described by *format_spec*.**__ge__**(*value*, /)

Return self>=value.

__getattr__(*name*, /)Return getattr(self, *name*).**__getitem__**(*key*, /)

Return self[key].

__gt__(*value*, /)

Return self>value.

__hash__()

Return hash(self).

__iter__()

Implement iter(self).

__le__(*value*, /)

Return self<=value.

__len__()

Return len(self).

__lt__(*value*, /)

Return self<value.

__mod__(*value*, /)

Return self%value.

__mul__(*value*, /)

Return self*value.

__ne__(*value*, /)

Return self!=value.

__new__(*value*)**__repr__**()

Return repr(self).

__rmod__(*value*, /)

Return *value*%*self*.

__rmul__(*value*, /)

Return *value***self*.

__sizeof__()

Return the size of the string in memory, in bytes.

__str__()

Return *str*(*self*).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(*width*, *fillchar*=' ', /)

Return a centered string of length *width*.

Padding is done using the specified fill character (default is a space).

count(*sub*[, *start*[, *end*]]) → *int*

Return the number of non-overlapping occurrences of substring *sub* in string *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

encode(*encoding*='utf-8', *errors*='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a `UnicodeEncodeError`. Other possible values are 'ignore', 'replace' and 'xmlchar-refreplace' as well as any other name registered with `codecs.register_error` that can handle `UnicodeEncodeErrors`.

endswith(*suffix*[, *start*[, *end*]]) → *bool*

Return True if *S* ends with the specified *suffix*, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *suffix* can also be a tuple of strings to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → *int*

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → *str*

Return a formatted version of *S*, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → *str*

Return a formatted version of *S*, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

isalnum()

Return `True` if the string is an alpha-numeric string, `False` otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return `True` if the string is an alphabetic string, `False` otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return `True` if all characters in the string are ASCII, `False` otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return `True` if the string is a decimal string, `False` otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return `True` if the string is a digit string, `False` otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return `True` if the string is a valid Python identifier, `False` otherwise.

Call `keyword.iskeyword(s)` to test whether string *s* is a reserved identifier, such as “`def`” or “`class`”.

islower()

Return `True` if the string is a lowercase string, `False` otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return `True` if the string is a numeric string, `False` otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return `True` if the string is printable, `False` otherwise.

A string is printable if all of its characters are considered printable in `repr()` or if it is empty.

isspace()

Return `True` if the string is a whitespace string, `False` otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs'])` -> `'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(prefix, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(suffix, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(old, new, count=-1, /)

Return a copy with all occurrences of substring old replaced by new.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind(sub[, start[, end]]) → int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

rindex(sub[, start[, end]]) → int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

rjust(width, fillchar=' ', /)

Return a right-justified string of length width.

Padding is done using the specified fill character (default is a space).

rpartition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(chars=None, /)

Return a copy of the string with trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

split(sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, str.split() is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if *S* starts with the specified prefix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *prefix* can also be a tuple of strings to try.

strip(*chars=None, /*)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table, /*)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(*width, /*)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.PollLimit(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

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.

New in version 20.0.

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 = 600

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()*.

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 = 10

Maximum number of strings passed in a *list* to the *options* parameter of *telegram.Bot.send_poll()*.

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_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 = 2

Minimum number of strings passed in a *list* to the *options* parameter of *telegram.Bot.send_poll()*.

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

__abs__()

abs(self)

__add__(value, /)

Return self+value.

__and__(value, /)

Return self&value.

`__bool__()`
True if self else False

`__ceil__()`
Ceiling of an Integral returns itself.

`__divmod__(value, /)`
Return divmod(self, value).

`__eq__(value, /)`
Return self==value.

`__float__()`
float(self)

`__floor__()`
Flooring an Integral returns itself.

`__floordiv__(value, /)`
Return self//value.

`__format__(format_spec, /)`
Convert to a string according to format_spec.

`__ge__(value, /)`
Return self>=value.

`__getattr__(name, /)`
Return getattr(self, name).

`__gt__(value, /)`
Return self>value.

`__hash__()`
Return hash(self).

`__index__()`
Return self converted to an integer, if self is suitable for use as an index into a list.

`__int__()`
int(self)

`__invert__()`
~self

`__le__(value, /)`
Return self<=value.

`__lshift__(value, /)`
Return self<<value.

`__lt__(value, /)`
Return self<value.

`__mod__(value, /)`
Return self%value.

`__mul__(value, /)`
Return self*value.

`__ne__(value, /)`
Return self!=value.

`__neg__()`
-self

`__new__(value)`

`__or__(value, /)`
Return self|value.

`__pos__()`
+self

`__pow__(value, mod=None, /)`
Return pow(self, value, mod).

`__radd__(value, /)`
Return value+self.

`__rand__(value, /)`
Return value&self.

`__rdivmod__(value, /)`
Return divmod(value, self).

`__repr__()`
Return repr(self).

`__rfloordiv__(value, /)`
Return value//self.

`__rlshift__(value, /)`
Return value<<self.

`__rmod__(value, /)`
Return value%self.

`__rmul__(value, /)`
Return value*self.

`__ror__(value, /)`
Return value|self.

`__round__()`
Rounding an Integral returns itself.
Rounding with an ndigits argument also returns an integer.

`__rpow__(value, mod=None, /)`
Return pow(value, self, mod).

`__rrshift__(value, /)`
Return value>>self.

`__rshift__(value, /)`
Return self>>value.

`__rsub__(value, /)`
Return value-self.

`__rtruediv__(value, /)`
Return value/self.

`__rxor__(value, /)`
Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an `OverflowError` is raised.

class telegram.constants.PollType(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `str`, `enum.Enum`

This enum contains the available types for `telegram.Poll/ telegram.Bot.send_poll()`. The enum members of this enumeration are instances of `str` and can be treated as such.

New in version 20.0.

QUIZ = 'quiz'

quiz polls.

Type

`str`

REGULAR = 'regular'

regular polls.

Type

`str`

__add__(*value, /*)

Return self+value.

__contains__(*key, /*)

Return bool(key in self).

`__eq__`(*value*, /)
Return self==value.

`__format__`(*format_spec*)
Return a formatted version of the string as described by *format_spec*.

`__ge__`(*value*, /)
Return self>=value.

`__getattr__`(*name*, /)
Return getattr(self, name).

`__getitem__`(*key*, /)
Return self[key].

`__gt__`(*value*, /)
Return self>value.

`__hash__`()
Return hash(self).

`__iter__`()
Implement iter(self).

`__le__`(*value*, /)
Return self<=value.

`__len__`()
Return len(self).

`__lt__`(*value*, /)
Return self<value.

`__mod__`(*value*, /)
Return self%value.

`__mul__`(*value*, /)
Return self*value.

`__ne__`(*value*, /)
Return self!=value.

`__new__`(*value*)

`__repr__`()
Return repr(self).

`__rmod__`(*value*, /)
Return value%self.

`__rmul__`(*value*, /)
Return value*self.

`__sizeof__`()
Return the size of the string in memory, in bytes.

`__str__`()
Return str(self).

`capitalize`()
Return a capitalized version of the string.
More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(*width*, *fillchar*=' ', /)

Return a centered string of length *width*.

Padding is done using the specified fill character (default is a space).

count(*sub*[, *start*[, *end*]]) → *int*

Return the number of non-overlapping occurrences of substring *sub* in string *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

encode(*encoding*='utf-8', *errors*='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a `UnicodeEncodeError`. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with `codecs.register_error` that can handle `UnicodeEncodeErrors`.

endswith(*suffix*[, *start*[, *end*]]) → *bool*

Return True if *S* ends with the specified *suffix*, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *suffix* can also be a tuple of strings to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → *int*

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → *str*

Return a formatted version of *S*, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → *str*

Return a formatted version of *S*, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → *int*

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `‘.’.join([‘ab’, ‘pq’, ‘rs’]) -> ‘ab.pq.rs’`

ljust(*width*, *fillchar*=' ', /)

Return a left-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(*chars*=None, /)

Return a copy of the string with leading whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in *x* will be mapped to the character at the same position in *y*. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(*prefix*, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(*suffix*, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(*old*, *new*, *count*=-1, /)

Return a copy with all occurrences of substring *old* replaced by *new*.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(width, fillchar=' ', /)

Return a right-justified string of length width.

Padding is done using the specified fill character (default is a space).

rpartition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(chars=None, /)

Return a copy of the string with trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

split(sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, str.split() is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(keepends=False)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless keepends is given and true.

startswith(prefix[, start[, end]]) → bool

Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

strip(chars=None, /)

Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppcased characters and all remaining cased characters have lower case.

translate(*table*, /)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(*width*, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.**PollingLimit**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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.

New 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`

__abs__()

`abs(self)`

__add__(*value*, /)

Return self+value.

__and__(*value*, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

__divmod__(*value*, /)

Return divmod(self, value).

__eq__(*value*, /)

Return self==value.

__float__()
float(self)

__floor__()
Flooring an Integral returns itself.

__floordiv__(value, /)
Return self//value.

__format__(format_spec, /)
Convert to a string according to format_spec.

__ge__(value, /)
Return self>=value.

__getattr__(name, /)
Return getattr(self, name).

__gt__(value, /)
Return self>value.

__hash__()
Return hash(self).

__index__()
Return self converted to an integer, if self is suitable for use as an index into a list.

__int__()
int(self)

__invert__()
~self

__le__(value, /)
Return self<=value.

__lshift__(value, /)
Return self<<value.

__lt__(value, /)
Return self<value.

__mod__(value, /)
Return self%value.

__mul__(value, /)
Return self*value.

__ne__(value, /)
Return self!=value.

__neg__()
-self

__new__(value)

__or__(value, /)
Return self|value.

__pos__()
+self

__pow__(*value*, *mod*=None, /)
Return pow(self, value, mod).

__radd__(*value*, /)
Return value+self.

__rand__(*value*, /)
Return value&self.

__rdivmod__(*value*, /)
Return divmod(value, self).

__repr__()
Return repr(self).

__rfloordiv__(*value*, /)
Return value//self.

__rlshift__(*value*, /)
Return value<<self.

__rmod__(*value*, /)
Return value%self.

__rmul__(*value*, /)
Return value*self.

__ror__(*value*, /)
Return value|self.

__round__()
Rounding an Integral returns itself.
Rounding with an ndigits argument also returns an integer.

__rpow__(*value*, *mod*=None, /)
Return pow(value, self, mod).

__rrshift__(*value*, /)
Return value>>self.

__rshift__(*value*, /)
Return self>>value.

__rsub__(*value*, /)
Return value-self.

__rtruediv__(*value*, /)
Return value/self.

__rxor__(*value*, /)
Return value^self.

__sizeof__()
Returns size in memory, in bytes.

__str__()
Return repr(self).

__sub__(*value*, /)
Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with float.is_integer.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(length=1, byteorder='big', *, signed=False)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An OverflowError is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

```
class telegram.constants.ProfileAccentColor(value, names=None, *values, module=None,
                                             qualname=None, type=None, start=1,
                                             boundary=None)
```

Bases: [Enum](#)

This enum contains the available accent colors for `telegram.Chat.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.

New 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(value, names=None, *values, module=None,  
                                       qualname=None, type=None, start=1, boundary=None)
```

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.

New 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`

__add__(*value*, /)

Return self+value.

__contains__(*key*, /)

Return bool(key in self).

__eq__(*value*, /)

Return self==value.

__format__(*format_spec*)

Return a formatted version of the string as described by *format_spec*.

`__ge__(value, /)`
Return self>=value.

`__getattr__(name, /)`
Return getattr(self, name).

`__getitem__(key, /)`
Return self[key].

`__gt__(value, /)`
Return self>value.

`__hash__()`
Return hash(self).

`__iter__()`
Implement iter(self).

`__le__(value, /)`
Return self<=value.

`__len__()`
Return len(self).

`__lt__(value, /)`
Return self<value.

`__mod__(value, /)`
Return self%value.

`__mul__(value, /)`
Return self*value.

`__ne__(value, /)`
Return self!=value.

`__new__(value)`

`__repr__()`
Return repr(self).

`__rmod__(value, /)`
Return value%self.

`__rmul__(value, /)`
Return value*self.

`__sizeof__()`
Return the size of the string in memory, in bytes.

`__str__()`
Return str(self).

`capitalize()`
Return a capitalized version of the string.
More specifically, make the first character have upper case and the rest lower case.

`casefold()`
Return a version of the string suitable for caseless comparisons.

center(*width*, *fillchar*=' ', /)

Return a centered string of length *width*.

Padding is done using the specified fill character (default is a space).

count(*sub*[, *start*[, *end*]]) → int

Return the number of non-overlapping occurrences of substring *sub* in string *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

encode(*encoding*='utf-8', *errors*='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a `UnicodeEncodeError`. Other possible values are 'ignore', 'replace' and 'xmlchar-refreplace' as well as any other name registered with `codecs.register_error` that can handle `UnicodeEncodeErrors`.

endswith(*suffix*[, *start*[, *end*]]) → bool

Return True if *S* ends with the specified *suffix*, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *suffix* can also be a tuple of strings to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → str

Return a formatted version of *S*, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → str

Return a formatted version of *S*, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(*width*, *fillchar*=' ', /)

Return a left-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(*chars*=None, /)

Return a copy of the string with leading whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in *x* will be mapped to the character at the same position in *y*. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(*prefix*, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(*suffix*, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(*old*, *new*, *count*=-1, /)

Return a copy with all occurrences of substring *old* replaced by *new*.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(width, fillchar=' ', /)

Return a right-justified string of length width.

Padding is done using the specified fill character (default is a space).

rpartition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\r`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(chars=None, /)

Return a copy of the string with trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

split(sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\r`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(keepends=False)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless `keepends` is given and true.

startswith(prefix[, start[, end]]) → bool

Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. `prefix` can also be a tuple of strings to try.

strip(chars=None, /)

Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppcased characters and all remaining cased characters have lower case.

translate(*table*, /)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(*width*, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.**ReactionType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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.

New 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`

__add__(*value*, /)

Return self+value.

__contains__(*key*, /)

Return bool(key in self).

__eq__(*value*, /)

Return self==value.

__format__(*format_spec*)

Return a formatted version of the string as described by *format_spec*.

__ge__(*value*, /)

Return self>=value.

__getattr__(*name*, /)

Return getattr(self, name).

__getitem__(*key*, /)

Return self[key].

__gt__(*value*, /)

Return self>value.

__hash__()

Return hash(self).

__iter__()

Implement iter(self).

__le__(*value*, /)

Return self<=value.

__len__()

Return len(self).

__lt__(*value*, /)

Return self<value.

__mod__(*value*, /)

Return self%value.

__mul__(*value*, /)

Return self*value.

__ne__(*value*, /)

Return self!=value.

__new__(*value*)

__repr__()

Return repr(self).

__rmod__(*value*, /)

Return value%self.

__rmul__(*value*, /)

Return value*self.

__sizeof__()

Return the size of the string in memory, in bytes.

__str__()

Return str(self).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(*width*, *fillchar*=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(*sub*[, *start*[, *end*]]) → int

Return the number of non-overlapping occurrences of substring *sub* in string *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

encode(*encoding*='utf-8', *errors*='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a `UnicodeEncodeError`. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with `codecs.register_error` that can handle `UnicodeEncodeErrors`.

endswith(*suffix*[, *start*[, *end*]]) → *bool*

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. *suffix* can also be a tuple of strings to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → *int*

Return the lowest index in S where substring *sub* is found, such that *sub* is contained within S[start:end]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → *str*

Return a formatted version of S, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → *str*

Return a formatted version of S, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → *int*

Return the lowest index in S where substring *sub* is found, such that *sub* is contained within S[start:end]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `‘.’.join([‘ab’, ‘pq’, ‘rs’]) -> ‘ab.pq.rs’`

ljust(width, fillchar=‘ ’, /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in `x` will be mapped to the character at the same position in `y`. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(*prefix*, /)

Return a `str` with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(*suffix*, /)

Return a `str` with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(*old*, *new*, *count=-1*, /)

Return a copy with all occurrences of substring `old` replaced by `new`.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument `count` is given, only the first `count` occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in `S` where substring `sub` is found, such that `sub` is contained within `S[start:end]`. Optional arguments `start` and `end` are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in `S` where substring `sub` is found, such that `sub` is contained within `S[start:end]`. Optional arguments `start` and `end` are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(*width*, *fillchar*=' ', /)

Return a right-justified string of length `width`.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(*sep=None*, *maxsplit=-1*)

Return a list of the substrings in the string, using `sep` as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars=None, /*)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

split(*sep=None, maxsplit=-1*)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix[, start[, end]]*) → bool

Return True if *S* starts with the specified prefix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *prefix* can also be a tuple of strings to try.

strip(*chars=None, /*)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table, /*)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(width, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.**ReplyLimit**(value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None)

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.

New 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`

__abs__()

abs(self)

__add__(value, /)

Return self+value.

__and__(value, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

__divmod__(value, /)

Return divmod(self, value).

__eq__(value, /)

Return self==value.

__float__()

float(self)

__floor__()

Flooring an Integral returns itself.

__floordiv__(value, /)

Return self//value.

__format__(format_spec, /)

Convert to a string according to format_spec.

__ge__(value, /)

Return self>=value.

`__getattr__`(*name*, /)
Return `getattr(self, name)`.

`__gt__`(*value*, /)
Return `self > value`.

`__hash__`()
Return `hash(self)`.

`__index__`()
Return `self` converted to an integer, if `self` is suitable for use as an index into a list.

`__int__`()
`int(self)`

`__invert__`()
`~self`

`__le__`(*value*, /)
Return `self <= value`.

`__lshift__`(*value*, /)
Return `self << value`.

`__lt__`(*value*, /)
Return `self < value`.

`__mod__`(*value*, /)
Return `self % value`.

`__mul__`(*value*, /)
Return `self * value`.

`__ne__`(*value*, /)
Return `self != value`.

`__neg__`()
`-self`

`__new__`(*value*)

`__or__`(*value*, /)
Return `self | value`.

`__pos__`()
`+self`

`__pow__`(*value*, *mod*=None, /)
Return `pow(self, value, mod)`.

`__radd__`(*value*, /)
Return `value + self`.

`__rand__`(*value*, /)
Return `value & self`.

`__rdivmod__`(*value*, /)
Return `divmod(value, self)`.

`__repr__`()
Return `repr(self)`.

__rfloordiv__(value, /)

Return value//self.

__rlshift__(value, /)

Return value<<self.

__rmod__(value, /)

Return value%self.

__rmul__(value, /)

Return value*self.

__ror__(value, /)

Return value|self.

__round__()

Rounding an Integral returns itself.

Rounding with an ndigits argument also returns an integer.

__rpow__(value, mod=None, /)

Return pow(value, self, mod).

__rrshift__(value, /)

Return value>>self.

__rshift__(value, /)

Return self>>value.

__rsub__(value, /)

Return value-self.

__rtruediv__(value, /)

Return value/self.

__rxor__(value, /)

Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.


```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If `byteorder` is 'big', the most significant byte is at the beginning of the byte array. If `byteorder` is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If `signed` is `False` and a negative integer is given, an `OverflowError` is raised.

`telegram.constants.SUPPORTED_WEBHOOK_PORTS = [443, 80, 88, 8443]`

Ports supported by `telegram.Bot.set_webhook.url`.

Type

List[int]

class `telegram.constants.StickerFormat`(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

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.

New in version 20.2.

ANIMATED = 'animated'

Animated sticker.

Type

`str`

STATIC = 'static'

Static sticker.

Type

`str`

VIDEO = 'video'

Video sticker.

Type

`str`

__add__(*value, /*)

Return self+value.

__contains__(*key, /*)

Return bool(key in self).

__eq__(*value, /*)

Return self==value.

__format__(*format_spec*)

Return a formatted version of the string as described by `format_spec`.

__ge__(*value, /*)

Return self>=value.

__getattribute__(*name*, /)

Return getattr(self, name).

__getitem__(*key*, /)

Return self[key].

__gt__(*value*, /)

Return self>value.

__hash__()

Return hash(self).

__iter__()

Implement iter(self).

__le__(*value*, /)

Return self<=value.

__len__()

Return len(self).

__lt__(*value*, /)

Return self<value.

__mod__(*value*, /)

Return self%value.

__mul__(*value*, /)

Return self*value.

__ne__(*value*, /)

Return self!=value.

__new__(*value*)

__repr__()

Return repr(self).

__rmod__(*value*, /)

Return value%self.

__rmul__(*value*, /)

Return value*self.

__sizeof__()

Return the size of the string in memory, in bytes.

__str__()

Return str(self).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(*width*, *fillchar*=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(*sub*[, *start*[, *end*]]) → int

Return the number of non-overlapping occurrences of substring *sub* in string *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

encode(*encoding*='utf-8', *errors*='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a `UnicodeEncodeError`. Other possible values are 'ignore', 'replace' and 'xmlchar-refreplace' as well as any other name registered with `codecs.register_error` that can handle `UnicodeEncodeErrors`.

endswith(*suffix*[, *start*[, *end*]]) → bool

Return True if *S* ends with the specified suffix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *suffix* can also be a tuple of strings to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → str

Return a formatted version of *S*, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → str

Return a formatted version of *S*, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(*chars=None, /*)

Return a copy of the string with leading whitespace removed.

If *chars* is given and not *None*, remove characters in *chars* instead.

static maketrans()

Return a translation table usable for *str.translate()*.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or *None*. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in *x* will be mapped to the character at the same position in *y*. If there is a third argument, it must be a string, whose characters will be mapped to *None* in the result.

partition(*sep, /*)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(*prefix, /*)

Return a *str* with the given prefix string removed if present.

If the string starts with the prefix string, return *string[len(prefix):]*. Otherwise, return a copy of the original string.

removesuffix(*suffix, /*)

Return a *str* with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return *string[:-len(suffix)]*. Otherwise, return a copy of the original string.

replace(*old, new, count=-1, /*)

Return a copy with all occurrences of substring *old* replaced by *new*.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub[, start[, end]]*) → *int*

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub[, start[, end]]*) → *int*

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises *ValueError* when the substring is not found.

rjust(*width, fillchar=' ', /*)

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(*sep=None*, *maxsplit=-1*)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to *None* (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars=None*, /)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not *None*, remove characters in *chars* instead.

split(*sep=None*, *maxsplit=-1*)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to *None* (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if *S* starts with the specified prefix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *prefix* can also be a tuple of strings to try.

strip(*chars=None*, /)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not *None*, remove characters in *chars* instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table*, /)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(*width*, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.**StickerLimit**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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.

New 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()`.

New 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()`.

New 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()`.

New 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()`.

New in version 20.2.

Type`int`**__abs__()**

abs(self)

__add__(value, /)

Return self+value.

__and__(value, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

__divmod__(value, /)

Return divmod(self, value).

__eq__(value, /)

Return self==value.

__float__()

float(self)

__floor__()

Flooring an Integral returns itself.

__floordiv__(value, /)

Return self//value.

__format__(format_spec, /)

Convert to a string according to format_spec.

__ge__(value, /)

Return self>=value.

__getattr__(name, /)

Return getattr(self, name).

__gt__(value, /)

Return self>value.

__hash__()

Return hash(self).

__index__()

Return self converted to an integer, if self is suitable for use as an index into a list.

__int__()

int(self)

`__invert__()`
~self

`__le__(value, /)`
Return self<=value.

`__lshift__(value, /)`
Return self<<value.

`__lt__(value, /)`
Return self<value.

`__mod__(value, /)`
Return self%value.

`__mul__(value, /)`
Return self*value.

`__ne__(value, /)`
Return self!=value.

`__neg__()`
-self

`__new__(value)`

`__or__(value, /)`
Return self|value.

`__pos__()`
+self

`__pow__(value, mod=None, /)`
Return pow(self, value, mod).

`__radd__(value, /)`
Return value+self.

`__rand__(value, /)`
Return value&self.

`__rdivmod__(value, /)`
Return divmod(value, self).

`__repr__()`
Return repr(self).

`__rfloordiv__(value, /)`
Return value//self.

`__rlshift__(value, /)`
Return value<<self.

`__rmod__(value, /)`
Return value%self.

`__rmul__(value, /)`
Return value*self.

`__ror__(value, /)`
Return value|self.

__round__()

Rounding an Integral returns itself.

Rounding with an ndigits argument also returns an integer.

__rpow__(value, mod=None, /)

Return pow(value, self, mod).

__rrshift__(value, /)

Return value>>self.

__rshift__(value, /)

Return self>>value.

__rsub__(value, /)

Return value-self.

__rtruediv__(value, /)

Return value/self.

__rxor__(value, /)

Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with float.is_integer.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(length=1, byteorder='big', *, signed=False)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An OverflowError is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

```
class telegram.constants.StickerSetLimit(value, names=None, *values, module=None,
                                         qualname=None, type=None, start=1, boundary=None)
```

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.

New 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()`.

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`

__abs__()
abs(self)

__add__(value, /)
Return self+value.

__and__(value, /)
Return self&value.

__bool__()
True if self else False

__ceil__()
Ceiling of an Integral returns itself.

__divmod__(value, /)
Return divmod(self, value).

__eq__(value, /)
Return self==value.

__float__()
float(self)

__floor__()
Flooring an Integral returns itself.

__floordiv__(value, /)
Return self//value.

__format__(format_spec, /)
Convert to a string according to format_spec.

__ge__(value, /)
Return self>=value.

__getattr__(name, /)
Return getattr(self, name).

__gt__(value, /)
Return self>value.

__hash__()
Return hash(self).

__index__()
Return self converted to an integer, if self is suitable for use as an index into a list.

__int__()
int(self)

__invert__()
~self

__le__(value, /)
Return self<=value.

__lshift__(value, /)
Return self<<value.

__lt__(value, /)
Return self<value.

`__mod__(value, /)`
Return `self%value`.

`__mul__(value, /)`
Return `self*value`.

`__ne__(value, /)`
Return `self!=value`.

`__neg__()`
`-self`

`__new__(value)`

`__or__(value, /)`
Return `self|value`.

`__pos__()`
`+self`

`__pow__(value, mod=None, /)`
Return `pow(self, value, mod)`.

`__radd__(value, /)`
Return `value+self`.

`__rand__(value, /)`
Return `value&self`.

`__rdivmod__(value, /)`
Return `divmod(value, self)`.

`__repr__()`
Return `repr(self)`.

`__rfloordiv__(value, /)`
Return `value//self`.

`__rlshift__(value, /)`
Return `value<<self`.

`__rmod__(value, /)`
Return `value%self`.

`__rmul__(value, /)`
Return `value*self`.

`__ror__(value, /)`
Return `value|self`.

`__round__()`
Rounding an Integral returns itself.
Rounding with an `ndigits` argument also returns an integer.

`__rpow__(value, mod=None, /)`
Return `pow(value, self, mod)`.

`__rrshift__(value, /)`
Return `value>>self`.

`__rshift__(value, /)`
Return `self>>value`.

__rsub__(value, /)

Return value-self.

__rtruediv__(value, /)

Return value/self.

__rxor__(value, /)

Return value^self.

__sizeof__()

Returns size in memory, in bytes.

__str__()

Return repr(self).

__sub__(value, /)

Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If *byteorder* is 'big', the most significant byte is at the beginning of the byte array. If *byteorder* is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with `float.is_integer`.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If *byteorder* is 'big', the most significant byte is at the beginning of the byte array. If *byteorder* is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If *signed* is False and a negative integer is given, an `OverflowError` is raised.

class telegram.constants.**StickerType**(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `str`, `enum.Enum`

This enum contains the available types of `telegram.Sticker`. The enum members of this enumeration are instances of `str` and can be treated as such.

New in version 20.0.

CUSTOM_EMOJI = 'custom_emoji'

Custom emoji sticker.

Type

`str`

MASK = 'mask'

Mask sticker.

Type

`str`

REGULAR = 'regular'

Regular sticker.

Type

`str`

__add__(value, /)

Return self+value.

__contains__(key, /)

Return bool(key in self).

__eq__(value, /)

Return self==value.

__format__(format_spec)

Return a formatted version of the string as described by format_spec.

__ge__(value, /)

Return self>=value.

__getattr__(name, /)

Return getattr(self, name).

__getitem__(key, /)

Return self[key].

__gt__(value, /)

Return self>value.

__hash__()

Return hash(self).

__iter__()

Implement iter(self).

__le__(value, /)

Return self<=value.

__len__()

Return len(self).

__lt__(value, /)

Return self<value.

__mod__(value, /)

Return self%value.

__mul__(value, /)

Return self*value.

__ne__(value, /)

Return self!=value.

__new__(value)

__repr__()

Return repr(self).

__rmod__(*value*, /)

Return *value*%*self*.

__rmul__(*value*, /)

Return *value***self*.

__sizeof__()

Return the size of the string in memory, in bytes.

__str__()

Return *str*(*self*).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(*width*, *fillchar*=' ', /)

Return a centered string of length *width*.

Padding is done using the specified fill character (default is a space).

count(*sub*[, *start*[, *end*]]) → *int*

Return the number of non-overlapping occurrences of substring *sub* in string *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

encode(*encoding*='utf-8', *errors*='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a `UnicodeEncodeError`. Other possible values are 'ignore', 'replace' and 'xmlchar-refreplace' as well as any other name registered with `codecs.register_error` that can handle `UnicodeEncodeErrors`.

endswith(*suffix*[, *start*[, *end*]]) → *bool*

Return `True` if *S* ends with the specified *suffix*, `False` otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *suffix* can also be a tuple of strings to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → *int*

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → *str*

Return a formatted version of *S*, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → *str*

Return a formatted version of *S*, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → *int*

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises *ValueError* when the substring is not found.

isalnum()

Return *True* if the string is an alpha-numeric string, *False* otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return *True* if the string is an alphabetic string, *False* otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return *True* if all characters in the string are ASCII, *False* otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return *True* if the string is a decimal string, *False* otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return *True* if the string is a digit string, *False* otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return *True* if the string is a valid Python identifier, *False* otherwise.

Call *keyword.iskeyword(s)* to test whether string *s* is a reserved identifier, such as “def” or “class”.

islower()

Return *True* if the string is a lowercase string, *False* otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return *True* if the string is a numeric string, *False* otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return *True* if the string is printable, *False* otherwise.

A string is printable if all of its characters are considered printable in *repr()* or if it is empty.

isspace()

Return *True* if the string is a whitespace string, *False* otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(prefix, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(suffix, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(old, new, count=-1, /)

Return a copy with all occurrences of substring old replaced by new.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(*width*, *fillchar*=' ', /)

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars*=None, /)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

split(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if *S* starts with the specified prefix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *prefix* can also be a tuple of strings to try.

strip(*chars=None, /*)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table, /*)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(*width, /*)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class telegram.constants.UpdateType(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `str`, `enum.Enum`

This enum contains the available types of `telegram.Update`. The enum members of this enumeration are instances of `str` and can be treated as such.

New in version 20.0.

CALLBACK_QUERY = 'callback_query'

Updates with `telegram.Update.callback_query`.

Type

`str`

CHANNEL_POST = 'channel_post'

Updates with `telegram.Update.channel_post`.

Type

`str`

CHAT_BOOST = 'chat_boost'

Updates with *telegram.Update.chat_boost*.

New 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

EDITED_CHANNEL_POST = 'edited_channel_post'

Updates with *telegram.Update.edited_channel_post*.

Type

str

EDITED_MESSAGE = 'edited_message'

Updates with *telegram.Update.edited_message*.

Type

str

INLINE_QUERY = 'inline_query'

Updates with *telegram.Update.inline_query*.

Type

str

MESSAGE = 'message'

Updates with *telegram.Update.message*.

Type

str

MESSAGE_REACTION = 'message_reaction'

Updates with *telegram.Update.message_reaction*.

New in version 20.8.

Type

str

MESSAGE_REACTION_COUNT = 'message_reaction_count'

Updates with *telegram.Update.message_reaction_count*.

New 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

REMOVED_CHAT_BOOST = 'removed_chat_boost'

Updates with *telegram.Update.removed_chat_boost*.

New in version 20.8.

Type

str

SHIPPING_QUERY = 'shipping_query'

Updates with *telegram.Update.shipping_query*.

Type

str

__add__(value, /)

Return self+value.

__contains__(key, /)

Return bool(key in self).

__eq__(value, /)

Return self==value.

__format__(format_spec)

Return a formatted version of the string as described by format_spec.

__ge__(value, /)

Return self>=value.

__getattr__(name, /)

Return getattr(self, name).

__getitem__(key, /)

Return self[key].

__gt__(value, /)

Return self>value.

__hash__()

Return hash(self).

__iter__()

Implement iter(self).

__le__(value, /)

Return self<=value.

__len__()

Return len(self).

__lt__(value, /)

Return self<value.

__mod__(value, /)

Return self%value.

__mul__(value, /)

Return self*value.

__ne__(value, /)

Return self!=value.

__new__(value)

__repr__()

Return repr(self).

__rmod__(value, /)

Return value%self.

__rmul__(value, /)

Return value*self.

__sizeof__()

Return the size of the string in memory, in bytes.

__str__()

Return str(self).

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(width, fillchar=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(sub[, start[, end]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

encode(encoding='utf-8', errors='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding

errors raise a `UnicodeEncodeError`. Other possible values are 'ignore', 'replace' and 'xmlchar-refreplace' as well as any other name registered with `codecs.register_error` that can handle `UnicodeEncodeErrors`.

endswith(*suffix*[, *start*[, *end*]]) → bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. *suffix* can also be a tuple of strings to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int

Return the lowest index in S where substring *sub* is found, such that *sub* is contained within S[start:end]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → str

Return a formatted version of S, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → str

Return a formatted version of S, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in S where substring *sub* is found, such that *sub* is contained within S[start:end]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call `keyword.iskeyword(s)` to test whether string `s` is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in `repr()` or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `‘.’.join([‘ab’, ‘pq’, ‘rs’]) -> ‘ab.pq.rs’`

ljust(width, fillchar=‘ ’, /)

Return a left-justified string of length `width`.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If `chars` is given and not None, remove characters in `chars` instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in

x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(prefix, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return string[len(prefix):]. Otherwise, return a copy of the original string.

removesuffix(suffix, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return string[:-len(suffix)]. Otherwise, return a copy of the original string.

replace(old, new, count=-1, /)

Return a copy with all occurrences of substring old replaced by new.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind(sub[, start[, end]]) → int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

rindex(sub[, start[, end]]) → int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

rjust(width, fillchar=' ', /)

Return a right-justified string of length width.

Padding is done using the specified fill character (default is a space).

rpartition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars=None, /*)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not *None*, remove characters in *chars* instead.

split(*sep=None, maxsplit=-1*)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to *None* (the default value), will split on any whitespace character (including `\n`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix[, start[, end]]*) → *bool*

Return True if *S* starts with the specified prefix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *prefix* can also be a tuple of strings to try.

strip(*chars=None, /*)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not *None*, remove characters in *chars* instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table, /*)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or *None*.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to *None* are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(*width, /*)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

```
class telegram.constants.UserProfilePhotosLimit(value, names=None, *values, module=None,  
                                              qualname=None, type=None, start=1,  
                                              boundary=None)
```

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.

New 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`

__abs__()

abs(self)

__add__(*value, /*)

Return self+value.

__and__(*value, /*)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

__divmod__(*value, /*)

Return divmod(self, value).

__eq__(*value, /*)

Return self==value.

__float__()

float(self)

__floor__()

Flooring an Integral returns itself.

__floordiv__(*value, /*)

Return self//value.

__format__(*format_spec, /*)

Convert to a string according to format_spec.

__ge__(*value, /*)

Return self>=value.

__getattr__(*name, /*)

Return getattr(self, name).

__gt__(*value, /*)

Return self>value.

`__hash__()`

Return hash(self).

`__index__()`

Return self converted to an integer, if self is suitable for use as an index into a list.

`__int__()`

int(self)

`__invert__()`

~self

`__le__(value, /)`

Return self<=value.

`__lshift__(value, /)`

Return self<<value.

`__lt__(value, /)`

Return self<value.

`__mod__(value, /)`

Return self%value.

`__mul__(value, /)`

Return self*value.

`__ne__(value, /)`

Return self!=value.

`__neg__()`

-self

`__new__(value)`

`__or__(value, /)`

Return self|value.

`__pos__()`

+self

`__pow__(value, mod=None, /)`

Return pow(self, value, mod).

`__radd__(value, /)`

Return value+self.

`__rand__(value, /)`

Return value&self.

`__rdivmod__(value, /)`

Return divmod(value, self).

`__repr__()`

Return repr(self).

`__rfloordiv__(value, /)`

Return value//self.

`__rlshift__(value, /)`

Return value<<self.

`__rmod__(value, /)`
Return `value%self`.

`__rmul__(value, /)`
Return `value*self`.

`__ror__(value, /)`
Return `value|self`.

`__round__()`
Rounding an Integral returns itself.
Rounding with an `ndigits` argument also returns an integer.

`__rpow__(value, mod=None, /)`
Return `pow(value, self, mod)`.

`__rrshift__(value, /)`
Return `value>>self`.

`__rshift__(value, /)`
Return `self>>value`.

`__rsub__(value, /)`
Return `value-self`.

`__rtruediv__(value, /)`
Return `value/self`.

`__rxor__(value, /)`
Return `value^self`.

`__sizeof__()`
Returns size in memory, in bytes.

`__str__()`
Return `repr(self)`.

`__sub__(value, /)`
Return `self-value`.

`__truediv__(value, /)`
Return `self/value`.

`__trunc__()`
Truncating an Integral returns itself.

`__xor__(value, /)`
Return `self^value`.

`as_integer_ratio()`
Return a pair of integers, whose ratio is equal to the original int.
The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with float.is_integer.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An OverflowError is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of

the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use `'big'`.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

class telegram.constants.WebhookLimit(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

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.

New 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`

__abs__()

abs(self)

__add__(*value, /*)

Return self+value.

__and__(*value, /*)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

__divmod__(*value, /*)

Return divmod(self, value).

__eq__(*value, /*)

Return self==value.

__float__()
float(self)

__floor__()
Flooring an Integral returns itself.

__floordiv__(value, /)
Return self//value.

__format__(format_spec, /)
Convert to a string according to format_spec.

__ge__(value, /)
Return self>=value.

__getattr__(name, /)
Return getattr(self, name).

__gt__(value, /)
Return self>value.

__hash__()
Return hash(self).

__index__()
Return self converted to an integer, if self is suitable for use as an index into a list.

__int__()
int(self)

__invert__()
~self

__le__(value, /)
Return self<=value.

__lshift__(value, /)
Return self<<value.

__lt__(value, /)
Return self<value.

__mod__(value, /)
Return self%value.

__mul__(value, /)
Return self*value.

__ne__(value, /)
Return self!=value.

__neg__()
-self

__new__(value)

__or__(value, /)
Return self|value.

__pos__()
+self

__pow__(*value*, *mod*=None, /)
Return pow(self, value, mod).

__radd__(*value*, /)
Return value+self.

__rand__(*value*, /)
Return value&self.

__rdivmod__(*value*, /)
Return divmod(value, self).

__repr__()
Return repr(self).

__rfloordiv__(*value*, /)
Return value//self.

__rlshift__(*value*, /)
Return value<<self.

__rmod__(*value*, /)
Return value%self.

__rmul__(*value*, /)
Return value*self.

__ror__(*value*, /)
Return value|self.

__round__()
Rounding an Integral returns itself.
Rounding with an ndigits argument also returns an integer.

__rpow__(*value*, *mod*=None, /)
Return pow(value, self, mod).

__rrshift__(*value*, /)
Return value>>self.

__rshift__(*value*, /)
Return self>>value.

__rsub__(*value*, /)
Return value-self.

__rtruediv__(*value*, /)
Return value/self.

__rxor__(*value*, /)
Return value^self.

__sizeof__()
Returns size in memory, in bytes.

__str__()
Return repr(self).

__sub__(*value*, /)
Return self-value.

__truediv__(value, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(value, /)

Return self^value.

as_integer_ratio()

Return a pair of integers, whose ratio is equal to the original int.

The ratio is in lowest terms and has a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_integer()

Returns True. Exists for duck type compatibility with float.is_integer.

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(length=1, byteorder='big', *, signed=False)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An OverflowError is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

`telegram.constants.ZERO_DATE = datetime.datetime(1970, 1, 1, 0, 0, tzinfo=datetime.timezone.utc)`

`datetime.datetime`, value of unix 0. This date literal is used in `telegram.InaccessibleMessage`

New in version 20.8.

10.3.2 telegram.error Module

This module contains classes that represent Telegram errors.

Changed in version 20.0: Replaced Unauthorized by *Forbidden*.

exception `telegram.error.BadRequest`(message)

Bases: `telegram.error.NetworkError`

Raised when Telegram could not process the request correctly.

exception `telegram.error.ChatMigrated`(new_chat_id)

Bases: `telegram.error.TelegramError`

Raised when the requested group chat migrated to supergroup and has a new chat id.

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()`.

New 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.

New 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 were exceeded.

Changed in version 20.0: *retry_after* is now an integer to comply with the Bot API.

Parameters

retry_after ([int](#)) – Time in seconds, after which the bot can retry the request.

retry_after

Time in seconds, after which the bot can retry the request.

Type

[int](#)

__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.

New in version 20.0.

10.3.3 telegram.helpers Module

This module contains convenience helper functions.

Changed in version 20.0: Previously, the contents of this module were available through the (no longer existing) module `telegram.utils.helpers`.

`telegram.helpers.create_deep_linked_url` (*bot_username*, *payload=None*, *group=False*)

Creates a deep-linked URL for this *bot_username* with the specified *payload*. See <https://core.telegram.org/bots/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`

10.3.4 telegram.request Module

New in version 20.0.

BaseRequest

class telegram.request.BaseRequest

Bases: `typing.AsyncContextManager`, `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()
```

Use In

- `telegram.ext.ApplicationBuilder.get_updates_request()`
 - `telegram.ext.ApplicationBuilder.request()`
-

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](#)

New in version 20.0.

DEFAULT_NONE = None

A special object that indicates that an argument of a function was not explicitly passed. Used for the timeout parameters of `post()` and `do_request()`.

Example

When calling `request.post(url)`, `request` should use the default timeouts set on initialization. When calling `request.post(url, connect_timeout=5, read_timeout=None)`, `request` should use 5 for the connect timeout and `None` for the read timeout.

Use `if parameter is (not) BaseRequest.DEFAULT_NONE`: to check if the parameter was set.

Type

object

USER_AGENT = 'python-telegram-bot v20.8 (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.

abstract async do_request(url, method, request_data=None, read_timeout=None, write_timeout=None, connect_timeout=None, pool_timeout=None)

Makes a request to the Bot API. Must be implemented by a subclass.

Warning: This method will be called by `post()` and `retrieve()`. It should *not* be called manually.

Parameters

- **url** (str) – The URL to request.
- **method** (str) – HTTP method (i.e. 'POST', 'GET', etc.).
- **request_data** (`telegram.request.RequestData`, optional) – An object containing information about parameters and files to upload for the request.
- **read_timeout** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a response from Telegram's server instead of the time specified during creating of this object. Defaults to `DEFAULT_NONE`.
- **write_timeout** (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a write operation to complete (in terms of a network socket; i.e. POSTing a request or uploading a file) instead of the time specified during creating of this object. Defaults to `DEFAULT_NONE`.

- **`connect_timeout`** (`float` | `None`, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a connection attempt to a server to succeed instead of the time specified during creating of this object. Defaults to `DEFAULT_NONE`.
- **`pool_timeout`** (`float` | `None`, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a connection to become available instead of the time specified during creating of this object. Defaults to `DEFAULT_NONE`.

Returns

The HTTP return code & the payload part of the server response.

Return type

`Tuple[int, bytes]`

`abstract async initialize()`

Initialize resources used by this class. Must be implemented by a subclass.

`static parse_json_payload(payload)`

Parse the JSON returned from Telegram.

Tip: By default, this method uses the standard library's `json.loads()` and `errors="replace"` in `bytes.decode()`. You can override it to customize either of these behaviors.

Parameters

`payload` (`bytes`) – The UTF-8 encoded JSON payload as returned by Telegram.

Returns

A JSON parsed as Python dict with results.

Return type

`dict`

Raises

`TelegramError` – If loading the JSON data failed

`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.

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`.

New in version 20.7.

Warning: For now this property does not need to be implemented by subclasses and will raise `NotImplementedError` if accessed without being overridden. However, in future versions, this property will be abstract and must 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`

abstract 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.

New 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.

property json_payload

The `parameters` as UTF-8 encoded JSON payload.

Tip: By default, this property uses the standard library's `json.dumps()`. To use a custom library for JSON encoding, you can directly encode the keys of `parameters` - note that string valued keys should not be JSON encoded.

property multipart_data

Gives the files contained in this object as mapping of part name to encoded content.

property parameters

Gives the parameters as mapping of parameter name to the parameter value, which can be a single object of type `int`, `float`, `str` or `bool` or any (possibly nested) composition of lists, tuples and dictionaries, where each entry, key and value is of one of the mentioned types.

parametrized_url(*url, encode_kwargs=None*)

Shortcut for attaching the return value of `url_encoded_parameters()` to the `url`.

Parameters

- **url** (`str`) – The URL the parameters will be attached to.
- **encode_kwargs** (`Dict[str, any]`, optional) – Additional keyword arguments to pass along to `urllib.parse.urlencode()`.

`url_encoded_parameters`(*encode_kwargs=None*)

Encodes the parameters with `urllib.parse.urlencode()`.

Parameters

encode_kwargs (Dict[str, any], optional) – Additional keyword arguments to pass along to `urllib.parse.urlencode()`.

HTTPXRequest

```
class telegram.request.HTTPXRequest(connection_pool_size=1, proxy_url=None, read_timeout=5.0,
                                     write_timeout=5.0, connect_timeout=5.0, pool_timeout=1.0,
                                     http_version='1.1', socket_options=None, proxy=None)
```

Bases: `telegram.request.BaseRequest`

Implementation of `BaseRequest` using the library `httpx`.

Use In

- `telegram.ext.ApplicationBuilder.get_updates_request()`
 - `telegram.ext.ApplicationBuilder.request()`
-

New in version 20.0.

Parameters

- *connection_pool_size* (int, optional) – Number of connections to keep in the connection pool. Defaults to 1.

Note: Independent of the value, one additional connection will be reserved for `telegram.Bot.get_updates()`.

- *proxy_url* (str, optional) – Legacy name for *proxy*, kept for backward compatibility. Defaults to `None`.

Deprecated since version 20.7.

- *read_timeout* (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a response from Telegram’s server. This value is used unless a different value is passed to `do_request()`. Defaults to 5.
- *write_timeout* (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a write operation to complete (in terms of a network socket; i.e. POSTing a request or uploading a file). This value is used unless a different value is passed to `do_request()`. Defaults to 5.
- *connect_timeout* (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a connection attempt to a server to succeed. This value is used unless a different value is passed to `do_request()`. Defaults to 5.
- *pool_timeout* (float | None, optional) – If passed, specifies the maximum amount of time (in seconds) to wait for a connection to become available. This value is used unless a different value is passed to `do_request()`. Defaults to 1.

Warning: With a finite pool timeout, you must expect `telegram.error.TimedOut` exceptions to be thrown when more requests are made simultaneously than there are connections in the connection pool!

- **`http_version`** (`str`, optional) – If "2" or "2.0", HTTP/2 will be used instead of HTTP/1.1. Defaults to "1.1".

New 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.

New 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.
-

New in version 20.7.

`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](#).

New 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\(\)](#).

10.3.5 telegram.warnings Module

This module contains classes used for warnings issued by this library.

New in version 20.0.

exception telegram.warnings.PTBDeprecationWarning

Bases: [telegram.warnings.PTBUserWarning](#), [DeprecationWarning](#)

Custom warning class for deprecations in this library.

Changed in version 20.0: Renamed TelegramDeprecationWarning to PTBDeprecationWarning.

exception telegram.warnings.PTBRuntimeWarning

Bases: [telegram.warnings.PTBUserWarning](#), [RuntimeWarning](#)

Custom runtime warning class used for warnings in this library.

New in version 20.0.

exception telegram.warnings.PTBUserWarning

Bases: [UserWarning](#)

Custom user warning class used for warnings in this library.

See also:

[Exceptions, Warnings and Logging](#)

New in version 20.0.

10.4 Examples

In this section we display small examples to show what a bot written with `python-telegram-bot` looks like. Some bots focus on one specific aspect of the Telegram Bot API while others focus on one of the mechanics of this library. Except for the [rawapibot.py](#) example, they all use the high-level framework this library provides with the [telegram.ext](#) submodule.

All examples are licensed under the [CC0 License](#) and are therefore fully dedicated to the public domain. You can use them as the base for your own bots without worrying about copyrights.

Do note that we ignore one pythonic convention. Best practice would dictate, in many handler callbacks function signatures, to replace the argument `context` with an underscore, since `context` is an unused local variable in those callbacks. However, since these are examples and not having a name for that argument confuses beginners, we decided to have it present.

10.4.1 echobot.py

This is probably the base for most of the bots made with `python-telegram-bot`. It simply replies to each text message with a message that contains the same text.

10.4.2 timerbot.py

This bot uses the `telegram.ext.JobQueue` class to send timed messages. The user sets a timer by using `/set` command with a specific time, for example `/set 30`. The bot then sets up a job to send a message to that user after 30 seconds. The user can also cancel the timer by sending `/unset`. To learn more about the `JobQueue`, read [this wiki article](#). Note: To use `JobQueue`, you must install PTB via `pip install "python-telegram-bot[job-queue]"`

10.4.3 conversationbot.py

A common task for a bot is to ask information from the user. In v5.0 of this library, we introduced the `telegram.ext.ConversationHandler` for that exact purpose. This example uses it to retrieve user-information in a conversation-like style. To get a better understanding, take a look at the [state diagram](#).

10.4.4 conversationbot2.py

A more complex example of a bot that uses the `ConversationHandler`. It is also more confusing. Good thing there is a [fancy state diagram](#) for this one, too!

10.4.5 nestedconversationbot.py

A even more complex example of a bot that uses the nested `ConversationHandlers`. While it's certainly not that complex that you couldn't built it without nested `ConversationHandlers`, it gives a good impression on how to work with them. Of course, there is a [fancy state diagram](#) for this example, too!

10.4.6 persistentconversationbot.py

A basic example of a bot store conversation state and `user_data` over multiple restarts.

10.4.7 inlinekeyboard.py

This example sheds some light on inline keyboards, callback queries and message editing. A wiki site explaining this examples lives [here](#).

10.4.8 inlinekeyboard2.py

A more complex example about inline keyboards, callback queries and message editing. This example showcases how an interactive menu could be build using inline keyboards.

10.4.9 deeplinking.py

A basic example on how to use deeplinking with inline keyboards.

10.4.10 inlinebot.py

A basic example of an [inline bot](#). Don't forget to enable inline mode with [@BotFather](#).

10.4.11 pollbot.py

This example sheds some light on polls, poll answers and the corresponding handlers.

10.4.12 passportbot.py

A basic example of a bot that can accept passports. Use in combination with the [HTML page](#). Don't forget to enable and configure payments with [@BotFather](#). Check out this [guide](#) on Telegram passports in PTB. Note: To use Telegram Passport, you must install PTB via `pip install "python-telegram-bot[passport]"`

10.4.13 paymentbot.py

A basic example of a bot that can accept payments. Don't forget to enable and configure payments with [@BotFather](#).

10.4.14 errorhandlerbot.py

A basic example on how to set up a custom error handler.

10.4.15 chatmemberbot.py

A basic example on how `(my_)chat_member` updates can be used.

10.4.16 webappbot.py

A basic example of how [Telegram WebApps](#) can be used. Use in combination with the [HTML page](#). For your convenience, this file is hosted by the PTB team such that you don't need to host it yourself. Uses the [iro.js](#) JavaScript library to showcase a user interface that is hard to achieve with native Telegram functionality.

10.4.17 contexttypesbot.py

This example showcases how `telegram.ext.ContextTypes` can be used to customize the `context` argument of handler and job callbacks.

10.4.18 customwebhookbot.py

This example showcases how a custom webhook setup can be used in combination with `telegram.ext.Application`.

10.4.19 arbitrarycallbackdatabot.py

This example showcases how PTBs “arbitrary callback data” feature can be used. Note: To use arbitrary callback data, you must install PTB via `pip install "python-telegram-bot[callback-data]"`

10.4.20 Pure API

The `rawapibot.py` example uses only the pure, “bare-metal” API wrapper.

arbitrarycallbackdatabot.py

```
1  #!/usr/bin/env python
2  # pylint: disable=unused-argument
3  # This program is dedicated to the public domain under the CC0 license.
4
5  """This example showcases how PTBs "arbitrary callback data" feature can be used.
6
7  For detailed info on arbitrary callback data, see the wiki page at
8  https://github.com/python-telegram-bot/python-telegram-bot/wiki/Arbitrary-callback\_
9  ↪ data
10
11 Note:
12 To use arbitrary callback data, you must install PTB via
13 `pip install "python-telegram-bot[callback-data]"`
14 """
15
16 import logging
17 from typing import List, Tuple, cast
18
19 from telegram import InlineKeyboardButton, InlineKeyboardMarkup, Update
20 from telegram.ext import (
21     Application,
22     CallbackQueryHandler,
23     CommandHandler,
24     ContextTypes,
25     InvalidCallbackData,
26     PicklePersistence,
27 )
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 async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
39     """Sends a message with 5 inline buttons attached."""
40     number_list: List[int] = []
41     await update.message.reply_text("Please choose:", reply_markup=build_
42     ↪ keyboard(number_list))
43
44
```

(continues on next page)

(continued from previous page)

```

43 async def help_command(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
44     """Displays info on how to use the bot."""
45     await update.message.reply_text(
46         "Use /start to test this bot. Use /clear to clear the stored data so that you
↳ can see "
47         "what happens, if the button data is not available. "
48     )
49
50
51 async def clear(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
52     """Clears the callback data cache"""
53     context.bot.callback_data_cache.clear_callback_data()
54     context.bot.callback_data_cache.clear_callback_queries()
55     await update.effective_message.reply_text("All clear!")
56
57
58 def build_keyboard(current_list: List[int]) -> InlineKeyboardMarkup:
59     """Helper function to build the next inline keyboard."""
60     return InlineKeyboardMarkup.from_column(
61         [InlineKeyboardButton(str(i), callback_data=(i, current_list)) for i in
↳ range(1, 6)]
62     )
63
64
65 async def list_button(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
66     """Parses the CallbackQuery and updates the message text."""
67     query = update.callback_query
68     await query.answer()
69     # Get the data from the callback_data.
70     # If you're using a type checker like MyPy, you'll have to use typing.cast
71     # to make the checker get the expected type of the callback_data
72     number, number_list = cast(Tuple[int, List[int]], query.data)
73     # append the number to the list
74     number_list.append(number)
75
76     await query.edit_message_text(
77         text=f"So far you've selected {number_list}. Choose the next item:",
78         reply_markup=build_keyboard(number_list),
79     )
80
81     # we can delete the data stored for the query, because we've replaced the buttons
82     context.drop_callback_data(query)
83
84
85 async def handle_invalid_button(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
↳ None:
86     """Informs the user that the button is no longer available."""
87     await update.callback_query.answer()
88     await update.effective_message.edit_text(
89         "Sorry, I could not process this button click Please send /start to get a
↳ new keyboard."
90     )
91
92
93 def main() -> None:
94     """Run the bot."""

```

(continues on next page)

(continued from previous page)

```

95     # We use persistence to demonstrate how buttons can still work after the bot was
↪ restarted
96     persistence = PicklePersistence(filepath="arbitrarycallbackdatabot")
97     # Create the Application and pass it your bot's token.
98     application = (
99         Application.builder()
100         .token("TOKEN")
101         .persistence(persistence)
102         .arbitrary_callback_data(True)
103         .build()
104     )
105
106     application.add_handler(CommandHandler("start", start))
107     application.add_handler(CommandHandler("help", help_command))
108     application.add_handler(CommandHandler("clear", clear))
109     application.add_handler(
110         CallbackQueryHandler(handle_invalid_button, pattern=InvalidCallbackData)
111     )
112     application.add_handler(CallbackQueryHandler(list_button))
113
114     # Run the bot until the user presses Ctrl-C
115     application.run_polling(allowed_updates=Update.ALL_TYPES)
116
117
118 if __name__ == "__main__":
119     main()

```

chatmemberbot.py

```

1  #!/usr/bin/env python
2  # pylint: disable=unused-argument
3  # This program is dedicated to the public domain under the CC0 license.
4
5  """
6  Simple Bot to handle '(my_)chat_member' updates.
7  Greets new users & keeps track of which chats the bot is in.
8
9  Usage:
10 Press Ctrl-C on the command line or send a signal to the process to stop the
11 bot.
12 """
13
14 import logging
15 from typing import Optional, Tuple
16
17 from telegram import Chat, ChatMember, ChatMemberUpdated, Update
18 from telegram.constants import ParseMode
19 from telegram.ext import (
20     Application,
21     ChatMemberHandler,
22     CommandHandler,
23     ContextTypes,
24     MessageHandler,
25     filters,
26 )

```

(continues on next page)

(continued from previous page)

```

27
28 # Enable logging
29
30 logging.basicConfig(
31     format="%asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
32 )
33
34 # set higher logging level for httpx to avoid all GET and POST requests being logged
35 logging.getLogger("httpx").setLevel(logging.WARNING)
36
37 logger = logging.getLogger(__name__)
38
39
40 def extract_status_change(chat_member_update: ChatMemberUpdated) ->
↳ Optional[Tuple[bool, bool]]:
41     """Takes a ChatMemberUpdated instance and extracts whether the 'old_chat_member'
↳ was a member
42     of the chat and whether the 'new_chat_member' is a member of the chat. Returns
↳ None, if
43     the status didn't change.
44     """
45     status_change = chat_member_update.difference().get("status")
46     old_is_member, new_is_member = chat_member_update.difference().get("is_member",
↳ (None, None))
47
48     if status_change is None:
49         return None
50
51     old_status, new_status = status_change
52     was_member = old_status in [
53         ChatMember.MEMBER,
54         ChatMember.OWNER,
55         ChatMember.ADMINISTRATOR,
56     ] or (old_status == ChatMember.RESTRICTED and old_is_member is True)
57     is_member = new_status in [
58         ChatMember.MEMBER,
59         ChatMember.OWNER,
60         ChatMember.ADMINISTRATOR,
61     ] or (new_status == ChatMember.RESTRICTED and new_is_member is True)
62
63     return was_member, is_member
64
65
66 async def track_chats(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
67     """Tracks the chats the bot is in."""
68     result = extract_status_change(update.my_chat_member)
69     if result is None:
70         return
71     was_member, is_member = result
72
73     # Let's check who is responsible for the change
74     cause_name = update.effective_user.full_name
75
76     # Handle chat types differently:
77     chat = update.effective_chat
78     if chat.type == Chat.PRIVATE:

```

(continues on next page)

(continued from previous page)

```

79         if not was_member and is_member:
80             # This may not be really needed in practice because most clients will
81             ↪ automatically
82             # send a /start command after the user unblocks the bot, and start_
83             ↪ private_chat()
84             # will add the user to "user_ids".
85             # We're including this here for the sake of the example.
86             logger.info("%s unblocked the bot", cause_name)
87             context.bot_data.setdefault("user_ids", set()).add(chat.id)
88         elif was_member and not is_member:
89             logger.info("%s blocked the bot", cause_name)
90             context.bot_data.setdefault("user_ids", set()).discard(chat.id)
91         elif chat.type in [Chat.GROUP, Chat.SUPERGROUP]:
92             if not was_member and is_member:
93                 logger.info("%s added the bot to the group %s", cause_name, chat.title)
94                 context.bot_data.setdefault("group_ids", set()).add(chat.id)
95             elif was_member and not is_member:
96                 logger.info("%s removed the bot from the group %s", cause_name, chat.
97                 ↪ title)
98                 context.bot_data.setdefault("group_ids", set()).discard(chat.id)
99             elif not was_member and is_member:
100                 logger.info("%s added the bot to the channel %s", cause_name, chat.title)
101                 context.bot_data.setdefault("channel_ids", set()).add(chat.id)
102             elif was_member and not is_member:
103                 logger.info("%s removed the bot from the channel %s", cause_name, chat.title)
104                 context.bot_data.setdefault("channel_ids", set()).discard(chat.id)
105
106     async def show_chats(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
107         """Shows which chats the bot is in"""
108         user_ids = ", ".join(str(uid) for uid in context.bot_data.setdefault("user_ids",
109         ↪ set()))
110         group_ids = ", ".join(str(gid) for gid in context.bot_data.setdefault("group_ids",
111         ↪ set()))
112         channel_ids = ", ".join(str(cid) for cid in context.bot_data.setdefault("channel_
113         ↪ ids", set()))
114         text = (
115             f"@{context.bot.username} is currently in a conversation with the user IDs
116             ↪ {user_ids}."
117             f" Moreover it is a member of the groups with IDs {group_ids} "
118             f"and administrator in the channels with IDs {channel_ids}."
119         )
120         await update.effective_message.reply_text(text)
121
122     async def greet_chat_members(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
123     ↪ None:
124         """Greets new users in chats and announces when someone leaves"""
125         result = extract_status_change(update.chat_member)
126         if result is None:
127             return
128
129         was_member, is_member = result
130         cause_name = update.chat_member.from_user.mention_html()
131         member_name = update.chat_member.new_chat_member.user.mention_html()

```

(continues on next page)

(continued from previous page)

```

127     if not was_member and is_member:
128         await update.effective_chat.send_message(
129             f"{member_name} was added by {cause_name}. Welcome!",
130             parse_mode=ParseMode.HTML,
131         )
132     elif was_member and not is_member:
133         await update.effective_chat.send_message(
134             f"{member_name} is no longer with us. Thanks a lot, {cause_name} ...",
135             parse_mode=ParseMode.HTML,
136         )
137
138
139 async def start_private_chat(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
140     None:
141         """Greets the user and records that they started a chat with the bot if it's a
142         private chat.
143         Since no `my_chat_member` update is issued when a user starts a private chat with
144         the bot
145         for the first time, we have to track it explicitly here.
146         """
147         user_name = update.effective_user.full_name
148         chat = update.effective_chat
149         if chat.type != Chat.PRIVATE or chat.id in context.bot_data.get("user_ids",
150             set()):
151             return
152
153         logger.info("%s started a private chat with the bot", user_name)
154         context.bot_data.setdefault("user_ids", set()).add(chat.id)
155
156         await update.effective_message.reply_text(
157             f"Welcome {user_name}. Use /show_chats to see what chats I'm in."
158         )
159
160
161 def main() -> None:
162     """Start the bot."""
163     # Create the Application and pass it your bot's token.
164     application = Application.builder().token("TOKEN").build()
165
166     # Keep track of which chats the bot is in
167     application.add_handler(ChatMemberHandler(track_chats, ChatMemberHandler.MY_CHAT_
168         MEMBER))
169     application.add_handler(CommandHandler("show_chats", show_chats))
170
171     # Handle members joining/leaving chats.
172     application.add_handler(ChatMemberHandler(greet_chat_members, ChatMemberHandler.
173         CHAT_MEMBER))
174
175     # Interpret any other command or text message as a start of a private chat.
176     # This will record the user as being in a private chat with bot.
177     application.add_handler(MessageHandler(filters.ALL, start_private_chat))
178
179     # Run the bot until the user presses Ctrl-C
180     # We pass 'allowed_updates' handle *all* updates including `chat_member` updates
181     # To reset this, simply pass `allowed_updates=[]`
182     application.run_polling(allowed_updates=Update.ALL_TYPES)

```

(continues on next page)

(continued from previous page)

```

177
178
179 if __name__ == "__main__":
180     main()

```

contexttypesbot.py

```

1  #!/usr/bin/env python
2  # pylint: disable=unused-argument
3  # This program is dedicated to the public domain under the CC0 license.
4
5  """
6  Simple Bot to showcase `telegram.ext.ContextTypes`.
7
8  Usage:
9  Press Ctrl-C on the command line or send a signal to the process to stop the
10 bot.
11 """
12
13 import logging
14 from collections import defaultdict
15 from typing import DefaultDict, Optional, Set
16
17 from telegram import InlineKeyboardButton, InlineKeyboardMarkup, Update
18 from telegram.constants import ParseMode
19 from telegram.ext import (
20     Application,
21     CallbackContext,
22     CallbackQueryHandler,
23     CommandHandler,
24     ContextTypes,
25     ExtBot,
26     TypeHandler,
27 )
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
39 class ChatData:
40     """Custom class for chat_data. Here we store data per message."""
41
42     def __init__(self) -> None:
43         self.clicks_per_message: DefaultDict[int, int] = defaultdict(int)
44
45
46 # The [ExtBot, dict, ChatData, dict] is for type checkers like mypy
47 class CustomContext(CallbackContext[ExtBot, dict, ChatData, dict]):
48     """Custom class for context."""

```

(continues on next page)

(continued from previous page)

```

49
50 def __init__(
51     self,
52     application: Application,
53     chat_id: Optional[int] = None,
54     user_id: Optional[int] = None,
55 ):
56     super().__init__(application=application, chat_id=chat_id, user_id=user_id)
57     self._message_id: Optional[int] = None
58
59 @property
60 def bot_user_ids(self) -> Set[int]:
61     """Custom shortcut to access a value stored in the bot_data dict"""
62     return self.bot_data.setdefault("user_ids", set())
63
64 @property
65 def message_clicks(self) -> Optional[int]:
66     """Access the number of clicks for the message this context object was built
67     ↪ for."""
68     if self._message_id:
69         return self.chat_data.clicks_per_message[self._message_id]
70     return None
71
72 @message_clicks.setter
73 def message_clicks(self, value: int) -> None:
74     """Allow to change the count"""
75     if not self._message_id:
76         raise RuntimeError("There is no message associated with this context
77         ↪ object.")
78     self.chat_data.clicks_per_message[self._message_id] = value
79
80 @classmethod
81 def from_update(cls, update: object, application: "Application") -> "CustomContext
82 ↪ ":
83     """Override from_update to set _message_id."""
84     # Make sure to call super()
85     context = super().from_update(update, application)
86
87     if context.chat_data and isinstance(update, Update) and update.effective_
88     ↪ message:
89         # pylint: disable=protected-access
90         context._message_id = update.effective_message.message_id
91
92         # Remember to return the object
93         return context
94
95 async def start(update: Update, context: CustomContext) -> None:
96     """Display a message with a button."""
97     await update.message.reply_html(
98         "This button was clicked <i>0</i> times.",
99         reply_markup=InlineKeyboardMarkup.from_button(
100             InlineKeyboardButton(text="Click me!", callback_data="button")
101         ),
102     )

```

(continues on next page)

(continued from previous page)

```

101
102 async def count_click(update: Update, context: CustomContext) -> None:
103     """Update the click count for the message."""
104     context.message_clicks += 1
105     await update.callback_query.answer()
106     await update.effective_message.edit_text(
107         f"This button was clicked <i>{context.message_clicks}</i> times.",
108         reply_markup=InlineKeyboardMarkup.from_button(
109             InlineKeyboardButton(text="Click me!", callback_data="button")
110         ),
111         parse_mode=ParseMode.HTML,
112     )
113
114
115 async def print_users(update: Update, context: CustomContext) -> None:
116     """Show which users have been using this bot."""
117     await update.message.reply_text(
118         f"The following user IDs have used this bot: {' '.join(map(str, context.bot_
119 ↪ user_ids))}"
120     )
121
122
123 async def track_users(update: Update, context: CustomContext) -> None:
124     """Store the user id of the incoming update, if any."""
125     if update.effective_user:
126         context.bot_user_ids.add(update.effective_user.id)
127
128
129 def main() -> None:
130     """Run the bot."""
131     context_types = ContextTypes(context=CustomContext, chat_data=ChatData)
132     application = Application.builder().token("TOKEN").context_types(context_types).
133 ↪ build()
134
135     # run track_users in its own group to not interfere with the user handlers
136     application.add_handler(TypeHandler(Update, track_users), group=-1)
137     application.add_handler(CommandHandler("start", start))
138     application.add_handler(CallbackQueryHandler(count_click))
139     application.add_handler(CommandHandler("print_users", print_users))
140
141     application.run_polling(allowed_updates=Update.ALL_TYPES)
142
143
144 if __name__ == "__main__":
145     main()

```

`conversationbot.py`

```

1  #!/usr/bin/env python
2  # pylint: disable=unused-argument
3  # This program is dedicated to the public domain under the CC0 license.
4
5  """
6  First, a few callback functions are defined. Then, those functions are passed to
7  the Application and registered at their respective places.
8  Then, the bot is started and runs until we press Ctrl-C on the command line.
9
10 Usage:
11 Example of a bot-user conversation using ConversationHandler.
12 Send /start to initiate the conversation.
13 Press Ctrl-C on the command line or send a signal to the process to stop the
14 bot.
15 """
16
17 import logging
18
19 from telegram import ReplyKeyboardMarkup, ReplyKeyboardRemove, Update
20 from telegram.ext import (
21     Application,
22     CommandHandler,
23     ContextTypes,
24     ConversationHandler,
25     MessageHandler,
26     filters,
27 )
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 GENDER, PHOTO, LOCATION, BIO = range(4)
39
40
41 async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
42     """Starts the conversation and asks the user about their gender."""
43     reply_keyboard = [["Boy", "Girl", "Other"]]
44
45     await update.message.reply_text(
46         "Hi! My name is Professor Bot. I will hold a conversation with you. "
47         "Send /cancel to stop talking to me.\n\n"
48         "Are you a boy or a girl?",
49         reply_markup=ReplyKeyboardMarkup(
50             reply_keyboard, one_time_keyboard=True, input_field_placeholder="Boy or
↪ Girl?"
51         ),
52     )
53
54     return GENDER

```

(continues on next page)

(continued from previous page)

```

55
56
57 async def gender(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
58     """Stores the selected gender and asks for a photo."""
59     user = update.message.from_user
60     logger.info("Gender of %s: %s", user.first_name, update.message.text)
61     await update.message.reply_text(
62         "I see! Please send me a photo of yourself, "
63         "so I know what you look like, or send /skip if you don't want to.",
64         reply_markup=ReplyKeyboardRemove(),
65     )
66
67     return PHOTO
68
69
70 async def photo(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
71     """Stores the photo and asks for a location."""
72     user = update.message.from_user
73     photo_file = await update.message.photo[-1].get_file()
74     await photo_file.download_to_drive("user_photo.jpg")
75     logger.info("Photo of %s: %s", user.first_name, "user_photo.jpg")
76     await update.message.reply_text(
77         "Gorgeous! Now, send me your location please, or send /skip if you don't want_
↪to."
78     )
79
80     return LOCATION
81
82
83 async def skip_photo(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
84     """Skips the photo and asks for a location."""
85     user = update.message.from_user
86     logger.info("User %s did not send a photo.", user.first_name)
87     await update.message.reply_text(
88         "I bet you look great! Now, send me your location please, or send /skip."
89     )
90
91     return LOCATION
92
93
94 async def location(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
95     """Stores the location and asks for some info about the user."""
96     user = update.message.from_user
97     user_location = update.message.location
98     logger.info(
99         "Location of %s: %f / %f", user.first_name, user_location.latitude, user_
↪location.longitude
100     )
101     await update.message.reply_text(
102         "Maybe I can visit you sometime! At last, tell me something about yourself."
103     )
104
105     return BIO
106
107
108 async def skip_location(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:

```

(continues on next page)

(continued from previous page)

```

109     """Skips the location and asks for info about the user."""
110     user = update.message.from_user
111     logger.info("User %s did not send a location.", user.first_name)
112     await update.message.reply_text(
113         "You seem a bit paranoid! At last, tell me something about yourself."
114     )
115
116     return BIO
117
118
119 async def bio(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
120     """Stores the info about the user and ends the conversation."""
121     user = update.message.from_user
122     logger.info("Bio of %s: %s", user.first_name, update.message.text)
123     await update.message.reply_text("Thank you! I hope we can talk again some day.")
124
125     return ConversationHandler.END
126
127
128 async def cancel(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
129     """Cancels and ends the conversation."""
130     user = update.message.from_user
131     logger.info("User %s canceled the conversation.", user.first_name)
132     await update.message.reply_text(
133         "Bye! I hope we can talk again some day.", reply_markup=ReplyKeyboardRemove()
134     )
135
136     return ConversationHandler.END
137
138
139 def main() -> None:
140     """Run the bot."""
141     # Create the Application and pass it your bot's token.
142     application = Application.builder().token("TOKEN").build()
143
144     # Add conversation handler with the states GENDER, PHOTO, LOCATION and BIO
145     conv_handler = ConversationHandler(
146         entry_points=[CommandHandler("start", start)],
147         states={
148             GENDER: [MessageHandler(filters.Regex("^Boy|Girl|Other$"), gender)],
149             PHOTO: [MessageHandler(filters.PHOTO, photo), CommandHandler("skip", skip_
150 ↪photo)],
151             LOCATION: [
152                 MessageHandler(filters.LOCATION, location),
153                 CommandHandler("skip", skip_location),
154             ],
155             BIO: [MessageHandler(filters.TEXT & ~filters.COMMAND, bio)],
156         },
157         fallbacks=[CommandHandler("cancel", cancel)],
158     )
159
160     application.add_handler(conv_handler)
161
162     # Run the bot until the user presses Ctrl-C
163     application.run_polling(allowed_updates=Update.ALL_TYPES)

```

(continues on next page)

(continued from previous page)

```

164
165 if __name__ == "__main__":
166     main()

```

State Diagram

conversationbot2.py

```

1  #!/usr/bin/env python
2  # pylint: disable=unused-argument
3  # This program is dedicated to the public domain under the CC0 license.
4
5  """
6  First, a few callback functions are defined. Then, those functions are passed to
7  the Application and registered at their respective places.
8  Then, the bot is started and runs until we press Ctrl-C on the command line.
9
10 Usage:
11 Example of a bot-user conversation using ConversationHandler.
12 Send /start to initiate the conversation.
13 Press Ctrl-C on the command line or send a signal to the process to stop the
14 bot.
15 """
16
17 import logging
18 from typing import Dict
19
20 from telegram import ReplyKeyboardMarkup, ReplyKeyboardRemove, Update
21 from telegram.ext import (
22     Application,
23     CommandHandler,
24     ContextTypes,
25     ConversationHandler,
26     MessageHandler,
27     filters,
28 )
29
30 # Enable logging
31 logging.basicConfig(
32     format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
33 )
34 # set higher logging level for httpx to avoid all GET and POST requests being logged
35 logging.getLogger("httpx").setLevel(logging.WARNING)
36
37 logger = logging.getLogger(__name__)
38
39 CHOOSING, TYPING_REPLY, TYPING_CHOICE = range(3)
40
41 reply_keyboard = [
42     ["Age", "Favourite colour"],
43     ["Number of siblings", "Something else..."],
44     ["Done"],
45 ]
46 markup = ReplyKeyboardMarkup(reply_keyboard, one_time_keyboard=True)

```

(continues on next page)

(continued from previous page)

```

47
48
49 def facts_to_str(user_data: Dict[str, str]) -> str:
50     """Helper function for formatting the gathered user info."""
51     facts = [f"{key} - {value}" for key, value in user_data.items()]
52     return "\n".join(facts).join(["\n", "\n"])
53
54
55 async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
56     """Start the conversation and ask user for input."""
57     await update.message.reply_text(
58         "Hi! My name is Doctor Botter. I will hold a more complex conversation with
↪you. "
59         "Why don't you tell me something about yourself?",
60         reply_markup=markup,
61     )
62
63     return CHOOSING
64
65
66 async def regular_choice(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
67     """Ask the user for info about the selected predefined choice."""
68     text = update.message.text
69     context.user_data["choice"] = text
70     await update.message.reply_text(f"Your {text.lower()}? Yes, I would love to hear
↪about that!")
71
72     return TYPING_REPLY
73
74
75 async def custom_choice(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
76     """Ask the user for a description of a custom category."""
77     await update.message.reply_text(
78         'Alright, please send me the category first, for example "Most impressive
↪skill"'
79     )
80
81     return TYPING_CHOICE
82
83
84 async def received_information(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
↪int:
85     """Store info provided by user and ask for the next category."""
86     user_data = context.user_data
87     text = update.message.text
88     category = user_data["choice"]
89     user_data[category] = text
90     del user_data["choice"]
91
92     await update.message.reply_text(
93         "Neat! Just so you know, this is what you already told me:"
94         f"{facts_to_str(user_data)}You can tell me more, or change your opinion"
95         " on something.",
96         reply_markup=markup,
97     )
98

```

(continues on next page)

(continued from previous page)

```

99     return CHOOSING
100
101
102 async def done(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
103     """Display the gathered info and end the conversation."""
104     user_data = context.user_data
105     if "choice" in user_data:
106         del user_data["choice"]
107
108     await update.message.reply_text(
109         f"I learned these facts about you: {facts_to_str(user_data)}Until next time!",
110         reply_markup=ReplyKeyboardRemove(),
111     )
112
113     user_data.clear()
114     return ConversationHandler.END
115
116
117 def main() -> None:
118     """Run the bot."""
119     # Create the Application and pass it your bot's token.
120     application = Application.builder().token("TOKEN").build()
121
122     # Add conversation handler with the states CHOOSING, TYPING_CHOICE and TYPING_
↪REPLY
123     conv_handler = ConversationHandler(
124         entry_points=[CommandHandler("start", start)],
125         states={
126             CHOOSING: [
127                 MessageHandler(
128                     filters.Regex("^Age|Favourite colour|Number of siblings)$"), ↪
↪regular_choice
129                 ),
130                 MessageHandler(filters.Regex("^Something else...$"), custom_choice),
131             ],
132             TYPING_CHOICE: [
133                 MessageHandler(
134                     filters.TEXT & ~(filters.COMMAND | filters.Regex("^Done$")), ↪
↪regular_choice
135                 )
136             ],
137             TYPING_REPLY: [
138                 MessageHandler(
139                     filters.TEXT & ~(filters.COMMAND | filters.Regex("^Done$")),
140                     received_information,
141                 )
142             ],
143         },
144         fallbacks=[MessageHandler(filters.Regex("^Done$"), done)],
145     )
146
147     application.add_handler(conv_handler)
148
149     # Run the bot until the user presses Ctrl-C
150     application.run_polling(allowed_updates=Update.ALL_TYPES)
151

```

(continues on next page)

(continued from previous page)

```

152
153 if __name__ == "__main__":
154     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 existing 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!

```

1  #!/usr/bin/env python
2  # This program is dedicated to the public domain under the CC0 license.
3  # pylint: disable=import-error,unused-argument
4  """
5  Simple example of a bot that uses a custom webhook setup and handles custom updates.
6  For the custom webhook setup, the libraries `starlette` and `uvicorn` are used. Please
7  ↪install
8  them as `pip install starlette~=0.20.0 uvicorn~=0.23.2`.
9  Note that any other `asyncio` based web server framework can be used for a custom
10 ↪webhook setup
11 just as well.
12
13 Usage:
14 Set bot Token, URL, admin CHAT_ID and PORT after the imports.
15 You may also need to change the `listen` value in the uvicorn configuration to match
16 ↪your setup.
17 Press Ctrl-C on the command line or send a signal to the process to stop the bot.
18 """
19
20 import asyncio
21 import html
22 import logging
23 from dataclasses import dataclass
24 from http import HTTPStatus
25
26 import uvicorn
27 from starlette.applications import Starlette
28 from starlette.requests import Request
29 from starlette.responses import PlainTextResponse, Response
30 from starlette.routing import Route
31
32 from telegram import Update

```

(continues on next page)

(continued from previous page)

```

29 from telegram.constants import ParseMode
30 from telegram.ext import (
31     Application,
32     CallbackContext,
33     CommandHandler,
34     ContextTypes,
35     ExtBot,
36     TypeHandler,
37 )
38
39 # Enable logging
40 logging.basicConfig(
41     format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
42 )
43 # set higher logging level for httpx to avoid all GET and POST requests being logged
44 logging.getLogger("httpx").setLevel(logging.WARNING)
45
46 logger = logging.getLogger(__name__)
47
48 # Define configuration constants
49 URL = "https://domain.tld"
50 ADMIN_CHAT_ID = 123456
51 PORT = 8000
52 TOKEN = "123:ABC" # nsec B105
53
54
55 @dataclass
56 class WebhookUpdate:
57     """Simple dataclass to wrap a custom update type"""
58
59     user_id: int
60     payload: str
61
62
63 class CustomContext(CallbackContext[ExtBot, dict, dict, dict]):
64     """
65     Custom CallbackContext class that makes `user_data` available for updates of type
66     `WebhookUpdate`.
67     """
68
69     @classmethod
70     def from_update(
71         cls,
72         update: object,
73         application: "Application",
74     ) -> "CustomContext":
75         if isinstance(update, WebhookUpdate):
76             return cls(application=application, user_id=update.user_id)
77         return super().from_update(update, application)
78
79
80 async def start(update: Update, context: CustomContext) -> None:
81     """Display a message with instructions on how to use this bot."""
82     payload_url = html.escape(f"{URL}/submitpayload?user_id=<your user id>&payload=
83     ↪<payload>")
84     text = (

```

(continues on next page)

(continued from previous page)

```

84         f"To check if the bot is still running, call <code>{URL}/healthcheck</code>.\n\n"
85         f"To post a custom update, call <code>{payload_url}</code>."
86     )
87     await update.message.reply_html(text=text)
88
89
90 async def webhook_update(update: WebhookUpdate, context: CustomContext) -> None:
91     """Handle custom updates."""
92     chat_member = await context.bot.get_chat_member(chat_id=update.user_id, user_
93     id=update.user_id)
94     payloads = context.user_data.setdefault("payloads", [])
95     payloads.append(update.payload)
96     combined_payloads = "<code>\n• <code>".join(payloads)
97     text = (
98         f"The user {chat_member.user.mention_html()} has sent a new payload. "
99         f"So far they have sent the following payloads: \n\n• <code>{combined_
100     payloads}</code>"
101     )
102     await context.bot.send_message(chat_id=ADMIN_CHAT_ID, text=text, parse_
103     mode=ParseMode.HTML)
104
105
106 async def main() -> None:
107     """Set up PTB application and a web application for handling the incoming
108     requests."""
109     context_types = ContextTypes(context=CustomContext)
110     # Here we set updater to None because we want our custom webhook server to handle
111     # the updates
112     # and hence we don't need an Updater instance
113     application = (
114         Application.builder().token(TOKEN).updater(None).context_types(context_types).
115         build()
116     )
117
118     # register handlers
119     application.add_handler(CommandHandler("start", start))
120     application.add_handler(TypeHandler(type=WebhookUpdate, callback=webhook_update))
121
122     # Pass webhook settings to telegram
123     await application.bot.set_webhook(url=f"{URL}/telegram", allowed_updates=Update.
124     ALL_TYPES)
125
126     # Set up webserver
127     async def telegram(request: Request) -> Response:
128         """Handle incoming Telegram updates by putting them into the `update_queue`"""
129         await application.update_queue.put(
130             Update.de_json(data=await request.json(), bot=application.bot)
131         )
132         return Response()
133
134     async def custom_updates(request: Request) -> PlainTextResponse:
135         """
136         Handle incoming webhook updates by also putting them into the `update_queue`
137         if
138         the required parameters were passed correctly.

```

(continues on next page)

(continued from previous page)

```

131     """
132     try:
133         user_id = int(request.query_params["user_id"])
134         payload = request.query_params["payload"]
135     except KeyError:
136         return PlainTextResponse(
137             status_code=HTTPStatus.BAD_REQUEST,
138             content="Please pass both `user_id` and `payload` as query parameters.
139         ↪",
140         )
141     except ValueError:
142         return PlainTextResponse(
143             status_code=HTTPStatus.BAD_REQUEST,
144             content="The `user_id` must be a string!",
145         )
146
147     await application.update_queue.put(WebhookUpdate(user_id=user_id,
148 ↪payload=payload))
149     return PlainTextResponse("Thank you for the submission! It's being forwarded.
150 ↪")
151
152     async def health(_: Request) -> PlainTextResponse:
153         """For the health endpoint, reply with a simple plain text message."""
154         return PlainTextResponse(content="The bot is still running fine :)")
155
156     starlette_app = Starlette(
157         routes=[
158             Route("/telegram", telegram, methods=["POST"]),
159             Route("/healthcheck", health, methods=["GET"]),
160             Route("/submitpayload", custom_updates, methods=["POST", "GET"]),
161         ]
162     )
163
164     webserver = uvicorn.Server(
165         config=uvicorn.Config(
166             app=starlette_app,
167             port=PORT,
168             use_colors=False,
169             host="127.0.0.1",
170         )
171     )
172
173     # Run application and webserver together
174     async with application:
175         await application.start()
176         await webserver.serve()
177         await application.stop()
178
179 if __name__ == "__main__":
180     asyncio.run(main())

```

```

1  #!/usr/bin/env python
2  # This program is dedicated to the public domain under the CC0 license.
3  # pylint: disable=import-error,unused-argument
4  """
5  Simple example of a bot that uses a custom webhook setup and handles custom updates.

```

(continues on next page)

(continued from previous page)

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,
)

# 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" # nsec B105

@dataclass
class WebhookUpdate:
    """Simple dataclass to wrap a custom update type"""

    user_id: int
    payload: str
```

(continues on next page)

(continued from previous page)

```

59
60
61 class CustomContext(CallbackContext[ExtBot, dict, dict]):
62     """
63     Custom CallbackContext class that makes `user_data` available for updates of type
64     `WebhookUpdate`.
65     """
66
67     @classmethod
68     def from_update(
69         cls,
70         update: object,
71         application: "Application",
72     ) -> "CustomContext":
73         if isinstance(update, WebhookUpdate):
74             return cls(application=application, user_id=update.user_id)
75         return super().from_update(update, application)
76
77
78 async def start(update: Update, context: CustomContext) -> None:
79     """Display a message with instructions on how to use this bot."""
80     payload_url = html.escape(f"{URL}/submitpayload?user_id=<your user id>&payload=
81     <payload>")
82     text = (
83         f"To check if the bot is still running, call <code>{URL}/healthcheck</code>.\n\n"
84         f"To post a custom update, call <code>{payload_url}</code>."
85     )
86     await update.message.reply_html(text=text)
87
88 async def webhook_update(update: WebhookUpdate, context: CustomContext) -> None:
89     """Handle custom updates."""
90     chat_member = await context.bot.get_chat_member(chat_id=update.user_id, user_
91     id=update.user_id)
92     payloads = context.user_data.setdefault("payloads", [])
93     payloads.append(update.payload)
94     combined_payloads = "</code>\n• <code>".join(payloads)
95     text = (
96         f"The user {chat_member.user.mention_html()} has sent a new payload. "
97         f"So far they have sent the following payloads: \n\n• <code>{combined_
98     payloads}</code>"
99     )
100     await context.bot.send_message(chat_id=ADMIN_CHAT_ID, text=text, parse_
101     mode=ParseMode.HTML)
102
103 async def main() -> None:
104     """Set up PTB application and a web application for handling the incoming_
105     requests."""
106     context_types = ContextTypes(context=CustomContext)
107     # Here we set updater to None because we want our custom webhook server to handle_
108     the updates
109     # and hence we don't need an Updater instance
110     application = (
111         Application.builder().token(TOKEN).updater(None).context_types(context_types).

```

(continues on next page)

(continued from previous page)

```

108     ↪ build()
109     )
110
111     # register handlers
112     application.add_handler(CommandHandler("start", start))
113     application.add_handler(TypeHandler(type=WebhookUpdate, callback=webhook_update))
114
115     # Pass webhook settings to telegram
116     ↪ await application.bot.set_webhook(url=f"{URL}/telegram", allowed_updates=Update.
117     ↪ ALL_TYPES)
118
119     # Set up webserver
120     flask_app = Flask(__name__)
121
122     @flask_app.post("/telegram") # type: ignore[misc]
123     ↪ async def telegram() -> Response:
124         """Handle incoming Telegram updates by putting them into the `update_queue`"""
125         ↪ await application.update_queue.put(Update.de_json(data=request.json, ↪
126         ↪ bot=application.bot))
127         ↪ return Response(status=HTTPStatus.OK)
128
129     @flask_app.route("/submitpayload", methods=["GET", "POST"]) # type: ignore[misc]
130     ↪ async def custom_updates() -> Response:
131         """
132         ↪ Handle incoming webhook updates by also putting them into the `update_queue` ↪
133         ↪ the required parameters were passed correctly.
134         ↪ """
135         ↪ try:
136             ↪ user_id = int(request.args["user_id"])
137             ↪ payload = request.args["payload"]
138             ↪ except KeyError:
139                 ↪ abort(
140                     ↪ HTTPStatus.BAD_REQUEST,
141                     ↪ "Please pass both `user_id` and `payload` as query parameters.",
142                 )
143             ↪ except ValueError:
144                 ↪ abort(HTTPStatus.BAD_REQUEST, "The `user_id` must be a string!")
145
146             ↪ await application.update_queue.put(WebhookUpdate(user_id=user_id, ↪
147             ↪ payload=payload))
148             ↪ return Response(status=HTTPStatus.OK)
149
150     @flask_app.get("/healthcheck") # type: ignore[misc]
151     ↪ async def health() -> Response:
152         """For the health endpoint, reply with a simple plain text message."""
153         ↪ response = make_response("The bot is still running fine :)", HTTPStatus.OK)
154         ↪ response.mimetype = "text/plain"
155         ↪ return response
156
157     webserver = uvicorn.Server(
158         ↪ config=uvicorn.Config(
159             ↪ app=WsgiToAsgi(flask_app),
160             ↪ port=PORT,
161             ↪ use_colors=False,
162             ↪ host="127.0.0.1",

```

(continues on next page)

(continued from previous page)

```

159     )
160 )
161
162 # Run application and webserver together
163 async with application:
164     await application.start()
165     await webserver.serve()
166     await application.stop()
167
168
169 if __name__ == "__main__":
170     asyncio.run(main())

```

```

1  #!/usr/bin/env python
2  # This program is dedicated to the public domain under the CC0 license.
3  # pylint: disable=import-error,unused-argument
4  """
5  Simple example of a bot that uses a custom webhook setup and handles custom updates.
6  For the custom webhook setup, the libraries `quart` and `uvicorn` are used. Please
7  install them as `pip install quart~=0.18.4 uvicorn~=0.23.2`.
8  Note that any other `asyncio` based web server framework can be used for a custom
9  ↪webhook setup
10 just as well.
11
12 Usage:
13 Set bot Token, URL, admin CHAT_ID and PORT after the imports.
14 You may also need to change the `listen` value in the uvicorn configuration to match
15 ↪your setup.
16 Press Ctrl-C on the command line or send a signal to the process to stop the bot.
17 """
18 import asyncio
19 import html
20 import logging
21 from dataclasses import dataclass
22 from http import HTTPStatus
23
24 import uvicorn
25 from quart import Quart, Response, abort, make_response, request
26
27 from telegram import Update
28 from telegram.constants import ParseMode
29 from telegram.ext import (
30     Application,
31     CallbackContext,
32     CommandHandler,
33     ContextTypes,
34     ExtBot,
35     TypeHandler,
36 )
37
38 # Enable logging
39 logging.basicConfig(
40     format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
41 )
42 # set higher logging level for httpx to avoid all GET and POST requests being logged
43 logging.getLogger("httpx").setLevel(logging.WARNING)

```

(continues on next page)

(continued from previous page)

```

42 logger = logging.getLogger(__name__)
43
44 # Define configuration constants
45 URL = "https://domain.tld"
46 ADMIN_CHAT_ID = 123456
47 PORT = 8000
48 TOKEN = "123:ABC" # nsec B105
49
50
51
52 @dataclass
53 class WebhookUpdate:
54     """Simple dataclass to wrap a custom update type"""
55
56     user_id: int
57     payload: str
58
59
60 class CustomContext(CallbackContext[ExtBot, dict, dict, dict]):
61     """
62     Custom CallbackContext class that makes `user_data` available for updates of type
63     `WebhookUpdate`.
64     """
65
66     @classmethod
67     def from_update(
68         cls,
69         update: object,
70         application: "Application",
71     ) -> "CustomContext":
72         if isinstance(update, WebhookUpdate):
73             return cls(application=application, user_id=update.user_id)
74         return super().from_update(update, application)
75
76
77 async def start(update: Update, context: CustomContext) -> None:
78     """Display a message with instructions on how to use this bot."""
79     payload_url = html.escape(f"{URL}/submitpayload?user_id=<your user id>&payload=
80     ↪<payload>")
81     text = (
82         f"To check if the bot is still running, call <code>{URL}/healthcheck</code>.\n
83     ↪\n\n"
84         f"To post a custom update, call <code>{payload_url}</code>."
85     )
86     await update.message.reply_html(text=text)
87
88
89 async def webhook_update(update: WebhookUpdate, context: CustomContext) -> None:
90     """Handle custom updates."""
91     chat_member = await context.bot.get_chat_member(chat_id=update.user_id, user_
92     ↪id=update.user_id)
93     payloads = context.user_data.setdefault("payloads", [])
94     payloads.append(update.payload)
95     combined_payloads = "<code>\n• <code>".join(payloads)
96     text = (
97         f"The user {chat_member.user.mention_html()} has sent a new payload. "

```

(continues on next page)

(continued from previous page)

```

95         f"So far they have sent the following payloads: \n\n• <code>{combined_
↪ payloads}</code>"
96     )
97     await context.bot.send_message(chat_id=ADMIN_CHAT_ID, text=text, parse_
↪ mode=ParseMode.HTML)
98
99
100 async def main() -> None:
101     """Set up PTB application and a web application for handling the incoming_
↪ requests."""
102     context_types = ContextTypes(context=CustomContext)
103     # Here we set updater to None because we want our custom webhook server to handle_
↪ the updates
104     # and hence we don't need an Updater instance
105     application = (
106         Application.builder().token(TOKEN).updater(None).context_types(context_types).
↪ build()
107     )
108
109     # register handlers
110     application.add_handler(CommandHandler("start", start))
111     application.add_handler(TypeHandler(type=WebhookUpdate, callback=webhook_update))
112
113     # Pass webhook settings to telegram
114     await application.bot.set_webhook(url=f"{URL}/telegram", allowed_updates=Update.
↪ ALL_TYPES)
115
116     # Set up webserver
117     quart_app = Quart(__name__)
118
119     @quart_app.post("/telegram") # type: ignore[misc]
120     async def telegram() -> Response:
121         """Handle incoming Telegram updates by putting them into the `update_queue`"""
122         await application.update_queue.put(
123             Update.de_json(data=await request.get_json(), bot=application.bot)
124         )
125         return Response(status=HTTPStatus.OK)
126
127     @quart_app.route("/submitpayload", methods=["GET", "POST"]) # type: ignore[misc]
128     async def custom_updates() -> Response:
129         """
130         Handle incoming webhook updates by also putting them into the `update_queue`_
↪ if
131         the required parameters were passed correctly.
132         """
133         try:
134             user_id = int(request.args["user_id"])
135             payload = request.args["payload"]
136         except KeyError:
137             abort(
138                 HTTPStatus.BAD_REQUEST,
139                 "Please pass both `user_id` and `payload` as query parameters.",
140             )
141         except ValueError:
142             abort(HTTPStatus.BAD_REQUEST, "The `user_id` must be a string!")
143

```

(continues on next page)

(continued from previous page)

```

144     await application.update_queue.put(WebhookUpdate(user_id=user_id,
↳ payload=payload))
145     return Response(status=HTTPStatus.OK)
146
147     @quart_app.get("/healthcheck") # type: ignore[misc]
148     async def health() -> Response:
149         """For the health endpoint, reply with a simple plain text message."""
150         response = await make_response("The bot is still running fine :)", HTTPStatus.
↳ OK)
151         response.mimetype = "text/plain"
152         return response
153
154     webserver = uvicorn.Server(
155         config=uvicorn.Config(
156             app=quart_app,
157             port=PORT,
158             use_colors=False,
159             host="127.0.0.1",
160         )
161     )
162
163     # Run application and webserver together
164     async with application:
165         await application.start()
166         await webserver.serve()
167         await application.stop()
168
169
170 if __name__ == "__main__":
171     asyncio.run(main())

```

```

1  #!/usr/bin/env python
2  # This program is dedicated to the public domain under the CC0 license.
3  # pylint: disable=import-error,unused-argument
4  """
5  Simple example of a bot that uses a custom webhook setup and handles custom updates.
6  For the custom webhook setup, the libraries `Django` and `uvicorn` are used. Please
7  install them as `pip install Django~=4.2.4 uvicorn~=0.23.2`.
8  Note that any other `asyncio` based web server framework can be used for a custom
↳ webhook setup
9  just as well.
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  import asyncio
17  import html
18  import json
19  import logging
20  from dataclasses import dataclass
21  from uuid import uuid4
22
23  import uvicorn

```

(continues on next page)

(continued from previous page)

```

24 from django.conf import settings
25 from django.core.asgi import get_asgi_application
26 from django.http import HttpRequest, HttpResponse, HttpResponseBadRequest
27 from django.urls import path
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" # nsec 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]):
65     """
66     Custom CallbackContext class that makes `user_data` available for updates of type
67     `WebhookUpdate`.
68     """
69
70     @classmethod
71     def from_update(
72         cls,
73         update: object,
74         application: "Application",
75     ) -> "CustomContext":
76         if isinstance(update, WebhookUpdate):
77             return cls(application=application, user_id=update.user_id)
78         return super().from_update(update, application)
79

```

(continues on next page)

(continued from previous page)

```

80
81 async def start(update: Update, context: CustomContext) -> None:
82     """Display a message with instructions on how to use this bot."""
83     payload_url = html.escape(f"{URL}/submitpayload?user_id=<your user id>&payload=
84     ↪<payload>")
85     text = (
86         f"To check if the bot is still running, call <code>{URL}/healthcheck</code>.\
87     ↪\n\n"
88         f"To post a custom update, call <code>{payload_url}</code>."
89     )
90     await update.message.reply_html(text=text)
91
92 async def webhook_update(update: WebhookUpdate, context: CustomContext) -> None:
93     """Handle custom updates."""
94     chat_member = await context.bot.get_chat_member(chat_id=update.user_id, user_
95     ↪id=update.user_id)
96     payloads = context.user_data.setdefault("payloads", [])
97     payloads.append(update.payload)
98     combined_payloads = "</code>\n• <code>".join(payloads)
99     text = (
100         f"The user {chat_member.user.mention_html()} has sent a new payload. "
101         ↪f"So far they have sent the following payloads: \n\n• <code>{combined_
102         ↪payloads}</code>"
103     )
104     await context.bot.send_message(chat_id=ADMIN_CHAT_ID, text=text, parse_
105     ↪mode=ParseMode.HTML)
106
107 async def telegram(request: HttpRequest) -> HttpResponse:
108     """Handle incoming Telegram updates by putting them into the `update_queue`"""
109     await ptb_application.update_queue.put(
110         Update.de_json(data=json.loads(request.body), bot=ptb_application.bot)
111     )
112     return HttpResponse()
113
114 async def custom_updates(request: HttpRequest) -> HttpResponse:
115     """
116     Handle incoming webhook updates by also putting them into the `update_queue` if
117     the required parameters were passed correctly.
118     """
119     try:
120         user_id = int(request.GET["user_id"])
121         payload = request.GET["payload"]
122     except KeyError:
123         return HttpResponseBadRequest(
124             "Please pass both `user_id` and `payload` as query parameters.",
125         )
126     except ValueError:
127         return HttpResponseBadRequest("The `user_id` must be a string!")
128
129     await ptb_application.update_queue.put(WebhookUpdate(user_id=user_id,
130     ↪payload=payload))
131     return HttpResponse()

```

(continues on next page)

(continued from previous page)

```

130
131 async def health(_: HttpRequest) -> HttpResponse:
132     """For the health endpoint, reply with a simple plain text message."""
133     return HttpResponse("The bot is still running fine :)")
134
135
136 # Set up PTB application and a web application for handling the incoming requests.
137
138 context_types = ContextTypes(context=CustomContext)
139 # Here we set updater to None because we want our custom webhook server to handle the_
140 ↪updates
141 # and hence we don't need an Updater instance
142 ptb_application = (
143     Application.builder().token(TOKEN).updater(None).context_types(context_types).
144     ↪build()
145 )
146
147 # register handlers
148 ptb_application.add_handler(CommandHandler("start", start))
149 ptb_application.add_handler(TypeHandler(type=WebhookUpdate, callback=webhook_update))
150
151 urlpatterns = [
152     path("telegram", telegram, name="Telegram updates"),
153     path("submitpayload", custom_updates, name="custom updates"),
154     path("healthcheck", health, name="health check"),
155 ]
156
157 settings.configure(ROOT_URLCONF=__name__, SECRET_KEY=uuid4().hex)
158
159
160 async def main() -> None:
161     """Finalize configuration and run the applications."""
162     webserver = uvicorn.Server(
163         config=uvicorn.Config(
164             app=get_asgi_application(),
165             port=PORT,
166             use_colors=False,
167             host="127.0.0.1",
168         )
169     )
170
171     # Pass webhook settings to telegram
172     await ptb_application.bot.set_webhook(url=f"{URL}/telegram", allowed_
173     ↪updates=Update.ALL_TYPES)
174
175     # Run application and webserver together
176     async with ptb_application:
177         await ptb_application.start()
178         await webserver.serve()
179         await ptb_application.stop()
180
181 if __name__ == "__main__":
182     asyncio.run(main())
183

```

deeplinking.py

```

1  #!/usr/bin/env python
2  # pylint: disable=unused-argument
3  # This program is dedicated to the public domain under the CC0 license.
4
5  """Bot that explains Telegram's "Deep Linking Parameters" functionality.
6
7  This program is dedicated to the public domain under the CC0 license.
8
9  This Bot uses the Application class to handle the bot.
10
11 First, a few handler functions are defined. Then, those functions are passed to
12 the Application and registered at their respective places.
13 Then, the bot is started and runs until we press Ctrl-C on the command line.
14
15 Usage:
16 Deep Linking example. Send /start to get the link.
17 Press Ctrl-C on the command line or send a signal to the process to stop the
18 bot.
19 """
20
21 import logging
22
23 from telegram import InlineKeyboardButton, InlineKeyboardMarkup, Update, helpers
24 from telegram.constants import ParseMode
25 from telegram.ext import Application, CallbackQueryHandler, CommandHandler, ContextTypes, filters
26
27 # Enable logging
28 logging.basicConfig(
29     format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
30 )
31
32 # set higher logging level for httpx to avoid all GET and POST requests being logged
33 logging.getLogger("httpx").setLevel(logging.WARNING)
34
35 logger = logging.getLogger(__name__)
36
37 # Define constants that will allow us to reuse the deep-linking parameters.
38 CHECK_THIS_OUT = "check-this-out"
39 USING_ENTITIES = "using-entities-here"
40 USING_KEYBOARD = "using-keyboard-here"
41 SO_COOL = "so-cool"
42
43 # Callback data to pass in 3rd level deep-linking
44 KEYBOARD_CALLBACKDATA = "keyboard-callback-data"
45
46
47 async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
48     """Send a deep-linked URL when the command /start is issued."""
49     bot = context.bot
50     url = helpers.create_deep_linked_url(bot.username, CHECK_THIS_OUT, group=True)
51     text = "Feel free to tell your friends about it:\n\n" + url
52     await update.message.reply_text(text)
53
54

```

(continues on next page)

(continued from previous page)

```

55 async def deep_linked_level_1(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
    None:
56     """Reached through the CHECK_THIS_OUT payload"""
57     bot = context.bot
58     url = helpers.create_deep_linked_url(bot.username, SO_COOL)
59     text = (
60         "Awesome, you just accessed hidden functionality! Now let's get back to the
    private chat."
61     )
62     keyboard = InlineKeyboardMarkup.from_button(
63         InlineKeyboardButton(text="Continue here!", url=url)
64     )
65     await update.message.reply_text(text, reply_markup=keyboard)
66
67
68 async def deep_linked_level_2(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
    None:
69     """Reached through the SO_COOL payload"""
70     bot = context.bot
71     url = helpers.create_deep_linked_url(bot.username, USING_ENTITIES)
72     text = f'You can also mask the deep-linked URLs as links: <a href="{url}"> CLICK
    HERE</a>.'
73     await update.message.reply_text(text, parse_mode=ParseMode.HTML, disable_web_page_
    preview=True)
74
75
76 async def deep_linked_level_3(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
    None:
77     """Reached through the USING_ENTITIES payload"""
78     await update.message.reply_text(
79         "It is also possible to make deep-linking using InlineKeyboardButtons.",
80         reply_markup=InlineKeyboardMarkup(
81             [[InlineKeyboardButton(text="Like this!", callback_data=KEYBOARD_
    CALLBACKDATA)]]
82         ),
83     )
84
85
86 async def deep_link_level_3_callback(update: Update, context: ContextTypes.DEFAULT_
    TYPE) -> None:
87     """Answers CallbackQuery with deeplinking url."""
88     bot = context.bot
89     url = helpers.create_deep_linked_url(bot.username, USING_KEYBOARD)
90     await update.callback_query.answer(url=url)
91
92
93 async def deep_linked_level_4(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
    None:
94     """Reached through the USING_KEYBOARD payload"""
95     payload = context.args
96     await update.message.reply_text(
97         f"Congratulations! This is as deep as it gets \n\nThe payload was: {payload}"
98     )
99
100
101 def main() -> None:

```

(continues on next page)

(continued from previous page)

```

102 """Start the bot."""
103 # Create the Application and pass it your bot's token.
104 application = Application.builder().token("TOKEN").build()
105
106 # More info on what deep linking actually is (read this first if it's unclear to
107 ↪you):
108 # https://core.telegram.org/bots/features#deep-linking
109
110 # Register a deep-linking handler
111 application.add_handler(
112     CommandHandler("start", deep_linked_level_1, filters.Regex(CHECK_THIS_OUT))
113 )
114
115 # This one works with a textual link instead of an URL
116 ↪application.add_handler(CommandHandler("start", deep_linked_level_2, filters.
117 ↪Regex(SO_COOL)))
118
119 # We can also pass on the deep-linking payload
120 application.add_handler(
121     CommandHandler("start", deep_linked_level_3, filters.Regex(USING_ENTITIES))
122 )
123
124 # Possible with inline keyboard buttons as well
125 application.add_handler(
126     CommandHandler("start", deep_linked_level_4, filters.Regex(USING_KEYBOARD))
127 )
128
129 # register callback handler for inline keyboard button
130 application.add_handler(
131     CallbackQueryHandler(deep_link_level_3_callback, pattern=KEYBOARD_
132 ↪CALLBACKDATA)
133 )
134
135 # Make sure the deep-linking handlers occur *before* the normal /start handler.
136 application.add_handler(CommandHandler("start", start))
137
138 # Run the bot until the user presses Ctrl-C
139 application.run_polling(allowed_updates=Update.ALL_TYPES)
140
141 if __name__ == "__main__":
142     main()

```

echobot.py

```

1 #!/usr/bin/env python
2 # pylint: disable=unused-argument
3 # This program is dedicated to the public domain under the CC0 license.
4
5 """
6 Simple Bot to reply to Telegram messages.
7
8 First, a few handler functions are defined. Then, those functions are passed to
9 the Application and registered at their respective places.
10 Then, the bot is started and runs until we press Ctrl-C on the command line.

```

(continues on next page)

(continued from previous page)

```

11
12 Usage:
13 Basic Echobot example, repeats messages.
14 Press Ctrl-C on the command line or send a signal to the process to stop the
15 bot.
16 """
17
18 import logging
19
20 from telegram import ForceReply, Update
21 from telegram.ext import Application, CommandHandler, ContextTypes, MessageHandler, filters
22
23 # Enable logging
24 logging.basicConfig(
25     format="%(asctime)s - (name)s - (levelname)s - (message)s", level=logging.INFO
26 )
27 # set higher logging level for httpx to avoid all GET and POST requests being logged
28 logging.getLogger("httpx").setLevel(logging.WARNING)
29
30 logger = logging.getLogger(__name__)
31
32
33 # Define a few command handlers. These usually take the two arguments update and
34 # context.
35 async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
36     """Send a message when the command /start is issued."""
37     user = update.effective_user
38     await update.message.reply_html(
39         rf"Hi {user.mention_html()}!",
40         reply_markup=ForceReply(selective=True),
41     )
42
43
44 async def help_command(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
45     """Send a message when the command /help is issued."""
46     await update.message.reply_text("Help!")
47
48
49 async def echo(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
50     """Echo the user message."""
51     await update.message.reply_text(update.message.text)
52
53
54 def main() -> None:
55     """Start the bot."""
56     # Create the Application and pass it your bot's token.
57     application = Application.builder().token("TOKEN").build()
58
59     # on different commands - answer in Telegram
60     application.add_handler(CommandHandler("start", start))
61     application.add_handler(CommandHandler("help", help_command))
62
63     # on non command i.e message - echo the message on Telegram
64     application.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, echo))
65

```

(continues on next page)

(continued from previous page)

```

66     # Run the bot until the user presses Ctrl-C
67     application.run_polling(allowed_updates=Update.ALL_TYPES)
68
69
70 if __name__ == "__main__":
71     main()

```

errorhandlerbot.py

```

1  #!/usr/bin/env python
2  # pylint: disable=unused-argument
3  # This program is dedicated to the public domain under the CC0 license.
4
5  """This is a very simple example on how one could implement a custom error handler."""
6  import html
7  import json
8  import logging
9  import traceback
10
11 from telegram import Update
12 from telegram.constants import ParseMode
13 from telegram.ext import Application, CommandHandler, ContextTypes
14
15 # Enable logging
16 logging.basicConfig(
17     format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
18 )
19 # set higher logging level for httpx to avoid all GET and POST requests being logged
20 logging.getLogger("httpx").setLevel(logging.WARNING)
21
22 logger = logging.getLogger(__name__)
23
24 # This can be your own ID, or one for a developer group/channel.
25 # You can use the /start command of this bot to see your chat id.
26 DEVELOPER_CHAT_ID = 123456789
27
28
29 async def error_handler(update: object, context: ContextTypes.DEFAULT_TYPE) -> None:
30     """Log the error and send a telegram message to notify the developer."""
31     # Log the error before we do anything else, so we can see it even if something
32     ↪ breaks.
33     logger.error("Exception while handling an update:", exc_info=context.error)
34
35     # traceback.format_exception returns the usual python message about an exception,
36     ↪ but as a
37     # list of strings rather than a single string, so we have to join them together.
38     tb_list = traceback.format_exception(None, context.error, context.error.__
39     ↪ traceback__)
40     tb_string = "".join(tb_list)
41
42     # Build the message with some markup and additional information about what
43     ↪ happened.
44     # You might need to add some logic to deal with messages longer than the 4096
45     ↪ character limit.
46     update_str = update.to_dict() if isinstance(update, Update) else str(update)

```

(continues on next page)

(continued from previous page)

```

42     message = (
43         "An exception was raised while handling an update\n"
44         f"<pre>update = {html.escape(json.dumps(update_str, indent=2, ensure_
↪ascii=False))}"
45         "</pre>\n\n"
46         f"<pre>context.chat_data = {html.escape(str(context.chat_data))}</pre>\n\n"
47         f"<pre>context.user_data = {html.escape(str(context.user_data))}</pre>\n\n"
48         f"<pre>{html.escape(tb_string)}</pre>"
49     )
50
51     # Finally, send the message
52     await context.bot.send_message(
53         chat_id=DEVELOPER_CHAT_ID, text=message, parse_mode=ParseMode.HTML
54     )
55
56
57 async def bad_command(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
58     """Raise an error to trigger the error handler."""
59     await context.bot.wrong_method_name() # type: ignore[attr-defined]
60
61
62 async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
63     """Displays info on how to trigger an error."""
64     await update.effective_message.reply_html(
65         "Use /bad_command to cause an error.\n"
66         f"Your chat id is <code>{update.effective_chat.id}</code>."
67     )
68
69
70 def main() -> None:
71     """Run the bot."""
72     # Create the Application and pass it your bot's token.
73     application = Application.builder().token("TOKEN").build()
74
75     # Register the commands...
76     application.add_handler(CommandHandler("start", start))
77     application.add_handler(CommandHandler("bad_command", bad_command))
78
79     # ...and the error handler
80     application.add_error_handler(error_handler)
81
82     # Run the bot until the user presses Ctrl-C
83     application.run_polling(allowed_updates=Update.ALL_TYPES)
84
85
86 if __name__ == "__main__":
87     main()

```


inlinebot.py

```

1  #!/usr/bin/env python
2  # pylint: disable=unused-argument
3  # This program is dedicated to the public domain under the CC0 license.
4
5  """
6  Don't forget to enable inline mode with @BotFather
7
8  First, a few handler functions are defined. Then, those functions are passed to
9  the Application and registered at their respective places.
10 Then, the bot is started and runs until we press Ctrl-C on the command line.
11
12 Usage:
13 Basic inline bot example. Applies different text transformations.
14 Press Ctrl-C on the command line or send a signal to the process to stop the
15 bot.
16 """
17 import logging
18 from html import escape
19 from uuid import uuid4
20
21 from telegram import InlineQueryResultArticle, InputTextMessageContent, Update
22 from telegram.constants import ParseMode
23 from telegram.ext import Application, CommandHandler, ContextTypes, InlineQueryHandler
24
25 # Enable logging
26 logging.basicConfig(
27     format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
28 )
29 # set higher logging level for httpx to avoid all GET and POST requests being logged
30 logging.getLogger("httpx").setLevel(logging.WARNING)
31
32 logger = logging.getLogger(__name__)
33
34
35 # Define a few command handlers. These usually take the two arguments update and
36 # context.
37 async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
38     """Send a message when the command /start is issued."""
39     await update.message.reply_text("Hi!")
40
41
42 async def help_command(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
43     """Send a message when the command /help is issued."""
44     await update.message.reply_text("Help!")
45
46
47 async def inline_query(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
48     """Handle the inline query. This is run when you type: @botusername <query>"""
49     query = update.inline_query.query
50
51     if not query: # empty query should not be handled
52         return
53
54     results = [
55         InlineQueryResultArticle(

```

(continues on next page)

(continued from previous page)

```

56         id=str(uuid4()),
57         title="Caps",
58         input_message_content=InputTextMessageContent(query.upper()),
59     ),
60     InlineQueryResultArticle(
61         id=str(uuid4()),
62         title="Bold",
63         input_message_content=InputTextMessageContent(
64             f"<b>{escape(query)}</b>", parse_mode=ParseMode.HTML
65         ),
66     ),
67     InlineQueryResultArticle(
68         id=str(uuid4()),
69         title="Italic",
70         input_message_content=InputTextMessageContent(
71             f"<i>{escape(query)}</i>", parse_mode=ParseMode.HTML
72         ),
73     ),
74 ]
75
76 await update.inline_query.answer(results)
77
78
79 def main() -> None:
80     """Run the bot."""
81     # Create the Application and pass it your bot's token.
82     application = Application.builder().token("TOKEN").build()
83
84     # on different commands - answer in Telegram
85     application.add_handler(CommandHandler("start", start))
86     application.add_handler(CommandHandler("help", help_command))
87
88     # on inline queries - show corresponding inline results
89     application.add_handler(InlineQueryHandler(inline_query))
90
91     # Run the bot until the user presses Ctrl-C
92     application.run_polling(allowed_updates=Update.ALL_TYPES)
93
94
95 if __name__ == "__main__":
96     main()

```

inlinekeyboard.py

```

1  #!/usr/bin/env python
2  # pylint: disable=unused-argument
3  # This program is dedicated to the public domain under the CC0 license.
4
5  """
6  Basic example for a bot that uses inline keyboards. For an in-depth explanation,
7  ↪ check out
8  ↪ https://github.com/python-telegram-bot/python-telegram-bot/wiki/InlineKeyboard-
9  ↪ Example.
10 """
11
12 import logging

```

(continues on next page)

(continued from previous page)

```

10
11 from telegram import InlineKeyboardButton, InlineKeyboardMarkup, Update
12 from telegram.ext import Application, CallbackQueryHandler, CommandHandler,
13     ContextTypes
14
15 # Enable logging
16 logging.basicConfig(
17     format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
18 )
19 # set higher logging level for httpx to avoid all GET and POST requests being logged
20 logging.getLogger("httpx").setLevel(logging.WARNING)
21
22 logger = logging.getLogger(__name__)
23
24 async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
25     """Sends a message with three inline buttons attached."""
26     keyboard = [
27         [
28             InlineKeyboardButton("Option 1", callback_data="1"),
29             InlineKeyboardButton("Option 2", callback_data="2"),
30         ],
31         [InlineKeyboardButton("Option 3", callback_data="3")],
32     ]
33
34     reply_markup = InlineKeyboardMarkup(keyboard)
35
36     await update.message.reply_text("Please choose:", reply_markup=reply_markup)
37
38
39 async def button(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
40     """Parses the CallbackQuery and updates the message text."""
41     query = update.callback_query
42
43     # CallbackQueries need to be answered, even if no notification to the user is
44     # needed
45     # Some clients may have trouble otherwise. See https://core.telegram.org/bots/api
46     # #callbackquery
47     await query.answer()
48
49     await query.edit_message_text(text=f"Selected option: {query.data}")
50
51
52 async def help_command(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
53     """Displays info on how to use the bot."""
54     await update.message.reply_text("Use /start to test this bot.")
55
56
57 def main() -> None:
58     """Run the bot."""
59     # Create the Application and pass it your bot's token.
60     application = Application.builder().token("TOKEN").build()
61
62     application.add_handler(CommandHandler("start", start))
63     application.add_handler(CallbackQueryHandler(button))
64     application.add_handler(CommandHandler("help", help_command))

```

(continues on next page)

(continued from previous page)

```

63     # Run the bot until the user presses Ctrl-C
64     application.run_polling(allowed_updates=Update.ALL_TYPES)
65
66
67
68 if __name__ == "__main__":
69     main()

```

inlinekeyboard2.py

```

1  #!/usr/bin/env python
2  # pylint: disable=unused-argument
3  # This program is dedicated to the public domain under the CC0 license.
4
5  """Simple inline keyboard bot with multiple CallbackQueryHandlers.
6
7  This Bot uses the Application class to handle the bot.
8  First, a few callback functions are defined as callback query handler. Then, those
9  ↪ functions are
10 passed to the Application and registered at their respective places.
11 Then, the bot is started and runs until we press Ctrl-C on the command line.
12 Usage:
13 Example of a bot that uses inline keyboard that has multiple CallbackQueryHandlers
14 ↪ arranged in a
15 ConversationHandler.
16 Send /start to initiate the conversation.
17 Press Ctrl-C on the command line to stop the bot.
18 """
19 import logging
20
21 from telegram import InlineKeyboardButton, InlineKeyboardMarkup, Update
22 from telegram.ext import (
23     Application,
24     CallbackQueryHandler,
25     CommandHandler,
26     ContextTypes,
27     ConversationHandler,
28 )
29
30 # Enable logging
31 logging.basicConfig(
32     format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
33 )
34 # set higher logging level for httpx to avoid all GET and POST requests being logged
35 logging.getLogger("httpx").setLevel(logging.WARNING)
36
37 logger = logging.getLogger(__name__)
38
39 # Stages
40 START_ROUTES, END_ROUTES = range(2)
41 # Callback data
42 ONE, TWO, THREE, FOUR = range(4)
43
44 async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:

```

(continues on next page)

(continued from previous page)

```

44     """Send message on `/start`."""
45     # Get user that sent /start and log his name
46     user = update.message.from_user
47     logger.info("User %s started the conversation.", user.first_name)
48     # Build InlineKeyboard where each button has a displayed text
49     # and a string as callback_data
50     # The keyboard is a list of button rows, where each row is in turn
51     # a list (hence `[...]`).
52     keyboard = [
53         [
54             InlineKeyboardButton("1", callback_data=str(ONE)),
55             InlineKeyboardButton("2", callback_data=str(TWO)),
56         ]
57     ]
58     reply_markup = InlineKeyboardMarkup(keyboard)
59     # Send message with text and appended InlineKeyboard
60     await update.message.reply_text("Start handler, Choose a route", reply_
↪ markup=reply_markup)
61     # Tell ConversationHandler that we're in state `FIRST` now
62     return START_ROUTES
63
64
65 async def start_over(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
66     """Prompt same text & keyboard as `start` does but not as new message"""
67     # Get CallbackQuery from Update
68     query = update.callback_query
69     # CallbackQueries need to be answered, even if no notification to the user is
↪ needed
70     # Some clients may have trouble otherwise. See https://core.telegram.org/bots/api
↪ #callbackquery
71     await query.answer()
72     keyboard = [
73         [
74             InlineKeyboardButton("1", callback_data=str(ONE)),
75             InlineKeyboardButton("2", callback_data=str(TWO)),
76         ]
77     ]
78     reply_markup = InlineKeyboardMarkup(keyboard)
79     # Instead of sending a new message, edit the message that
80     # originated the CallbackQuery. This gives the feeling of an
81     # interactive menu.
82     await query.edit_message_text(text="Start handler, Choose a route", reply_
↪ markup=reply_markup)
83     return START_ROUTES
84
85
86 async def one(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
87     """Show new choice of buttons"""
88     query = update.callback_query
89     await query.answer()
90     keyboard = [
91         [
92             InlineKeyboardButton("3", callback_data=str(THREE)),
93             InlineKeyboardButton("4", callback_data=str(FOUR)),
94         ]
95     ]

```

(continues on next page)

(continued from previous page)

```

96     reply_markup = InlineKeyboardMarkup(keyboard)
97     await query.edit_message_text(
98         text="First CallbackQueryHandler, Choose a route", reply_markup=reply_markup
99     )
100     return START_ROUTES
101
102
103 async def two(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
104     """Show new choice of buttons"""
105     query = update.callback_query
106     await query.answer()
107     keyboard = [
108         [
109             InlineKeyboardButton("1", callback_data=str(ONE)),
110             InlineKeyboardButton("3", callback_data=str(THREE)),
111         ]
112     ]
113     reply_markup = InlineKeyboardMarkup(keyboard)
114     await query.edit_message_text(
115         text="Second CallbackQueryHandler, Choose a route", reply_markup=reply_markup
116     )
117     return START_ROUTES
118
119
120 async def three(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
121     """Show new choice of buttons. This is the end point of the conversation."""
122     query = update.callback_query
123     await query.answer()
124     keyboard = [
125         [
126             InlineKeyboardButton("Yes, let's do it again!", callback_data=str(ONE)),
127             InlineKeyboardButton("Nah, I've had enough ...", callback_data=str(TWO)),
128         ]
129     ]
130     reply_markup = InlineKeyboardMarkup(keyboard)
131     await query.edit_message_text(
132         text="Third CallbackQueryHandler. Do want to start over?", reply_markup=reply_
↩ markup
133     )
134     # Transfer to conversation state `SECOND`
135     return END_ROUTES
136
137
138 async def four(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
139     """Show new choice of buttons"""
140     query = update.callback_query
141     await query.answer()
142     keyboard = [
143         [
144             InlineKeyboardButton("2", callback_data=str(TWO)),
145             InlineKeyboardButton("3", callback_data=str(THREE)),
146         ]
147     ]
148     reply_markup = InlineKeyboardMarkup(keyboard)
149     await query.edit_message_text(
150         text="Fourth CallbackQueryHandler, Choose a route", reply_markup=reply_markup

```

(continues on next page)

(continued from previous page)

```

151     )
152     return START_ROUTES
153
154
155 async def end(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
156     """Returns `ConversationHandler.END`, which tells the
157     ConversationHandler that the conversation is over.
158     """
159     query = update.callback_query
160     await query.answer()
161     await query.edit_message_text(text="See you next time!")
162     return ConversationHandler.END
163
164
165 def main() -> None:
166     """Run the bot."""
167     # Create the Application and pass it your bot's token.
168     application = Application.builder().token("TOKEN").build()
169
170     # Setup conversation handler with the states FIRST and SECOND
171     # Use the pattern parameter to pass CallbackQueries with specific
172     # data pattern to the corresponding handlers.
173     # ^ means "start of line/string"
174     # $ means "end of line/string"
175     # So ^ABC$ will only allow 'ABC'
176     conv_handler = ConversationHandler(
177         entry_points=[CommandHandler("start", start)],
178         states={
179             START_ROUTES: [
180                 CallbackQueryHandler(one, pattern="^" + str(ONE) + "$"),
181                 CallbackQueryHandler(two, pattern="^" + str(TWO) + "$"),
182                 CallbackQueryHandler(three, pattern="^" + str(THREE) + "$"),
183                 CallbackQueryHandler(four, pattern="^" + str(FOUR) + "$"),
184             ],
185             END_ROUTES: [
186                 CallbackQueryHandler(start_over, pattern="^" + str(ONE) + "$"),
187                 CallbackQueryHandler(end, pattern="^" + str(TWO) + "$"),
188             ],
189         },
190         fallbacks=[CommandHandler("start", start)],
191     )
192
193     # Add ConversationHandler to application that will be used for handling updates
194     application.add_handler(conv_handler)
195
196     # Run the bot until the user presses Ctrl-C
197     application.run_polling(allowed_updates=Update.ALL_TYPES)
198
199
200 if __name__ == "__main__":
201     main()

```

nestedconversationbot.py

```

1  #!/usr/bin/env python
2  # pylint: disable=unused-argument
3  # This program is dedicated to the public domain under the CC0 license.
4
5  """
6  First, a few callback functions are defined. Then, those functions are passed to
7  the Application and registered at their respective places.
8  Then, the bot is started and runs until we press Ctrl-C on the command line.
9
10 Usage:
11 Example of a bot-user conversation using nested ConversationHandlers.
12 Send /start to initiate the conversation.
13 Press Ctrl-C on the command line or send a signal to the process to stop the
14 bot.
15 """
16
17 import logging
18 from typing import Any, Dict, Tuple
19
20 from telegram import InlineKeyboardButton, InlineKeyboardMarkup, Update
21 from telegram.ext import (
22     Application,
23     CallbackQueryHandler,
24     CommandHandler,
25     ContextTypes,
26     ConversationHandler,
27     MessageHandler,
28     filters,
29 )
30
31 # Enable logging
32 logging.basicConfig(
33     format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
34 )
35 # set higher logging level for httpx to avoid all GET and POST requests being logged
36 logging.getLogger("httpx").setLevel(logging.WARNING)
37
38 logger = logging.getLogger(__name__)
39
40 # State definitions for top level conversation
41 SELECTING_ACTION, ADDING_MEMBER, ADDING_SELF, DESCRIBING_SELF = map(chr, range(4))
42 # State definitions for second level conversation
43 SELECTING_LEVEL, SELECTING_GENDER = map(chr, range(4, 6))
44 # State definitions for descriptions conversation
45 SELECTING_FEATURE, TYPING = map(chr, range(6, 8))
46 # Meta states
47 STOPPING, SHOWING = map(chr, range(8, 10))
48 # Shortcut for ConversationHandler.END
49 END = ConversationHandler.END
50
51 # Different constants for this example
52 (
53     PARENTS,
54     CHILDREN,
55     SELF,

```

(continues on next page)

(continued from previous page)

```

56     GENDER,
57     MALE,
58     FEMALE,
59     AGE,
60     NAME,
61     START_OVER,
62     FEATURES,
63     CURRENT_FEATURE,
64     CURRENT_LEVEL,
65 ) = map(chr, range(10, 22))
66
67
68 # Helper
69 def _name_switcher(level: str) -> Tuple[str, str]:
70     if level == PARENTS:
71         return "Father", "Mother"
72     return "Brother", "Sister"
73
74
75 # Top level conversation callbacks
76 async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
77     """Select an action: Adding parent/child or show data."""
78     text = (
79         "You may choose to add a family member, yourself, show the gathered data, or
↪end the "
80         "conversation. To abort, simply type /stop."
81     )
82
83     buttons = [
84         [
85             InlineKeyboardButton(text="Add family member", callback_data=str(ADDING_
↪MEMBER)),
86             InlineKeyboardButton(text="Add yourself", callback_data=str(ADDING_SELF)),
87         ],
88         [
89             InlineKeyboardButton(text="Show data", callback_data=str(SHOWING)),
90             InlineKeyboardButton(text="Done", callback_data=str(END)),
91         ],
92     ]
93     keyboard = InlineKeyboardMarkup(buttons)
94
95     # If we're starting over we don't need to send a new message
96     if context.user_data.get(START_OVER):
97         await update.callback_query.answer()
98         await update.callback_query.edit_message_text(text=text, reply_
↪markup=keyboard)
99     else:
100         await update.message.reply_text(
101             "Hi, I'm Family Bot and I'm here to help you gather information about
↪your family."
102         )
103         await update.message.reply_text(text=text, reply_markup=keyboard)
104
105     context.user_data[START_OVER] = False
106     return SELECTING_ACTION
107

```

(continues on next page)

(continued from previous page)

```

108
109 async def adding_self(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
110     """Add information about yourself."""
111     context.user_data[CURRENT_LEVEL] = SELF
112     text = "Okay, please tell me about yourself."
113     button = InlineKeyboardButton(text="Add info", callback_data=STR(MALE))
114     keyboard = InlineKeyboardMarkup.from_button(button)
115
116     await update.callback_query.answer()
117     await update.callback_query.edit_message_text(text=text, reply_markup=keyboard)
118
119     return DESCRIBING_SELF
120
121
122 async def show_data(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
123     """Pretty print gathered data."""
124
125     def pretty_print(data: Dict[str, Any], level: str) -> str:
126         people = data.get(level)
127         if not people:
128             return "\nNo information yet."
129
130         return_str = ""
131         if level == SELF:
132             for person in data[level]:
133                 return_str += f"\nName: {person.get(NAME, '-')}, Age: {person.get(AGE,
134 ↪ '-'})}"
135         else:
136             male, female = _name_switcher(level)
137
138             for person in data[level]:
139                 gender = female if person[GENDER] == FEMALE else male
140                 return_str += (
141 ↪ f"\n{gender}: Name: {person.get(NAME, '-')}, Age: {person.get(AGE,
142 ↪ '-'})}"
143             )
144         return return_str
145
146     user_data = context.user_data
147     text = f"Youself:{pretty_print(user_data, SELF)}"
148     text += f"\n\nParents:{pretty_print(user_data, PARENTS)}"
149     text += f"\n\nChildren:{pretty_print(user_data, CHILDREN)}"
150
151     buttons = [[InlineKeyboardButton(text="Back", callback_data=STR(END))]]
152     keyboard = InlineKeyboardMarkup(buttons)
153
154     await update.callback_query.answer()
155     await update.callback_query.edit_message_text(text=text, reply_markup=keyboard)
156     user_data[START_OVER] = True
157
158     return SHOWING
159
160
161 async def stop(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
162     """End Conversation by command."""
163     await update.message.reply_text("Okay, bye.")

```

(continues on next page)

(continued from previous page)

```

162     return END
163
164
165
166 async def end(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
167     """End conversation from InlineKeyboardButton."""
168     await update.callback_query.answer()
169
170     text = "See you around!"
171     await update.callback_query.edit_message_text(text=text)
172
173     return END
174
175
176 # Second level conversation callbacks
177 async def select_level(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
178     """Choose to add a parent or a child."""
179     text = "You may add a parent or a child. Also you can show the gathered data or ↩go back."
180     buttons = [
181         [
182             InlineKeyboardButton(text="Add parent", callback_data=str(PARENTS)),
183             InlineKeyboardButton(text="Add child", callback_data=str(CHILDREN)),
184         ],
185         [
186             InlineKeyboardButton(text="Show data", callback_data=str(SHOWING)),
187             InlineKeyboardButton(text="Back", callback_data=str(END)),
188         ],
189     ]
190     keyboard = InlineKeyboardMarkup(buttons)
191
192     await update.callback_query.answer()
193     await update.callback_query.edit_message_text(text=text, reply_markup=keyboard)
194
195     return SELECTING_LEVEL
196
197
198 async def select_gender(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
199     """Choose to add mother or father."""
200     level = update.callback_query.data
201     context.user_data[CURRENT_LEVEL] = level
202
203     text = "Please choose, whom to add."
204
205     male, female = _name_switcher(level)
206
207     buttons = [
208         [
209             InlineKeyboardButton(text=f"Add {male}", callback_data=str(MALE)),
210             InlineKeyboardButton(text=f"Add {female}", callback_data=str(FEMALE)),
211         ],
212         [
213             InlineKeyboardButton(text="Show data", callback_data=str(SHOWING)),
214             InlineKeyboardButton(text="Back", callback_data=str(END)),
215         ],
216     ]

```

(continues on next page)

(continued from previous page)

```

217     keyboard = InlineKeyboardMarkup(buttons)
218
219     await update.callback_query.answer()
220     await update.callback_query.edit_message_text(text=text, reply_markup=keyboard)
221
222     return SELECTING_GENDER
223
224
225 async def end_second_level(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
226     """Return to top level conversation."""
227     context.user_data[START_OVER] = True
228     await start(update, context)
229
230     return END
231
232
233 # Third level callbacks
234 async def select_feature(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
235     """Select a feature to update for the person."""
236     buttons = [
237         [
238             InlineKeyboardButton(text="Name", callback_data=str(NAME)),
239             InlineKeyboardButton(text="Age", callback_data=str(AGE)),
240             InlineKeyboardButton(text="Done", callback_data=str(END)),
241         ]
242     ]
243     keyboard = InlineKeyboardMarkup(buttons)
244
245     # If we collect features for a new person, clear the cache and save the gender
246     if not context.user_data.get(START_OVER):
247         context.user_data[FEATURES] = {GENDER: update.callback_query.data}
248         text = "Please select a feature to update."
249
250         await update.callback_query.answer()
251         await update.callback_query.edit_message_text(text=text, reply_
↪ markup=keyboard)
252     # But after we do that, we need to send a new message
253     else:
254         text = "Got it! Please select a feature to update."
255         await update.message.reply_text(text=text, reply_markup=keyboard)
256
257     context.user_data[START_OVER] = False
258     return SELECTING_FEATURE
259
260
261 async def ask_for_input(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
262     """Prompt user to input data for selected feature."""
263     context.user_data[CURRENT_FEATURE] = update.callback_query.data
264     text = "Okay, tell me."
265
266     await update.callback_query.answer()
267     await update.callback_query.edit_message_text(text=text)
268
269     return TYPING
270
271

```

(continues on next page)

(continued from previous page)

```

272 async def save_input(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
273     """Save input for feature and return to feature selection."""
274     user_data = context.user_data
275     user_data[FEATURES][user_data[CURRENT_FEATURE]] = update.message.text
276
277     user_data[START_OVER] = True
278
279     return await select_feature(update, context)
280
281
282 async def end_describing(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
283     """End gathering of features and return to parent conversation."""
284     user_data = context.user_data
285     level = user_data[CURRENT_LEVEL]
286     if not user_data.get(level):
287         user_data[level] = []
288     user_data[level].append(user_data[FEATURES])
289
290     # Print upper level menu
291     if level == SELF:
292         user_data[START_OVER] = True
293         await start(update, context)
294     else:
295         await select_level(update, context)
296
297     return END
298
299
300 async def stop_nested(update: Update, context: ContextTypes.DEFAULT_TYPE) -> str:
301     """Completely end conversation from within nested conversation."""
302     await update.message.reply_text("Okay, bye.")
303
304     return STOPPING
305
306
307 def main() -> None:
308     """Run the bot."""
309     # Create the Application and pass it your bot's token.
310     application = Application.builder().token("TOKEN").build()
311
312     # Set up third level ConversationHandler (collecting features)
313     description_conv = ConversationHandler(
314         entry_points=[
315             CallbackQueryHandler(
316                 select_feature, pattern="^" + str(MALE) + "$|^" + str(FEMALE) + "$"
317             )
318         ],
319         states={
320             SELECTING_FEATURE: [
321                 CallbackQueryHandler(ask_for_input, pattern="^(?!" + str(END) + ").*$"
322                 ↪")
323             ],
324             TYPING: [MessageHandler(filters.TEXT & ~filters.COMMAND, save_input)],
325         },
326         fallbacks=[
327             CallbackQueryHandler(end_describing, pattern="^" + str(END) + "$"),

```

(continues on next page)

(continued from previous page)

```

327         CommandHandler("stop", stop_nested),
328     ],
329     map_to_parent={
330         # Return to second level menu
331         END: SELECTING_LEVEL,
332         # End conversation altogether
333         STOPPING: STOPPING,
334     },
335 )
336
337 # Set up second level ConversationHandler (adding a person)
338 add_member_conv = ConversationHandler(
339     entry_points=[CallbackQueryHandler(select_level, pattern="^" + str(ADDING_
↪ MEMBER) + "$")],
340     states={
341         SELECTING_LEVEL: [
342             CallbackQueryHandler(select_gender, pattern=f"^{PARENTS}$|^{CHILDREN}$
↪ ")
343         ],
344         SELECTING_GENDER: [description_conv],
345     },
346     fallbacks=[
347         CallbackQueryHandler(show_data, pattern="^" + str(SHOWING) + "$"),
348         CallbackQueryHandler(end_second_level, pattern="^" + str(END) + "$"),
349         CommandHandler("stop", stop_nested),
350     ],
351     map_to_parent={
352         # After showing data return to top level menu
353         SHOWING: SHOWING,
354         # Return to top level menu
355         END: SELECTING_ACTION,
356         # End conversation altogether
357         STOPPING: END,
358     },
359 )
360
361 # Set up top level ConversationHandler (selecting action)
362 # Because the states of the third level conversation map to the ones of the
↪ second level
363 # conversation, we need to make sure the top level conversation can also handle
↪ them
364 selection_handlers = [
365     add_member_conv,
366     CallbackQueryHandler(show_data, pattern="^" + str(SHOWING) + "$"),
367     CallbackQueryHandler(adding_self, pattern="^" + str(ADDING_SELF) + "$"),
368     CallbackQueryHandler(end, pattern="^" + str(END) + "$"),
369 ]
370 conv_handler = ConversationHandler(
371     entry_points=[CommandHandler("start", start)],
372     states={
373         SHOWING: [CallbackQueryHandler(start, pattern="^" + str(END) + "$")],
374         SELECTING_ACTION: selection_handlers,
375         SELECTING_LEVEL: selection_handlers,
376         DESCRIBING_SELF: [description_conv],
377         STOPPING: [CommandHandler("start", start)],
378     },

```

(continues on next page)

(continued from previous page)

```

379     fallbacks=[CommandHandler("stop", stop)],
380 )
381
382 application.add_handler(conv_handler)
383
384 # Run the bot until the user presses Ctrl-C
385 application.run_polling(allowed_updates=Update.ALL_TYPES)
386
387
388 if __name__ == "__main__":
389     main()

```

State Diagram

passportbot.py

```

1  #!/usr/bin/env python
2  # pylint: disable=unused-argument
3  # This program is dedicated to the public domain under the CC0 license.
4
5  """
6  Simple Bot to print/download all incoming passport data
7
8  See https://telegram.org/blog/passport for info about what telegram passport is.
9
10 See https://github.com/python-telegram-bot/python-telegram-bot/wiki/Telegram-Passport
11 for how to use Telegram Passport properly with python-telegram-bot.
12
13 Note:
14 To use Telegram Passport, you must install PTB via
15 `pip install "python-telegram-bot[passport]"`
16 """
17 import logging
18 from pathlib import Path
19
20 from telegram import Update
21 from telegram.ext import Application, ContextTypes, MessageHandler, filters
22
23 # Enable logging
24
25 logging.basicConfig(
26     format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
27 )
28
29 # set higher logging level for httpx to avoid all GET and POST requests being logged
30 logging.getLogger("httpx").setLevel(logging.WARNING)
31
32 logger = logging.getLogger(__name__)
33
34
35 async def msg(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
36     """Downloads and prints the received passport data."""
37     # Retrieve passport data
38     passport_data = update.message.passport_data

```

(continues on next page)

(continued from previous page)

```

39  # If our nonce doesn't match what we think, this Update did not originate from us
40  # Ideally you would randomize the nonce on the server
41  if passport_data.decrypted_credentials.nonce != "thisisatest":
42      return
43
44  # Print the decrypted credential data
45  # For all elements
46  # Print their decrypted data
47  # Files will be downloaded to current directory
48  for data in passport_data.decrypted_data: # This is where the data gets decrypted
49      if data.type == "phone_number":
50          print("Phone: ", data.phone_number)
51      elif data.type == "email":
52          print("Email: ", data.email)
53      if data.type in (
54          "personal_details",
55          "passport",
56          "driver_license",
57          "identity_card",
58          "internal_passport",
59          "address",
60      ):
61          print(data.type, data.data)
62      if data.type in (
63          "utility_bill",
64          "bank_statement",
65          "rental_agreement",
66          "passport_registration",
67          "temporary_registration",
68      ):
69          print(data.type, len(data.files), "files")
70          for file in data.files:
71              actual_file = await file.get_file()
72              print(actual_file)
73              await actual_file.download_to_drive()
74          if (
75      ↪ data.type in ("passport", "driver_license", "identity_card", "internal_
76      ↪ passport")
77          and data.front_side
78      ):
79          front_file = await data.front_side.get_file()
80          print(data.type, front_file)
81          await front_file.download_to_drive()
82      if data.type in ("driver_license" and "identity_card") and data.reverse_side:
83          reverse_file = await data.reverse_side.get_file()
84          print(data.type, reverse_file)
85          await reverse_file.download_to_drive()
86      if (
87      ↪ data.type in ("passport", "driver_license", "identity_card", "internal_
88      ↪ passport")
89          and data.selfie
90      ):
91          selfie_file = await data.selfie.get_file()
92          print(data.type, selfie_file)
93          await selfie_file.download_to_drive()
94      if data.translation and data.type in (

```

(continues on next page)

(continued from previous page)

```

93         "passport",
94         "driver_license",
95         "identity_card",
96         "internal_passport",
97         "utility_bill",
98         "bank_statement",
99         "rental_agreement",
100        "passport_registration",
101        "temporary_registration",
102    ):
103        print(data.type, len(data.translation), "translation")
104        for file in data.translation:
105            actual_file = await file.get_file()
106            print(actual_file)
107            await actual_file.download_to_drive()
108
109
110 def main() -> None:
111     """Start the bot."""
112     # Create the Application and pass it your token and private key
113     private_key = Path("private.key")
114     application = (
115         Application.builder().token("TOKEN").private_key(private_key.read_bytes()).
116         ↪ build()
117     )
118
119     # On messages that include passport data call msg
120     application.add_handler(MessageHandler(filters.PASSPORT_DATA, msg))
121
122     # Run the bot until the user presses Ctrl-C
123     application.run_polling(allowed_updates=Update.ALL_TYPES)
124
125 if __name__ == "__main__":
126     main()

```

HTML Page

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <title>Telegram passport test!</title>
5      <meta charset="utf-8">
6      <meta content="IE=edge" http-equiv="X-UA-Compatible">
7      <meta content="width=device-width, initial-scale=1" name="viewport">
8  </head>
9  <body>
10     <h1>Telegram passport test</h1>
11
12     <div id="telegram_passport_auth"></div>
13 </body>
14
15 <!-- Needs file from https://github.com/TelegramMessenger/TGPassportJsSDK downloaded_
    ↪ -->

```

(continues on next page)

(continued from previous page)

```

16 <script src="telegram-passport.js"></script>
17 <script>
18     "use strict";
19
20     Telegram.Passport.createAuthButton('telegram_passport_auth', {
21         bot_id: 1234567890, // YOUR BOT ID
22         scope: {
23             data: [{
24                 type: 'id_document',
25                 selfie: true
26             }, 'address_document', 'phone_number', 'email'], v: 1
27         }, // WHAT DATA YOU WANT TO RECEIVE
28         public_key: '-----BEGIN PUBLIC KEY-----\n', // YOUR PUBLIC KEY
29         nonce: 'thisisatest', // YOUR BOT WILL RECEIVE THIS DATA WITH THE REQUEST
30         callback_url: 'https://example.org' // TELEGRAM WILL SEND YOUR USER BACK TO
31         ↪ THIS URL
32     });
33 </script>
34 </html>

```

paymentbot.py

```

1  #!/usr/bin/env python
2  # pylint: disable=unused-argument
3  # This program is dedicated to the public domain under the CC0 license.
4
5  """Basic example for a bot that can receive payment from user."""
6
7  import logging
8
9  from telegram import LabeledPrice, ShippingOption, Update
10 from telegram.ext import (
11     Application,
12     CommandHandler,
13     ContextTypes,
14     MessageHandler,
15     PreCheckoutQueryHandler,
16     ShippingQueryHandler,
17     filters,
18 )
19
20 # Enable logging
21 logging.basicConfig(
22     format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
23 )
24 # set higher logging level for httpx to avoid all GET and POST requests being logged
25 logging.getLogger("httpx").setLevel(logging.WARNING)
26
27 logger = logging.getLogger(__name__)
28
29 PAYMENT_PROVIDER_TOKEN = "PAYMENT_PROVIDER_TOKEN"
30
31
32 async def start_callback(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:

```

(continues on next page)

(continued from previous page)

```

33     """Displays info on how to use the bot."""
34     msg = (
35         "Use /shipping to get an invoice for shipping-payment, or /noshipping for an "
36         "invoice without shipping."
37     )
38
39     await update.message.reply_text(msg)
40
41
42     async def start_with_shipping_callback(update: Update, context: ContextTypes.DEFAULT_
43     ↪TYPE) -> None:
44         """Sends an invoice with shipping-payment."""
45         chat_id = update.message.chat_id
46         title = "Payment Example"
47         description = "Payment Example using python-telegram-bot"
48         # select a payload just for you to recognize its the donation from your bot
49         payload = "Custom-Payload"
50         # In order to get a provider_token see https://core.telegram.org/bots/payments
51         ↪#getting-a-token
52         currency = "USD"
53         # price in dollars
54         price = 1
55         # price * 100 so as to include 2 decimal points
56         # check https://core.telegram.org/bots/payments#supported-currencies for more_
57         ↪details
58         prices = [LabeledPrice("Test", price * 100)]
59
60         # optionally pass need_name=True, need_phone_number=True,
61         # need_email=True, need_shipping_address=True, is_flexible=True
62         await context.bot.send_invoice(
63             chat_id,
64             title,
65             description,
66             payload,
67             PAYMENT_PROVIDER_TOKEN,
68             currency,
69             prices,
70             need_name=True,
71             need_phone_number=True,
72             need_email=True,
73             need_shipping_address=True,
74             is_flexible=True,
75         )
76
77     async def start_without_shipping_callback(
78         update: Update, context: ContextTypes.DEFAULT_TYPE
79     ) -> None:
80         """Sends an invoice without shipping-payment."""
81         chat_id = update.message.chat_id
82         title = "Payment Example"
83         description = "Payment Example using python-telegram-bot"
84         # select a payload just for you to recognize its the donation from your bot
85         payload = "Custom-Payload"
86         # In order to get a provider_token see https://core.telegram.org/bots/payments
87         ↪#getting-a-token

```

(continues on next page)

(continued from previous page)

```

85     currency = "USD"
86     # price in dollars
87     price = 1
88     # price * 100 so as to include 2 decimal points
89     prices = [LabeledPrice("Test", price * 100)]
90
91     # optionally pass need_name=True, need_phone_number=True,
92     # need_email=True, need_shipping_address=True, is_flexible=True
93     await context.bot.send_invoice(
94         chat_id, title, description, payload, PAYMENT_PROVIDER_TOKEN, currency, prices
95     )
96
97
98     async def shipping_callback(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
99     None:
100         """Answers the ShippingQuery with ShippingOptions"""
101         query = update.shipping_query
102         # check the payload, is this from your bot?
103         if query.invoice_payload != "Custom-Payload":
104             # answer False pre_checkout_query
105             await query.answer(ok=False, error_message="Something went wrong...")
106             return
107
108         # First option has a single LabeledPrice
109         options = [ShippingOption("1", "Shipping Option A", [LabeledPrice("A", 100)])]
110         # second option has an array of LabeledPrice objects
111         price_list = [LabeledPrice("B1", 150), LabeledPrice("B2", 200)]
112         options.append(ShippingOption("2", "Shipping Option B", price_list))
113         await query.answer(ok=True, shipping_options=options)
114
115     # after (optional) shipping, it's the pre-checkout
116     async def precheckout_callback(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
117     None:
118         """Answers the PreQecheckoutQuery"""
119         query = update.pre_checkout_query
120         # check the payload, is this from your bot?
121         if query.invoice_payload != "Custom-Payload":
122             # answer False pre_checkout_query
123             await query.answer(ok=False, error_message="Something went wrong...")
124         else:
125             await query.answer(ok=True)
126
127     # finally, after contacting the payment provider...
128     async def successful_payment_callback(update: Update, context: ContextTypes.DEFAULT_
129     TYPE) -> None:
130         """Confirms the successful payment."""
131         # do something after successfully receiving payment?
132         await update.message.reply_text("Thank you for your payment!")
133
134     def main() -> None:
135         """Run the bot."""
136         # Create the Application and pass it your bot's token.
137         application = Application.builder().token("TOKEN").build()

```

(continues on next page)

(continued from previous page)

```

138
139     # simple start function
140     application.add_handler(CommandHandler("start", start_callback))
141
142     # Add command handler to start the payment invoice
143     application.add_handler(CommandHandler("shipping", start_with_shipping_callback))
144     application.add_handler(CommandHandler("noshipping", start_without_shipping_
145     ↪callback))
146
147     # Optional handler if your product requires shipping
148     application.add_handler(ShippingQueryHandler(shipping_callback))
149
150     # Pre-checkout handler to final check
151     application.add_handler(PreCheckoutQueryHandler(precheckout_callback))
152
153     # Success! Notify your user!
154     application.add_handler(
155         MessageHandler(filters.SUCCESSFUL_PAYMENT, successful_payment_callback)
156     )
157
158     # Run the bot until the user presses Ctrl-C
159     application.run_polling(allowed_updates=Update.ALL_TYPES)
160
161 if __name__ == "__main__":
162     main()

```

persistentconversationbot.py

```

1  #!/usr/bin/env python
2  # pylint: disable=unused-argument
3  # This program is dedicated to the public domain under the CC0 license.
4
5  """
6  First, a few callback functions are defined. Then, those functions are passed to
7  the Application and registered at their respective places.
8  Then, the bot is started and runs until we press Ctrl-C on the command line.
9
10 Usage:
11 Example of a bot-user conversation using ConversationHandler.
12 Send /start to initiate the conversation.
13 Press Ctrl-C on the command line or send a signal to the process to stop the
14 bot.
15 """
16
17 import logging
18 from typing import Dict
19
20 from telegram import ReplyKeyboardMarkup, ReplyKeyboardRemove, Update
21 from telegram.ext import (
22     Application,
23     CommandHandler,
24     ContextTypes,
25     ConversationHandler,
26     MessageHandler,

```

(continues on next page)

(continued from previous page)

```

27     PicklePersistence,
28     filters,
29 )
30
31 # Enable logging
32 logging.basicConfig(
33     format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
34 )
35 # set higher logging level for httpx to avoid all GET and POST requests being logged
36 logging.getLogger("httpx").setLevel(logging.WARNING)
37
38 logger = logging.getLogger(__name__)
39
40 CHOOSING, TYPING_REPLY, TYPING_CHOICE = range(3)
41
42 reply_keyboard = [
43     ["Age", "Favourite colour"],
44     ["Number of siblings", "Something else..."],
45     ["Done"],
46 ]
47 markup = ReplyKeyboardMarkup(reply_keyboard, one_time_keyboard=True)
48
49
50 def facts_to_str(user_data: Dict[str, str]) -> str:
51     """Helper function for formatting the gathered user info."""
52     facts = [f"{key} - {value}" for key, value in user_data.items()]
53     return "\n".join(facts).join(["\n", "\n"])
54
55
56 async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
57     """Start the conversation, display any stored data and ask user for input."""
58     reply_text = "Hi! My name is Doctor Botter."
59     if context.user_data:
60         reply_text += (
61             f" You already told me your {', '.join(context.user_data.keys())}. Why don
        ↪ 't you "
62             "tell me something more about yourself? Or change anything I already know.
        ↪ "
63         )
64     else:
65         reply_text += (
66             " I will hold a more complex conversation with you. Why don't you tell me
        ↪ "
67             "something about yourself?"
68         )
69     await update.message.reply_text(reply_text, reply_markup=markup)
70
71     return CHOOSING
72
73
74 async def regular_choice(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
75     """Ask the user for info about the selected predefined choice."""
76     text = update.message.text.lower()
77     context.user_data["choice"] = text
78     if context.user_data.get(text):
79         reply_text = (

```

(continues on next page)

(continued from previous page)

```

80         f"Your {text}? I already know the following about that: {context.user_
↳data[text]}]"
81     )
82     else:
83         reply_text = f"Your {text}? Yes, I would love to hear about that!"
84         await update.message.reply_text(reply_text)
85
86     return TYPING_REPLY
87
88
89 async def custom_choice(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
90     """Ask the user for a description of a custom category."""
91     await update.message.reply_text(
92         'Alright, please send me the category first, for example "Most impressive_
↳skill"'
93     )
94
95     return TYPING_CHOICE
96
97
98 async def received_information(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
↳int:
99     """Store info provided by user and ask for the next category."""
100     text = update.message.text
101     category = context.user_data["choice"]
102     context.user_data[category] = text.lower()
103     del context.user_data["choice"]
104
105     await update.message.reply_text(
106         "Neat! Just so you know, this is what you already told me:"
107         f"{facts_to_str(context.user_data)}"
108         "You can tell me more, or change your opinion on something.",
109         reply_markup=markup,
110     )
111
112     return CHOOSING
113
114
115 async def show_data(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
116     """Display the gathered info."""
117     await update.message.reply_text(
118         f"This is what you already told me: {facts_to_str(context.user_data)}"
119     )
120
121
122 async def done(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
123     """Display the gathered info and end the conversation."""
124     if "choice" in context.user_data:
125         del context.user_data["choice"]
126
127     await update.message.reply_text(
128         f"I learned these facts about you: {facts_to_str(context.user_data)}Until_
↳next time!",
129         reply_markup=ReplyKeyboardRemove(),
130     )
131     return ConversationHandler.END

```

(continues on next page)

```

132
133
134 def main() -> None:
135     """Run the bot."""
136     # Create the Application and pass it your bot's token.
137     persistence = PicklePersistence(filepath="conversationbot")
138     application = Application.builder().token("TOKEN").persistence(persistence).
↳ build()
139
140     # Add conversation handler with the states CHOOSING, TYPING_CHOICE and TYPING_
↳ REPLY
141     conv_handler = ConversationHandler(
142         entry_points=[CommandHandler("start", start)],
143         states={
144             CHOOSING: [
145                 MessageHandler(
146                     filters.Regex("^Age|Favourite colour|Number of siblings)$"),↳
↳ regular_choice
147                 ),
148                 MessageHandler(filters.Regex("^Something else...$"), custom_choice),
149             ],
150             TYPING_CHOICE: [
151                 MessageHandler(
152                     filters.TEXT & ~(filters.COMMAND | filters.Regex("^Done$")),↳
↳ regular_choice
153                 )
154             ],
155             TYPING_REPLY: [
156                 MessageHandler(
157                     filters.TEXT & ~(filters.COMMAND | filters.Regex("^Done$")),
158                     received_information,
159                 )
160             ],
161         },
162         fallbacks=[MessageHandler(filters.Regex("^Done$"), done)],
163         name="my_conversation",
164         persistent=True,
165     )
166
167     application.add_handler(conv_handler)
168
169     show_data_handler = CommandHandler("show_data", show_data)
170     application.add_handler(show_data_handler)
171
172     # Run the bot until the user presses Ctrl-C
173     application.run_polling(allowed_updates=Update.ALL_TYPES)
174
175
176 if __name__ == "__main__":
177     main()

```


pollbot.py

```

1  #!/usr/bin/env python
2  # pylint: disable=unused-argument
3  # This program is dedicated to the public domain under the CC0 license.
4
5  """
6  Basic example for a bot that works with polls. Only 3 people are allowed to interact,
7  ↪ with each
8  poll/quiz the bot generates. The preview command generates a closed poll/quiz,
9  ↪ exactly like the
10 one the user sends the bot
11 """
12 import logging
13
14 from telegram import (
15     KeyboardButton,
16     KeyboardButtonPollType,
17     Poll,
18     ReplyKeyboardMarkup,
19     ReplyKeyboardRemove,
20     Update,
21 )
22 from telegram.constants import ParseMode
23 from telegram.ext import (
24     Application,
25     CommandHandler,
26     ContextTypes,
27     MessageHandler,
28     PollAnswerHandler,
29     PollHandler,
30     filters,
31 )
32
33 # Enable logging
34 logging.basicConfig(
35     format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
36 )
37 # set higher logging level for httpx to avoid all GET and POST requests being logged
38 logging.getLogger("httpx").setLevel(logging.WARNING)
39
40 logger = logging.getLogger(__name__)
41
42 TOTAL_VOTER_COUNT = 3
43
44 async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
45     """Inform user about what this bot can do"""
46     await update.message.reply_text(
47         "Please select /poll to get a Poll, /quiz to get a Quiz or /preview"
48         " to generate a preview for your poll"
49     )
50
51
52 async def poll(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
53     """Sends a predefined poll"""

```

(continues on next page)

(continued from previous page)

```

54     questions = ["Good", "Really good", "Fantastic", "Great"]
55     message = await context.bot.send_poll(
56         update.effective_chat.id,
57         "How are you?",
58         questions,
59         is_anonymous=False,
60         allows_multiple_answers=True,
61     )
62     # Save some info about the poll the bot_data for later use in receive_poll_answer
63     payload = {
64         message.poll.id: {
65             "questions": questions,
66             "message_id": message.message_id,
67             "chat_id": update.effective_chat.id,
68             "answers": 0,
69         }
70     }
71     context.bot_data.update(payload)
72
73
74     async def receive_poll_answer(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
75         """Summarize a users poll vote"""
76         answer = update.poll_answer
77         answered_poll = context.bot_data[answer.poll_id]
78         try:
79             questions = answered_poll["questions"]
80             # this means this poll answer update is from an old poll, we can't do our
81             answering then
82             except KeyError:
83                 return
84             selected_options = answer.option_ids
85             answer_string = ""
86             for question_id in selected_options:
87                 if question_id != selected_options[-1]:
88                     answer_string += questions[question_id] + " and "
89                 else:
90                     answer_string += questions[question_id]
91             await context.bot.send_message(
92                 answered_poll["chat_id"],
93                 f"{update.effective_user.mention_html()} feels {answer_string}!",
94                 parse_mode=ParseMode.HTML,
95             )
96             answered_poll["answers"] += 1
97             # Close poll after three participants voted
98             if answered_poll["answers"] == TOTAL_VOTER_COUNT:
99                 await context.bot.stop_poll(answered_poll["chat_id"], answered_poll["message_
100                 id"])
101
102     async def quiz(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
103         """Send a predefined poll"""
104         questions = ["1", "2", "4", "20"]
105         message = await update.effective_message.reply_poll(
106             "How many eggs do you need for a cake?", questions, type=Poll.QUIZ, correct_
107             option_id=2

```

(continues on next page)

(continued from previous page)

```

106     )
107     # Save some info about the poll the bot_data for later use in receive_quiz_answer
108     payload = {
109         message.poll.id: {"chat_id": update.effective_chat.id, "message_id": message.
↪message_id}
110     }
111     context.bot_data.update(payload)
112
113
114 async def receive_quiz_answer(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
↪None:
115     """Close quiz after three participants took it"""
116     # the bot can receive closed poll updates we don't care about
117     if update.poll.is_closed:
118         return
119     if update.poll.total_voter_count == TOTAL_VOTER_COUNT:
120         try:
121             quiz_data = context.bot_data[update.poll.id]
122             # this means this poll answer update is from an old poll, we can't stop it
↪then
123         except KeyError:
124             return
125         await context.bot.stop_poll(quiz_data["chat_id"], quiz_data["message_id"])
126
127
128 async def preview(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
129     """Ask user to create a poll and display a preview of it"""
130     # using this without a type lets the user chooses what he wants (quiz or poll)
131     button = [[KeyboardButton("Press me!", request_poll=KeyboardButtonPollType())]]
132     message = "Press the button to let the bot generate a preview for your poll"
133     # using one_time_keyboard to hide the keyboard
134     await update.effective_message.reply_text(
135         message, reply_markup=ReplyKeyboardMarkup(button, one_time_keyboard=True)
136     )
137
138
139 async def receive_poll(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
140     """On receiving polls, reply to it by a closed poll copying the received poll"""
141     actual_poll = update.effective_message.poll
142     # Only need to set the question and options, since all other parameters don't
↪matter for
143     # a closed poll
144     await update.effective_message.reply_poll(
145         question=actual_poll.question,
146         options=[o.text for o in actual_poll.options],
147         # with is_closed true, the poll/quiz is immediately closed
148         is_closed=True,
149         reply_markup=ReplyKeyboardRemove(),
150     )
151
152
153 async def help_handler(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
154     """Display a help message"""
155     await update.message.reply_text("Use /quiz, /poll or /preview to test this bot.")
156
157

```

(continues on next page)

(continued from previous page)

```

158 def main() -> None:
159     """Run bot."""
160     # Create the Application and pass it your bot's token.
161     application = Application.builder().token("TOKEN").build()
162     application.add_handler(CommandHandler("start", start))
163     application.add_handler(CommandHandler("poll", poll))
164     application.add_handler(CommandHandler("quiz", quiz))
165     application.add_handler(CommandHandler("preview", preview))
166     application.add_handler(CommandHandler("help", help_handler))
167     application.add_handler(MessageHandler(filters.POLL, receive_poll))
168     application.add_handler(PollAnswerHandler(receive_poll_answer))
169     application.add_handler(PollHandler(receive_quiz_answer))
170
171     # Run the bot until the user presses Ctrl-C
172     application.run_polling(allowed_updates=Update.ALL_TYPES)
173
174
175 if __name__ == "__main__":
176     main()

```

rawapibot.py

This example uses only the pure, “bare-metal” API wrapper.

```

1  #!/usr/bin/env python
2  """Simple Bot to reply to Telegram messages.
3
4  This is built on the API wrapper, see echobot.py to see the same example built
5  on the telegram.ext bot framework.
6  This program is dedicated to the public domain under the CC0 license.
7  """
8  import asyncio
9  import contextlib
10 import logging
11 from typing import NoReturn
12
13 from telegram import Bot, Update
14 from telegram.error import Forbidden, NetworkError
15
16 logging.basicConfig(
17     format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
18 )
19 # set higher logging level for httpx to avoid all GET and POST requests being logged
20 logging.getLogger("httpx").setLevel(logging.WARNING)
21
22 logger = logging.getLogger(__name__)
23
24
25 async def main() -> NoReturn:
26     """Run the bot."""
27     # Here we use the `async with` syntax to properly initialize and shutdown
28     ↪resources.
29     async with Bot("TOKEN") as bot:
30         # get the first pending update_id, this is so we can skip over it in case
31         # we get a "Forbidden" exception.

```

(continues on next page)

(continued from previous page)

```

31     try:
32         update_id = (await bot.get_updates())[0].update_id
33     except IndexError:
34         update_id = None
35
36     logger.info("listening for new messages...")
37     while True:
38         try:
39             update_id = await echo(bot, update_id)
40         except NetworkError:
41             await asyncio.sleep(1)
42         except Forbidden:
43             # The user has removed or blocked the bot.
44             update_id += 1
45
46
47 async def echo(bot: Bot, update_id: int) -> int:
48     """Echo the message the user sent."""
49     # Request updates after the last update_id
50     updates = await bot.get_updates(offset=update_id, timeout=10, allowed_
↪updates=Update.ALL_TYPES)
51     for update in updates:
52         next_update_id = update.update_id + 1
53
54         # your bot can receive updates without messages
55         # and not all messages contain text
56         if update.message and update.message.text:
57             # Reply to the message
58             logger.info("Found message %s!", update.message.text)
59             await update.message.reply_text(update.message.text)
60         return next_update_id
61     return update_id
62
63
64 if __name__ == "__main__":
65     with contextlib.suppress(KeyboardInterrupt): # Ignore exception when Ctrl-C is_
↪pressed
66         asyncio.run(main())

```

timerbot.py

```

1  #!/usr/bin/env python
2  # pylint: disable=unused-argument
3  # This program is dedicated to the public domain under the CC0 license.
4
5  """
6  Simple Bot to send timed Telegram messages.
7
8  This Bot uses the Application class to handle the bot and the JobQueue to send
9  timed messages.
10
11  First, a few handler functions are defined. Then, those functions are passed to
12  the Application and registered at their respective places.
13  Then, the bot is started and runs until we press Ctrl-C on the command line.
14

```

(continues on next page)

(continued from previous page)

```

15 Usage:
16 Basic Alarm Bot example, sends a message after a set time.
17 Press Ctrl-C on the command line or send a signal to the process to stop the
18 bot.
19
20 Note:
21 To use the JobQueue, you must install PTB via
22 `pip install "python-telegram-bot[job-queue]"`
23 """
24
25 import logging
26
27 from telegram import Update
28 from telegram.ext import Application, CommandHandler, ContextTypes
29
30 # Enable logging
31 logging.basicConfig(
32     format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
33 )
34
35 # Define a few command handlers. These usually take the two arguments update and
36 # context.
37 # Best practice would be to replace context with an underscore,
38 # since context is an unused local variable.
39 # This being an example and not having context present confusing beginners,
40 # we decided to have it present as context.
41
42 async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
43     """Sends explanation on how to use the bot."""
44     await update.message.reply_text("Hi! Use /set <seconds> to set a timer")
45
46
47 async def alarm(context: ContextTypes.DEFAULT_TYPE) -> None:
48     """Send the alarm message."""
49     job = context.job
50     await context.bot.send_message(job.chat_id, text=f"Beep! {job.data} seconds are_
51     ↪over!")
52
53 def remove_job_if_exists(name: str, context: ContextTypes.DEFAULT_TYPE) -> bool:
54     """Remove job with given name. Returns whether job was removed."""
55     current_jobs = context.job_queue.get_jobs_by_name(name)
56     if not current_jobs:
57         return False
58     for job in current_jobs:
59         job.schedule_removal()
60     return True
61
62
63 async def set_timer(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
64     """Add a job to the queue."""
65     chat_id = update.effective_message.chat_id
66     try:
67         # args[0] should contain the time for the timer in seconds
68         due = float(context.args[0])
69         if due < 0:

```

(continues on next page)

(continued from previous page)

```

70         await update.effective_message.reply_text("Sorry we can not go back to_
↪future!")
71         return
72
73         job_removed = remove_job_if_exists(str(chat_id), context)
74         context.job_queue.run_once(alarm, due, chat_id=chat_id, name=str(chat_id),_
↪data=due)
75
76         text = "Timer successfully set!"
77         if job_removed:
78             text += " Old one was removed."
79         await update.effective_message.reply_text(text)
80
81     except (IndexError, ValueError):
82         await update.effective_message.reply_text("Usage: /set <seconds>")
83
84
85 async def unset(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
86     """Remove the job if the user changed their mind."""
87     chat_id = update.message.chat_id
88     job_removed = remove_job_if_exists(str(chat_id), context)
89     text = "Timer successfully cancelled!" if job_removed else "You have no active_
↪timer."
90     await update.message.reply_text(text)
91
92
93 def main() -> None:
94     """Run bot."""
95     # Create the Application and pass it your bot's token.
96     application = Application.builder().token("TOKEN").build()
97
98     # on different commands - answer in Telegram
99     application.add_handler(CommandHandler(["start", "help"], start))
100    application.add_handler(CommandHandler("set", set_timer))
101    application.add_handler(CommandHandler("unset", unset))
102
103    # Run the bot until the user presses Ctrl-C
104    application.run_polling(allowed_updates=Update.ALL_TYPES)
105
106
107 if __name__ == "__main__":
108     main()

```

webappbot.py

```

1  #!/usr/bin/env python
2  # pylint: disable=unused-argument
3  # This program is dedicated to the public domain under the CC0 license.
4
5  """
6  Simple example of a Telegram WebApp which displays a color picker.
7  The static website for this website is hosted by the PTB team for your convenience.
8  Currently only showcases starting the WebApp via a KeyboardButton, as all other_
↪methods would
9  require a bot token.

```

(continues on next page)

(continued from previous page)

```

10 """
11 import json
12 import logging
13
14 from telegram import KeyboardButton, ReplyKeyboardMarkup, ReplyKeyboardRemove, Update,
15     ↳ WebAppInfo
16 from telegram.ext import Application, CommandHandler, ContextTypes, MessageHandler,
17     ↳ filters
18
19 # Enable logging
20 logging.basicConfig(
21     format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
22 )
23 # set higher logging level for httpx to avoid all GET and POST requests being logged
24 logging.getLogger("httpx").setLevel(logging.WARNING)
25
26 logger = logging.getLogger(__name__)
27
28 # Define a `/start` command handler.
29 async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
30     """Send a message with a button that opens a the web app."""
31     await update.message.reply_text(
32         "Please press the button below to choose a color via the WebApp.",
33         reply_markup=ReplyKeyboardMarkup.from_button(
34             KeyboardButton(
35                 text="Open the color picker!",
36                 web_app=WebAppInfo(url="https://python-telegram-bot.org/static/
37     ↳webappbot"),
38             )
39         ),
40     )
41
42 # Handle incoming WebAppData
43 async def web_app_data(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
44     """Print the received data and remove the button."""
45     # Here we use `json.loads`, since the WebApp sends the data JSON serialized string
46     # (see webappbot.html)
47     data = json.loads(update.effective_message.web_app_data.data)
48     await update.message.reply_html(
49         text=(
50             f"You selected the color with the HEX value <code>{data['hex']}</code>."
51     ↳The "
52             f"corresponding RGB value is <code>{tuple(data['rgb'].values())}</code>."
53         ),
54         reply_markup=ReplyKeyboardRemove(),
55     )
56
57 def main() -> None:
58     """Start the bot."""
59     # Create the Application and pass it your bot's token.
60     application = Application.builder().token("TOKEN").build()
61
62     application.add_handler(CommandHandler("start", start))

```

(continues on next page)

(continued from previous page)

```

62     application.add_handler(MessageHandler(filters.StatusUpdate.WEB_APP_DATA, web_app_
    ↪data))
63
64     # Run the bot until the user presses Ctrl-C
65     application.run_polling(allowed_updates=Update.ALL_TYPES)
66
67
68 if __name__ == "__main__":
69     main()

```

HTML Page

```

1  <!--
2      Simple static Telegram WebApp. Does not verify the WebAppInitData, as a bot token
    ↪would be needed for that.
3  -->
4  <!DOCTYPE html>
5  <html lang="en">
6  <head>
7      <meta charset="UTF-8">
8      <title>python-telegram-bot Example WebApp</title>
9      <script src="https://telegram.org/js/telegram-web-app.js"></script>
10     <script src="https://cdn.jsdelivr.net/npm/@jaames/iro@5"></script>
11 </head>
12 <script type="text/javascript">
13     const colorPicker = new iro.ColorPicker('#picker', {
14         borderColor: "#ffffff",
15         borderWidth: 1,
16         width: Math.round(document.documentElement.clientWidth / 2),
17     });
18     colorPicker.on('color:change', function (color) {
19         document.body.style.background = color.hexString;
20     });
21
22     Telegram.WebApp.ready();
23     Telegram.WebApp.MainButton.setText('Choose Color').show().onClick(function () {
24         const data = JSON.stringify({hex: colorPicker.color.hexString, rgb:
    ↪colorPicker.color.rgb});
25         Telegram.WebApp.sendData(data);
26         Telegram.WebApp.close();
27     });
28 </script>
29 <body style="background-color: #ffffff">
30 <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
31 #000000;">
32     <div id="picker"
33         style="display: flex; justify-content: center; align-items: center; height:
    ↪100%; width: 100%"></div>
34 </div>
35 </body>
36 <script type="text/javascript">
37     Telegram.WebApp.expand();

```

(continues on next page)

```
</script>
</html>
```

10.5 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.

10.5.1 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.

10.5.2 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 were 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.

10.5.3 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

10.6 Changelog

10.6.1 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)

10.6.2 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)

10.6.3 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)

10.6.4 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)

10.6.5 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)

10.6.6 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)

10.6.7 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](#))

10.6.8 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 Builds 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)

10.6.9 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)

10.6.10 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)

10.6.11 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)

10.6.12 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)

10.6.13 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)

10.6.14 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.logout` and Improve Related Unit Tests (#3147)
- Make `Bot.delete_my_commands` a Coroutine Function (#3136)
- Fix `ConversationHandler.check_update` not respecting `per_user` (#3128)

Minor Changes, Documentation Improvements and CI

- Add Python 3.11 to Test Suite & Adapt Enum Behaviour (#3168)
- Drop Manual Token Validation (#3167)
- Simplify Unit Tests for `Bot.send_chat_action` (#3151)
- Drop pre-commit Dependencies from `requirements-dev.txt` (#3120)
- Change Default Values for `concurrent_updates` and `connection_pool_size` (#3127)
- Documentation Improvements (#3139, #3153, #3135)
- Type Hinting Fixes (#3202)

Dependencies

- Bump `sphinx` from 5.0.2 to 5.1.1 (#3177)
- Update pre-commit Dependencies (#3085)
- Bump `pytest-asyncio` from 0.18.3 to 0.19.0 (#3158)
- Update `tornado` requirement from `~6.1` to `~6.2` (#3149)
- Bump `black` from 22.3.0 to 22.6.0 (#3132)
- Bump `actions/setup-python` from 3 to 4 (#3131)

10.6.15 Version 20.0a2

Released 2022-06-27

This is the technical changelog for version 20.0a2. More elaborate release notes can be found in the news channel [@pythontelegrambotchannel](#).

Major Changes

- Full Support for API 6.1 (#3112)

New Features

- Add Additional Shortcut Methods to Chat (#3115)
- Mermaid-based Example State Diagrams (#3090)

Minor Changes, Documentation Improvements and CI

- Documentation Improvements (#3103, #3121, #3098)
- Stabilize CI (#3119)
- Bump pyupgrade from 2.32.1 to 2.34.0 (#3096)
- Bump furo from 2022.6.4 to 2022.6.4.1 (#3095)
- Bump mypy from 0.960 to 0.961 (#3093)

10.6.16 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)

10.6.17 Version 20.0a0

Released 2022-05-06

This is the technical changelog for version 20.0a0. More elaborate release notes can be found in the news channel [@pythontelegrambotchannel](#).

Major Changes:

- Refactor Initialization of Persistence Classes (#2604)
- Drop Non-CallbackContext API (#2617)
- Remove `__dict__` from `__slots__` and drop Python 3.6 (#2619, #2636)
- Move and Rename `TelegramDecryptionError` to `telegram.error.PassportDecryptionError` (#2621)
- Make BasePersistence Methods Abstract (#2624)
- Remove `day_is_strict` argument of `JobQueue.run_monthly` (#2634 by [iota-008](#))
- Move Defaults to `telegram.ext` (#2648)
- Remove Deprecated Functionality (#2644, #2740, #2745)
- Overhaul of Filters (#2759, #2922)
- Switch to `asyncio` and Refactor PTBs Architecture (#2731)
- Improve `Job.__getattr__` (#2832)
- Remove `telegram.ReplyMarkup` (#2870)
- Persistence of Bots: Refactor Automatic Replacement and Integration with `TelegramObject` (#2893)

New Features:

- Introduce Builder Pattern (#2646)
- Add `Filters.update.edited` (#2705 by [PhilippFr](#))
- Introduce Enums for `telegram.constants` (#2708)
- Accept File Paths for `private_key` (#2724)
- Associate Jobs with `chat/user_id` (#2731)
- Convenience Functionality for `ChatInviteLinks` (#2782)
- Add `Dispatcher.add_handlers` (#2823)
- Improve Error Messages in `CommandHandler.__init__` (#2837)
- `Defaults.protect_content` (#2840)
- Add `Dispatcher.migrate_chat_data` (#2848 by [DonalDuck004](#))
- Add Method `drop_chat/user_data` to `Dispatcher` and `Persistence` (#2852)
- Add methods `ChatPermissions.{all, no}_permissions` (#2948)
- Full Support for API 6.0 (#2956)
- Add Python 3.10 to Test Suite (#2968)

Bug Fixes & Minor Changes:

- Improve Type Hinting for `CallbackContext` (#2587 by [revolter](#))
- Fix Signatures and Improve `test_official` (#2643)
- Refine `Dispatcher.dispatch_error` (#2660)
- Make `InlineQuery.answer` Raise `ValueError` (#2675)
- Improve Signature Inspection for Bot Methods (#2686)
- Introduce `TelegramObject.set/get_bot` (#2712 by [zpavloudis](#))
- Improve Subscription of `TelegramObject` (#2719 by [SimonDamberg](#))
- Use Enums for Dynamic Types & Rename Two Attributes in `ChatMember` (#2817)
- Return Plain Dicts from `BasePersistence.get_*_data` (#2873)
- Fix a Bug in `ChatMemberUpdated.difference` (#2947)
- Update Dependency Policy (#2958)

Internal Restructurings & Improvements:

- Add User Friendly Type Check For Init Of `{Inline, Reply}KeyboardMarkup` (#2657)
- Warnings Overhaul (#2662)
- Clear Up Import Policy (#2671)
- Mark Internal Modules As Private (#2687 by [kencx](#))
- Handle Filepaths via the `pathlib` Module (#2688 by [eldbud](#))
- Refactor MRO of `InputMedia*` and Some File-Like Classes (#2717 by [eldbud](#))
- Update Exceptions for Immutable Attributes (#2749)
- Refactor Warnings in `ConversationHandler` (#2755, #2784)
- Use `__all__` Consistently (#2805)

CI, Code Quality & Test Suite Improvements:

- Add Custom `pytest` Marker to Ease Development (#2628)
- Pass Failing Jobs to Error Handlers (#2692)
- Update Notification Workflows (#2695)
- Use Error Messages for `pylint` Instead of Codes (#2700 by [Piraty](#))
- Make Tests Agnostic of the CWD (#2727 by [eldbud](#))
- Update Code Quality Dependencies (#2748)
- Improve Code Quality (#2783)
- Update `pre-commit` Settings & Improve a Test (#2796)
- Improve Code Quality & Test Suite (#2843)
- Fix failing animation tests (#2865)
- Update and Expand Tests & pre-commit Settings and Improve Code Quality (#2925)
- Extend Code Formatting With `Black` (#2972)
- Update Workflow Permissions (#2984)

- Adapt Tests to Changed Bot.get_file Behavior (#2995)

Documentation Improvements:

- Doc Fixes (#2597)
- Add Code Comment Guidelines to Contribution Guide (#2612)
- Add Cross-References to External Libraries & Other Documentation Improvements (#2693, #2691 by joesinghh, #2739 by eldbud)
- Use Furo Theme, Make Parameters Referenceable, Add Documentation Building to CI, Improve Links to Source Code & Other Improvements (#2856, #2798, #2854, #2841)
- Documentation Fixes & Improvements (#2822)
- Replace git.io Links (#2872 by murugu-21)
- Overhaul Readmes, Update RTD Startpage & Other Improvements (#2969)

10.6.18 Version 13.11

Released 2022-02-02

This is the technical changelog for version 13.11. More elaborate release notes can be found in the news channel [@pythontelegrambotchannel](#).

Major Changes:

- Full Support for Bot API 5.7 (#2881)

10.6.19 Version 13.10

Released 2022-01-03

This is the technical changelog for version 13.10. More elaborate release notes can be found in the news channel [@pythontelegrambotchannel](#).

Major Changes:

- Full Support for API 5.6 (#2835)

Minor Changes & Doc fixes:

- Update Copyright to 2022 (#2836)
- Update Documentation of BotCommand (#2820)

10.6.20 Version 13.9

Released 2021-12-11

This is the technical changelog for version 13.9. More elaborate release notes can be found in the news channel [@pythontelegrambotchannel](#).

Major Changes:

- Full Support for Api 5.5 (#2809)

Minor Changes

- Adjust Automated Locking of Inactive Issues (#2775)

10.6.21 Version 13.8.1

Released 2021-11-08

This is the technical changelog for version 13.8.1. More elaborate release notes can be found in the news channel [@pythontelegrambotchannel](#).

Doc fixes:

- Add ChatJoinRequest(Handler) to Docs (#2771)

10.6.22 Version 13.8

Released 2021-11-08

This is the technical changelog for version 13.8. More elaborate release notes can be found in the news channel [@pythontelegrambotchannel](#).

Major Changes:

- Full support for API 5.4 (#2767)

Minor changes, CI improvements, Doc fixes and Type hinting:

- Create Issue Template Forms (#2689)
- Fix camelCase Functions in ExtBot (#2659)
- Fix Empty Captions not Being Passed by Bot.copy_message (#2651)
- Fix Setting Thumbs When Uploading A Single File (#2583)
- Fix Bug in BasePersistence.insert/replace_bot for Objects with __dict__ not in __slots__ (#2603)

10.6.23 Version 13.7

Released 2021-07-01

This is the technical changelog for version 13.7. More elaborate release notes can be found in the news channel [@pythontelegrambotchannel](#).

Major Changes:

- Full support for Bot API 5.3 (#2572)

Bug Fixes:

- Fix Bug in BasePersistence.insert/replace_bot for Objects with __dict__ in their slots (#2561)
- Remove Incorrect Warning About Defaults and ExtBot (#2553)

Minor changes, CI improvements, Doc fixes and Type hinting:

- Type Hinting Fixes (#2552)
- Doc Fixes (#2551)
- Improve Deprecation Warning for __slots__ (#2574)
- Stabilize CI (#2575)
- Fix Coverage Configuration (#2571)
- Better Exception-Handling for BasePersistence.replace/insert_bot (#2564)
- Remove Deprecated pass_args from Deeplinking Example (#2550)

10.6.24 Version 13.6

Released 2021-06-06

New Features:

- Arbitrary `callback_data` (#1844)
- Add `ContextTypes` & `BasePersistence.refresh_user/chat/bot_data` (#2262)
- Add `Filters.attachment` (#2528)
- Add `pattern` Argument to `ChosenInlineResultHandler` (#2517)

Major Changes:

- Add `slots` (#2345)

Minor changes, CI improvements, Doc fixes and Type hinting:

- Doc Fixes (#2495, #2510)
- Add `max_connections` Parameter to `Updater.start_webhook` (#2547)
- Fix for `Promise.done_callback` (#2544)
- Improve Code Quality (#2536, #2454)
- Increase Test Coverage of `CallbackQueryHandler` (#2520)
- Stabilize CI (#2522, #2537, #2541)
- Fix `send_phone_number_to_provider` argument for `Bot.send_invoice` (#2527)
- Handle Classes as Input for `BasePersistence.replace/insert_bot` (#2523)
- Bump Tornado Version and Remove Workaround from #2067 (#2494)

10.6.25 Version 13.5

Released 2021-04-30

Major Changes:

- Full support of Bot API 5.2 (#2489).

Note: The `start_parameter` argument of `Bot.send_invoice` and the corresponding shortcuts is now optional, so the order of parameters had to be changed. Make sure to update your method calls accordingly.

- Update `ChatActions`, Deprecating `ChatAction.RECORD_AUDIO` and `ChatAction.UPLOAD_AUDIO` (#2460)

New Features:

- Convenience Utilities & Example for Handling `ChatMemberUpdated` (#2490)
- `Filters.forwarded_from` (#2446)

Minor changes, CI improvements, Doc fixes and Type hinting:

- Improve Timeouts in `ConversationHandler` (#2417)
- Stabilize CI (#2480)
- Doc Fixes (#2437)
- Improve Type Hints of Data Filters (#2456)
- Add Two `UserWarnings` (#2464)
- Improve Code Quality (#2450)

- Update Fallback Test-Bots (#2451)
- Improve Examples (#2441, #2448)

10.6.26 Version 13.4.1

Released 2021-03-14

Hot fix release:

- Fixed a bug in `setup.py` (#2431)

10.6.27 Version 13.4

Released 2021-03-14

Major Changes:

- Full support of Bot API 5.1 (#2424)

Minor changes, CI improvements, doc fixes and type hinting:

- Improve `Updater.set_webhook` (#2419)
- Doc Fixes (#2404)
- Type Hinting Fixes (#2425)
- Update pre-commit Settings (#2415)
- Fix Logging for Vendored `urllib3` (#2427)
- Stabilize Tests (#2409)

10.6.28 Version 13.3

Released 2021-02-19

Major Changes:

- Make cryptography Dependency Optional & Refactor Some Tests (#2386, #2370)
- Deprecate `MessageQueue` (#2393)

Bug Fixes:

- Refactor Defaults Integration (#2363)
- Add Missing `telegram.SecureValue` to init and Docs (#2398)

Minor changes:

- Doc Fixes (#2359)

10.6.29 Version 13.2

Released 2021-02-02

Major Changes:

- Introduce `python-telegram-bot-raw` (#2324)
- Explicit Signatures for Shortcuts (#2240)

New Features:

- Add Missing Shortcuts to `Message` (#2330)

- Rich Comparison for Bot (#2320)
- Add `run_async` Parameter to `ConversationHandler` (#2292)
- Add New Shortcuts to Chat (#2291)
- Add New Constant `MAX_ANSWER_CALLBACK_QUERY_TEXT_LENGTH` (#2282)
- Allow Passing Custom Filename For All Media (#2249)
- Handle Bytes as File Input (#2233)

Bug Fixes:

- Fix Escaping in Nested Entities in Message Properties (#2312)
- Adjust Calling of `Dispatcher.update_persistence` (#2285)
- Add `quote` kwarg to `Message.reply_copy` (#2232)
- `ConversationHandler`: Docs & `edited_channel_post` behavior (#2339)

Minor changes, CI improvements, doc fixes and type hinting:

- Doc Fixes (#2253, #2225)
- Reduce Usage of `typing.Any` (#2321)
- Extend Deeplinking Example (#2335)
- Add `pyupgrade` to pre-commit Hooks (#2301)
- Add PR Template (#2299)
- Drop Nightly Tests & Update Badges (#2323)
- Update Copyright (#2289, #2287)
- Change Order of Class DocStrings (#2256)
- Add macOS to Test Matrix (#2266)
- Start Using Versioning Directives in Docs (#2252)
- Improve Annotations & Docs of Handlers (#2243)

10.6.30 Version 13.1

Released 2020-11-29

Major Changes:

- Full support of Bot API 5.0 (#2181, #2186, #2190, #2189, #2183, #2184, #2188, #2185, #2192, #2196, #2193, #2223, #2199, #2187, #2147, #2205)

New Features:

- Add `Defaults.run_async` (#2210)
- Improve and Expand `CallbackQuery` Shortcuts (#2172)
- Add XOR Filters and make `Filters.name` a Property (#2179)
- Add `Filters.document.file_extension` (#2169)
- Add `Filters.caption_regex` (#2163)
- Add `Filters.chat_type` (#2128)
- Handle Non-Binary File Input (#2202)

Bug Fixes:

- Improve Handling of Custom Objects in `BasePersistence.insert/replace_bot` (#2151)

- Fix bugs in `replace/insert_bot` (#2218)

Minor changes, CI improvements, doc fixes and type hinting:

- Improve Type hinting (#2204, #2118, #2167, #2136)
- Doc Fixes & Extensions (#2201, #2161)
- Use F-Strings Where Possible (#2222)
- Rename kwargs to `_kwargs` where possible (#2182)
- Comply with PEP561 (#2168)
- Improve Code Quality (#2131)
- Switch Code Formatting to Black (#2122, #2159, #2158)
- Update Wheel Settings (#2142)
- Update `timerbot.py` to v13.0 (#2149)
- Overhaul Constants (#2137)
- Add Python 3.9 to Test Matrix (#2132)
- Switch Codecov to GitHub Action (#2127)
- Specify Required pytz Version (#2121)

10.6.31 Version 13.0

Released 2020-10-07

For a detailed guide on how to migrate from v12 to v13, see this [wiki page](#).

Major Changes:

- Deprecate old-style callbacks, i.e. set `use_context=True` by default (#2050)
- Refactor Handling of Message VS Update Filters (#2032)
- Deprecate `Message.default_quote` (#1965)
- Refactor persistence of Bot instances (#1994)
- Refactor `JobQueue` (#1981)
- Refactor handling of kwargs in Bot methods (#1924)
- Refactor `Dispatcher.run_async`, deprecating the `@run_async` decorator (#2051)

New Features:

- Type Hinting (#1920)
- Automatic Pagination for `answer_inline_query` (#2072)
- `Defaults.tzinfo` (#2042)
- Extend rich comparison of objects (#1724)
- Add `Filters.via_bot` (#2009)
- Add missing shortcuts (#2043)
- Allow `DispatcherHandlerStop` in `ConversationHandler` (#2059)
- Make Errors picklable (#2106)

Minor changes, CI improvements, doc fixes or bug fixes:

- Fix Webhook not working on Windows with Python 3.8+ (#2067)
- Fix setting thumbs with `send_media_group` (#2093)

- Make MessageHandler filter for Filters.update first (#2085)
- Fix PicklePersistence.flush() with only bot_data (#2017)
- Add test for clean argument of Updater.start_polling/webhook (#2002)
- Doc fixes, refinements and additions (#2005, #2008, #2089, #2094, #2090)
- CI fixes (#2018, #2061)
- Refine pollbot.py example (#2047)
- Refine Filters in examples (#2027)
- Rename echobot examples (#2025)
- Use Lock-Bot to lock old threads (#2048, #2052, #2049, #2053)

10.6.32 Version 12.8

Released 2020-06-22

Major Changes:

- Remove Python 2 support (#1715)
- Bot API 4.9 support (#1980)
- IDs/Usernames of Filters.user and Filters.chat can now be updated (#1757)

Minor changes, CI improvements, doc fixes or bug fixes:

- Update contribution guide and stale bot (#1937)
- Remove NullHandlers (#1913)
- Improve and expand examples (#1943, #1995, #1983, #1997)
- Doc fixes (#1940, #1962)
- Add User.send_poll() shortcut (#1968)
- Ignore private attributes en TelegramObject.to_dict() (#1989)
- Stabilize CI (#2000)

10.6.33 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:

- Stabilize CI (#1919, #1931)
- Use ABCs @abstractmethod instead of raising NotImplementedError for Handler, BasePersistence and BaseFilter (#1905)

- Doc fixes (#1914, #1902, #1910)

10.6.34 Version 12.6.1

Released 2020-04-11

Bug fixes:

- Fix serialization of `reply_markup` in media messages (#1889)

10.6.35 Version 12.6

Released 2020-04-10

Major Changes:

- Bot API 4.7 support. **Note:** In `Bot.create_new_sticker_set` and `Bot.add_sticker_to_set`, the order of the parameters had be changed, as the `png_sticker` parameter is now optional. (#1858)

Minor changes, CI improvements or bug fixes:

- Add tests for `swtich_inline_query(_current_chat)` with empty string (#1635)
- Doc fixes (#1854, #1874, #1884)
- Update issue templates (#1880)
- Favor concrete types over “Iterable” (#1882)
- Pass last valid `CallbackContext` to `TIMEOUT` handlers of `ConversationHandler` (#1826)
- Tweak handling of persistence and update persistence after job calls (#1827)
- Use `checkout@v2` for GitHub actions (#1887)

10.6.36 Version 12.5.1

Released 2020-03-30

Minor changes, doc fixes or bug fixes:

- Add missing docs for `PollHandler` and `PollAnswerHandler` (#1853)
- Fix wording in `Filters` docs (#1855)
- Reorder tests to make them more stable (#1835)
- Make `ConversationHandler` attributes immutable (#1756)
- Make `PrefixHandler` attributes `command` and `prefix` editable (#1636)
- Fix UTC as default `tzinfo` for `Job` (#1696)

10.6.37 Version 12.5

Released 2020-03-29

New Features:

- `Bot.link` gives the *t.me* link of the bot (#1770)

Major Changes:

- Bot API 4.5 and 4.6 support. (#1508, #1723)

Minor changes, CI improvements or bug fixes:

- Remove legacy CI files (#1783, #1791)
- Update pre-commit config file (#1787)
- Remove builtin names (#1792)
- CI improvements (#1808, #1848)
- Support Python 3.8 (#1614, #1824)
- Use stale bot for auto closing stale issues (#1820, #1829, #1840)
- Doc fixes (#1778, #1818)
- Fix typo in `edit_message_media` (#1779)
- In examples, answer CallbackQueries and use `edit_message_text` shortcut (#1721)
- Revert accidental change in vendored urllib3 (#1775)

10.6.38 Version 12.4.2

Released 2020-02-10

Bug Fixes

- Pass correct `parse_mode` to `InlineResults` if `bot.defaults` is `None` (#1763)
- Make sure PP can read files that dont have `bot_data` (#1760)

10.6.39 Version 12.4.1

Released 2020-02-08

This is a quick release for #1744 which was accidently left out of v12.4.0 though mentioned in the release notes.

10.6.40 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` now 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 `dispatcher` 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 assignment of `Dispatcher.job_queue` (#1698)
- Expose dispatcher as property for `CallbackContext` (#1684)
- Fix `None` check in `JobQueue._put()` (#1707)
- Log datetimes correctly in `JobQueue` (#1714)
- Fix false `Message.link` creation for private groups (#1741)
- Add option `--with-upstream-urllib3` to `setup.py` to allow using non-vendored version (#1725)
- Fix persistence for nested `ConversationHandlers` (#1679)
- Improve handling of non-decodable server responses (#1623)
- Fix download for files without `file_path` (#1591)
- `test_webhook_invalid_posts` is now considered flaky and retried on failure (#1758)

10.6.41 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 assignments for optional arguments (#1600)
- Allow private groups for `Message.link` (#1619).
- Fix wrong signature call for `ConversationHandler.TIMEOUT` handlers (#1653).

10.6.42 Version 12.2.0

Released 2019-10-14

New features:

- Nested `ConversationHandlers` (#1512).

Minor changes, CI improvements or bug fixes:

- Fix CI failures due to non-backward compat attrs dependency (#1540).
- `travis.yaml`: `TEST_OFFICIAL` removed from `allowed_failures`.
- Fix typos in examples (#1537).

- Fix Bot.to_dict to use proper first_name (#1525).
- Refactor test_commandhandler.py (#1408).
- Add Python 3.8 (RC version) to Travis testing matrix (#1543).
- test_bot.py: Add to_dict test (#1544).
- Flake config moved into setup.cfg (#1546).

10.6.43 Version 12.1.1

Released 2019-09-18

Hot fix release

Fixed regression in the vendored urllib3 (#1517).

10.6.44 Version 12.1.0

Released 2019-09-13

Major changes:

- Bot API 4.4 support (#1464, #1510)
- Add `get_file` method to `Animation` & `ChatPhoto`. Add, `get_small_file` & `get_big_file` methods to `ChatPhoto` (#1489)
- Tools for deep linking (#1049)

Minor changes and/or bug fixes:

- Documentation fixes (#1500, #1499)
- Improved examples (#1502)

10.6.45 Version 12.0.0

Released 2019-08-29

Well... This felt like decades. But here we are with a new release.

Expect minor releases soon (mainly complete Bot API 4.4 support)

Major and/or breaking changes:

- Context based callbacks
- Persistence
- PrefixHandler added (Handler overhaul)
- Deprecation of `RegexHandler` and `edited_messages`, `channel_post`, etc. arguments (Filter overhaul)
- Various `ConversationHandler` changes and fixes
- Bot API 4.1, 4.2, 4.3 support
- Python 3.4 is no longer supported
- Error Handler now handles all types of exceptions (#1485)
- Return UTC from `from_timestamp()` (#1485)

See the wiki page at <https://github.com/python-telegram-bot/python-telegram-bot/wiki/Transition-guide-to-Version-12.0> for a detailed guide on how to migrate from version 11 to version 12.

Context based callbacks (#1100)

- Use of `pass_` in handlers is deprecated.
- Instead use `use_context=True` on `Updater` or `Dispatcher` and change callback from `(bot, update, others...)` to `(update, context)`.
- This also applies to error handlers `Dispatcher.add_error_handler` and `JobQueue` jobs (change `(bot, job)` to `(context)` here).
- For users with custom handlers subclassing `Handler`, this is mostly backwards compatible, but to use the new context based callbacks you need to implement the new `collect_additional_context` method.
- Passing `bot` to `JobQueue.__init__` is deprecated. Use `JobQueue.set_dispatcher` with a dispatcher instead.
- `Dispatcher` makes sure to use a single *CallbackContext* for a entire update. This means that if an update is handled by multiple handlers (by using the group argument), you can add custom arguments to the *CallbackContext* in a lower group handler and use it in higher group handler. NOTE: Never use with `@run_async`, see docs for more info. (#1283)
- If you have custom handlers they will need to be updated to support the changes in this release.
- Update all examples to use context based callbacks.

Persistence (#1017)

- Added `PicklePersistence` and `DictPersistence` for adding persistence to your bots.
- `BasePersistence` can be subclassed for all your persistence needs.
- Add a new example that shows a persistent `ConversationHandler` bot

Handler overhaul (#1114)

- `CommandHandler` now only triggers on actual commands as defined by telegram servers (everything that the clients mark as a tabable link).
- `PrefixHandler` can be used if you need to trigger on prefixes (like all messages starting with a “/” (old `CommandHandler` behaviour) or even custom prefixes like “#” or “!”).

Filter overhaul (#1221)

- `RegexHandler` is deprecated and should be replaced with a `MessageHandler` with a regex filter.
- Use update filters to filter update types instead of arguments (`message_updates`, `channel_post_updates` and `edited_updates`) on the handlers.
- Completely remove `allow_edited` argument - it has been deprecated for a while.
- `data_filters` now exist which allows filters that return data into the callback function. This is how the regex filter is implemented.
- All this means that it no longer possible to use a list of filters in a handler. Use bitwise operators instead!

ConversationHandler

- Remove `run_async_timeout` and `timed_out_behavior` arguments (#1344)
- Replace with `WAITING` constant and behavior from states (#1344)
- Only emit one warning for multiple `CallbackQueryHandlers` in a `ConversationHandler` (#1319)
- Use `warnings.warn` for `ConversationHandler` warnings (#1343)
- Fix unresolvable promises (#1270)

Bug fixes & improvements

- Handlers should be faster due to deduped logic.
- Avoid compiling compiled regex in regex filter. (#1314)
- Add missing `left_chat_member` to `Message.MESSAGE_TYPES` (#1336)
- Make custom timeouts actually work properly (#1330)
- Add convenience classmethods (`from_button`, `from_row` and `from_column`) to `InlineKeyboardMarkup`
- Small typo fix in `setup.py` (#1306)
- Add Conflict error (HTTP error code 409) (#1154)
- Change `MAX_CAPTION_LENGTH` to 1024 (#1262)
- Remove some unnecessary clauses (#1247, #1239)
- Allow filenames without dots in them when sending files (#1228)
- Fix uploading files with unicode filenames (#1214)
- Replace `http.server` with `Tornado` (#1191)
- Allow `SOCKSConnection` to parse username and password from URL (#1211)
- Fix for arguments in `passport/data.py` (#1213)
- Improve message entity parsing by adding `text_mention` (#1206)
- Documentation fixes (#1348, #1397, #1436)
- Merged filters short-circuit (#1350)
- Fix webhook listen with `tornado` (#1383)
- Call `task_done()` on update queue after update processing finished (#1428)
- Fix `send_location()` - latitude may be 0 (#1437)
- Make `MessageEntity` objects comparable (#1465)
- Add prefix to thread names (#1358)

Buf fixes since v12.0.0b1

- Fix setting bot on `ShippingQuery` (#1355)
- Fix `_trigger_timeout()` missing 1 required positional argument: `'job'` (#1367)
- Add missing `message.text` check in `PrefixHandler` `check_update` (#1375)
- Make updates persist even on `DispatcherHandlerStop` (#1463)
- `Dispatcher` force updating persistence object's chat data attribute (#1462)

Internal improvements

- Finally fix our CI builds mostly (too many commits and PRs to list)
- Use multiple bots for CI to improve testing times significantly.
- Allow pypy to fail in CI.
- Remove the last CamelCase CheckUpdate methods from the handlers we missed earlier.
- test_official is now executed in a different job

10.6.46 Version 11.1.0

Released 2018-09-01

Fixes and updates for Telegram Passport: (#1198)

- Fix passport decryption failing at random times
- Added support for middle names.
- Added support for translations for documents
- Add errors for translations for documents
- Added support for requesting names in the language of the user's country of residence
- Replaced the payload parameter with the new parameter nonce
- Add hash to EncryptedPassportElement

10.6.47 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 forsquare_type to Venue, InlineQueryResultVenue, InputVenueMessageContent, and Bot.send_venue. (#1170)
- Add vCard support by adding vcard field to Contact, InlineQueryResultContact, InputContactMessageContent, and Bot.send_contact. (#1166)
- **Support new message entities: CASHTAG and PHONE_NUMBER. (#1179)**
 - Cashtag seems to be things like *\$USD* and *\$GBP*, but it seems telegram doesn't currently send them to bots.
 - Phone number also seems to have limited support for now
- Add Bot.send_animation, add width, height, and duration to Animation, and add Filters.animation. (#1172)

Non Bot API 4.0 changes:

- Minor integer comparison fix (#1147)
- Fix Filters.regex failing on non-text message (#1158)
- Fix ProcessLookupError if process finishes before we kill it (#1126)
- Add t.me links for User, Chat and Message if available and update User.mention_* (#1092)
- Fix mention_markdown/html on py2 (#1112)

10.6.48 Version 10.1.0

Released 2018-05-02

Fixes changing previous behaviour:

- Add urllib3 fix for socks5h support (#1085)
- Fix send_sticker() timeout=20 (#1088)

Fixes:

- Add a caption_entity filter for filtering caption entities (#1068)
- Inputfile encode filenames (#1086)
- InputFile: Fix proper naming of file when reading from subprocess.PIPE (#1079)
- Remove pytest-catchlog from requirements (#1099)
- Documentation fixes (#1061, #1078, #1081, #1096)

10.6.49 Version 10.0.2

Released 2018-04-17

Important fix:

- Handle utf8 decoding errors (#1076)

New features:

- Added Filter.regex (#1028)
- Filters for Category and file types (#1046)
- Added video note filter (#1067)

Fixes:

- Fix in telegram.Message (#1042)
- Make chat_id a positional argument inside shortcut methods of Chat and User classes (#1050)
- Make Bot.full_name return a unicode object. (#1063)
- CommandHandler faster check (#1074)
- Correct documentation of Dispatcher.add_handler (#1071)
- Various small fixes to documentation.

10.6.50 Version 10.0.1

Released 2018-03-05

Fixes:

- Fix conversationhandler timeout (PR #1032)
- Add missing docs utils (PR #912)

10.6.51 Version 10.0.0

Released 2018-03-02

Non backward compatible changes and changed defaults

- JobQueue: Remove deprecated prevent_autostart & put() (PR #1012)
- Bot, Updater: Remove deprecated network_delay (PR #1012)
- Remove deprecated Message.new_chat_member (PR #1012)
- Retry bootstrap phase indefinitely (by default) on network errors (PR #1018)

New Features

- Support v3.6 API (PR #1006)
- User.full_name convinience property (PR #949)
- Add *send_phone_number_to_provider* and *send_email_to_provider* arguments to *send_invoice* (PR #986)
- Bot: Add shortcut methods *reply_{markdown,html}* (PR #827)
- Bot: Add shortcut method *reply_media_group* (PR #994)
- Added *utils.helpers.effective_message_type* (PR #826)
- Bot.get_file now allows passing a file in addition to file_id (PR #963)
- Add *.get_file()* to Audio, Document, PhotoSize, Sticker, Video, VideoNote and Voice (PR #963)
- Add *.send_**() methods to User and Chat (PR #963)
- Get jobs by name (PR #1011)
- Add Message caption html/markdown methods (PR #1013)
- File.download_as_bytearray - new method to get a d/led file as bytearray (PR #1019)
- File.download(): Now returns a meaningful return value (PR #1019)
- Added conversation timeout in ConversationHandler (PR #895)

Changes

- Store bot in PreCheckoutQuery (PR #953)

- Updater: Issue INFO log upon received signal (PR #951)
- JobQueue: Thread safety fixes (PR #977)
- WebhookHandler: Fix exception thrown during error handling (PR #985)
- Explicitly check update.effective_chat in ConversationHandler.check_update (PR #959)
- Updater: Better handling of timeouts during get_updates (PR #1007)
- Remove unnecessary to_dict() (PR #834)
- CommandHandler - ignore strings in entities and “/” followed by whitespace (PR #1020)
- Documentation & style fixes (PR #942, PR #956, PR #962, PR #980, PR #983)

10.6.52 Version 9.0.0

Released 2017-12-08

Breaking changes (possibly)

- Drop support for python 3.3 (PR #930)

New Features

- Support Bot API 3.5 (PR #920)

Changes

- Fix race condition in dispatcher start/stop (#887)
- Log error trace if there is no error handler registered (#694)
- Update examples with consistent string formatting (#870)
- Various changes and improvements to the docs.

10.6.53 Version 8.1.1

Released 2017-10-15

- Fix Commandhandler crashing on single character messages (PR #873).

10.6.54 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 unittest improvements. - Documentation fixes.

10.6.55 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 initialization of Updater (PR #793).
- Catch exceptions in error handler for errors that happen during polling (PR #810).
- For testing we switched to pytest (PR #788).
- Lots of small improvements to our tests and documentation.

10.6.56 Version 7.0.1

Released 2017-07-28

- Fix TypeError exception in RegexHandler (PR #751).
- Small documentation fix (PR #749).

10.6.57 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 connecting to Telegram servers (Issue #720).
- ConversationHandler - 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 unittests.

10.6.58 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 comparison 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

10.7 Contributor Covenant Code of Conduct

10.7.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

10.7.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Publication of any content supporting, justifying or otherwise affiliating with terror and/or hate towards others
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

10.7.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

10.7.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

10.7.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at devs@python-telegram-bot.org. The project team will review and investigate all complaints, and will respond in a way that it deems appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

10.7.6 Attribution

This Code of Conduct is adapted from the [Contributor Covenant](https://www.contributor-covenant.org/version/1/4), version 1.4, available at <https://www.contributor-covenant.org/version/1/4>.

10.8 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.

10.8.1 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 dependencies:

```
$ pip install -r requirements-all.txt
```

5. Install pre-commit hooks:

```
$ pre-commit install
```

10.8.2 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.

10.8.3 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 `Black` for code formatting. Plugins for `Black` exist for some [popular editors](#). You can use those instead of manually formatting everything.
- Please ensure that the code you write is well-tested 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

```
$ pre-commit run -a
```

- To actually make the commit (this will trigger tests style & type checks automatically):

```
$ git add your-file-changed.py
```

- Finally, push it to your GitHub fork, run:

```
$ git push origin your-branch-name
```

4. When your feature is ready to merge, create a pull request.

- Go to your fork on GitHub, select your branch from the dropdown menu, and click “New pull request”.
- Add a descriptive comment explaining the purpose of the branch (e.g. “Add the new API feature to create inline bot queries.”). This will tell the reviewer what the purpose of the branch is.
- Click “Create pull request”. An admin will assign a reviewer to your commit.

5. Address review comments until all reviewers give LGTM (‘looks good to me’).

- When your reviewer has reviewed the code, you’ll get a notification. You’ll need to respond in two ways:
 - Make a new commit addressing the comments you agree with, and push it to the same branch. Ideally, the commit message would explain what the commit does (e.g. “Fix lint error”), but if there are lots of disparate review comments, it’s fine to refer to the original commit message and add something like “(address review comments)”.
 - In order to keep the commit history intact, please avoid squashing or amending history and then force-pushing to the PR. Reviewers often want to look at individual commits.
 - In addition, please reply to each comment. Each reply should be either “Done” or a response explaining why the corresponding suggestion wasn’t implemented. All comments must be resolved before LGTM can be given.
- Resolve any merge conflicts that arise. To resolve conflicts between ‘your-branch-name’ (in your fork) and ‘master’ (in the python-telegram-bot repository), run:

```
$ git checkout your-branch-name
$ git fetch upstream
$ git merge upstream/master
$ ...[fix the conflicts]...
$ ...[make sure the tests pass before committing]...
$ git commit -a
$ git push origin your-branch-name
```

- At the end, the reviewer will merge the pull request.

6. Tidy up! Delete the feature branch from both your local clone and the GitHub repository:

```
$ git branch -D your-branch-name
$ git push origin --delete your-branch-name
```

7. Celebrate. Congratulations, you have contributed to python-telegram-bot!

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.

- Added `.. versionadded:: NEXT.VERSION`, `.. versionchanged:: NEXT.VERSION` or `.. deprecated:: 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](#)
- Added myself alphabetically to `AUTHORS.rst` (optional)
- Added new classes & modules to the docs and all suitable `__all__` s
- Checked the [Stability Policy](#) 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](#)
- Created a PR to remove functionality deprecated in the previous Bot API release ([see here](#))
- New classes:
 - Added `self._id_attrs` and corresponding documentation
 - `__init__` accepts `api_kwargs` as kw-only
- Added new shortcuts:
 - In [Chat](#) & [User](#) for all methods that accept `chat/user_id`
 - In [Message](#) for all methods that accept `chat_id` and `message_id`
 - For new [Message](#) shortcuts: Added `quote` argument if methods accepts `reply_to_message_id`
 - In [CallbackQuery](#) 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
 - Link new and existing constants in docstrings instead of hard-coded numbers and strings
 - Add new message types to [telegram.Message.effective_attachment](#)
 - Added new handlers for new update types
 - * Add the handlers to the warning loop in the [ConversationHandler](#)
 - 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` and `README_RAW.rst` (including the badge), as well as `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 [Message](#)

10.8.4 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 3.9 or above and have the required dependencies:

```
$ pip install -r docs/requirements-docs.txt
```

then run the following from the PTB root directory:

```
$ make -C docs html
```

or, if you don't have make available (e.g. on Windows):

```
$ sphinx-build docs/source docs/build/html
```

Once the process terminates, you can view the built documentation by opening `docs/build/html/index.html` with a browser.

- Add `.. versionadded:: 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.

10.8.5 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.

10.9 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 `requirements-dev.txt` file in the root of the repository.

10.9.1 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=loadgroup
```

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
```

10.9.2 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.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
```

10.9.3 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 same bot is 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`, [21](#)
- `telegram.constants`, [651](#)
- `telegram.error`, [945](#)
- `telegram.ext`, [516](#)
- `telegram.ext.filters`, [589](#)
- `telegram.helpers`, [948](#)
- `telegram.warnings`, [957](#)

Symbols

- `__abs__()` (*telegram.constants.BotCommandLimit method*), 653
- `__abs__()` (*telegram.constants.BotDescriptionLimit method*), 664
- `__abs__()` (*telegram.constants.BotNameLimit method*), 668
- `__abs__()` (*telegram.constants.BulkRequestLimit method*), 672
- `__abs__()` (*telegram.constants.CallbackQueryLimit method*), 676
- `__abs__()` (*telegram.constants.ChatID method*), 693
- `__abs__()` (*telegram.constants.ChatInviteLinkLimit method*), 697
- `__abs__()` (*telegram.constants.ChatLimit method*), 702
- `__abs__()` (*telegram.constants.ChatPhotoSize method*), 712
- `__abs__()` (*telegram.constants.ContactLimit method*), 722
- `__abs__()` (*telegram.constants.CustomEmojiStickerLimit method*), 726
- `__abs__()` (*telegram.constants.DiceLimit method*), 737
- `__abs__()` (*telegram.constants.FileSizeLimit method*), 741
- `__abs__()` (*telegram.constants.FloodLimit method*), 745
- `__abs__()` (*telegram.constants.ForumIconColor method*), 750
- `__abs__()` (*telegram.constants.ForumTopicLimit method*), 754
- `__abs__()` (*telegram.constants.GiveawayLimit method*), 758
- `__abs__()` (*telegram.constants.InlineKeyboardButtonLimit method*), 762
- `__abs__()` (*telegram.constants.InlineKeyboardMarkupLimit method*), 767
- `__abs__()` (*telegram.constants.InlineQueryLimit method*), 771
- `__abs__()` (*telegram.constants.InlineQueryResultLimit method*), 775
- `__abs__()` (*telegram.constants.InlineQueryResultsButtonLimit method*), 786
- `__abs__()` (*telegram.constants.InvoiceLimit method*), 798
- `__abs__()` (*telegram.constants.KeyboardButtonRequestUsersLimit method*), 802
- `__abs__()` (*telegram.constants.LocationLimit method*), 807
- `__abs__()` (*telegram.constants.MediaGroupLimit method*), 817
- `__abs__()` (*telegram.constants.MessageLimit method*), 843
- `__abs__()` (*telegram.constants.PollLimit method*), 870
- `__abs__()` (*telegram.constants.PollingLimit method*), 880
- `__abs__()` (*telegram.constants.ReplyLimit method*), 904
- `__abs__()` (*telegram.constants.StickerLimit method*), 915
- `__abs__()` (*telegram.constants.StickerSetLimit method*), 919
- `__abs__()` (*telegram.constants.UserProfilePhotosLimit method*), 937
- `__abs__()` (*telegram.constants.WebhookLimit method*), 941
- `__add__()` (*telegram.constants.BotCommandLimit method*), 653
- `__add__()` (*telegram.constants.BotCommandScopeType method*), 658
- `__add__()` (*telegram.constants.BotDescriptionLimit method*), 664
- `__add__()` (*telegram.constants.BotNameLimit method*), 668
- `__add__()` (*telegram.constants.BulkRequestLimit method*), 672
- `__add__()` (*telegram.constants.CallbackQueryLimit method*), 676
- `__add__()` (*telegram.constants.ChatAction method*), 681
- `__add__()` (*telegram.constants.ChatBoostSources method*), 687
- `__add__()` (*telegram.constants.ChatID method*), 693
- `__add__()` (*telegram.constants.ChatInviteLinkLimit method*), 697
- `__add__()` (*telegram.constants.ChatLimit method*), 702
- `__add__()` (*telegram.constants.ChatMemberStatus method*), 706
- `__add__()` (*telegram.constants.ChatPhotoSize method*), 712

- `__add__()` (*telegram.constants.ChatType* method), 716
- `__add__()` (*telegram.constants.ContactLimit* method), 722
- `__add__()` (*telegram.constants.CustomEmojiStickerLimit* method), 726
- `__add__()` (*telegram.constants.DiceEmoji* method), 730
- `__add__()` (*telegram.constants.DiceLimit* method), 737
- `__add__()` (*telegram.constants.FileSizeLimit* method), 741
- `__add__()` (*telegram.constants.FloodLimit* method), 745
- `__add__()` (*telegram.constants.ForumIconColor* method), 750
- `__add__()` (*telegram.constants.ForumTopicLimit* method), 754
- `__add__()` (*telegram.constants.GiveawayLimit* method), 758
- `__add__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 762
- `__add__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 767
- `__add__()` (*telegram.constants.InlineQueryLimit* method), 771
- `__add__()` (*telegram.constants.InlineQueryResultLimit* method), 775
- `__add__()` (*telegram.constants.InlineQueryResultType* method), 780
- `__add__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 786
- `__add__()` (*telegram.constants.InputMediaType* method), 791
- `__add__()` (*telegram.constants.InvoiceLimit* method), 798
- `__add__()` (*telegram.constants.KeyboardButtonRequestUser* method), 802
- `__add__()` (*telegram.constants.LocationLimit* method), 807
- `__add__()` (*telegram.constants.MaskPosition* method), 812
- `__add__()` (*telegram.constants.MediaGroupLimit* method), 818
- `__add__()` (*telegram.constants.MenuButtonType* method), 822
- `__add__()` (*telegram.constants.MessageAttachmentType* method), 829
- `__add__()` (*telegram.constants.MessageEntityType* method), 836
- `__add__()` (*telegram.constants.MessageLimit* method), 843
- `__add__()` (*telegram.constants.MessageOriginType* method), 847
- `__add__()` (*telegram.constants.MessageType* method), 858
- `__add__()` (*telegram.constants.ParseMode* method), 864
- `__add__()` (*telegram.constants.PollLimit* method), 870
- `__add__()` (*telegram.constants.PollType* method), 874
- `__add__()` (*telegram.constants.PollingLimit* method), 880
- `__add__()` (*telegram.constants.ReactionEmoji* method), 892
- `__add__()` (*telegram.constants.ReactionType* method), 898
- `__add__()` (*telegram.constants.ReplyLimit* method), 904
- `__add__()` (*telegram.constants.StickerFormat* method), 908
- `__add__()` (*telegram.constants.StickerLimit* method), 915
- `__add__()` (*telegram.constants.StickerSetLimit* method), 920
- `__add__()` (*telegram.constants.StickerType* method), 924
- `__add__()` (*telegram.constants.UpdateType* method), 931
- `__add__()` (*telegram.constants.UserProfilePhotosLimit* method), 937
- `__add__()` (*telegram.constants.WebhookLimit* method), 941
- `__aenter__()` (*telegram.Bot* method), 27
- `__aenter__()` (*telegram.ext.Application* method), 519
- `__aenter__()` (*telegram.ext.BaseUpdateProcessor* method), 547
- `__aenter__()` (*telegram.ext.Updater* method), 570
- `__aenter__()` (*telegram.request.BaseRequest* method), 951
- `__aexit__()` (*telegram.Bot* method), 27
- `__aexit__()` (*telegram.ext.Application* method), 520
- `__aexit__()` (*telegram.ext.BaseUpdateProcessor* method), 547
- `__aexit__()` (*telegram.ext.Updater* method), 570
- `__aexit__()` (*telegram.request.BaseRequest* method), 951
- `__and__()` (*telegram.constants.BotCommandLimit* method), 653
- `__and__()` (*telegram.constants.BotDescriptionLimit* method), 664
- `__and__()` (*telegram.constants.BotNameLimit* method), 668
- `__and__()` (*telegram.constants.BulkRequestLimit* method), 672
- `__and__()` (*telegram.constants.CallbackQueryLimit* method), 676
- `__and__()` (*telegram.constants.ChatID* method), 693
- `__and__()` (*telegram.constants.ChatInviteLinkLimit* method), 697
- `__and__()` (*telegram.constants.ChatLimit* method), 702
- `__and__()` (*telegram.constants.ChatPhotoSize* method), 712
- `__and__()` (*telegram.constants.ContactLimit* method), 722
- `__and__()` (*telegram.constants.CustomEmojiStickerLimit* method), 726

- `method`), 726
- `__and__()` (`telegram.constants.DiceLimit` `method`), 737
- `__and__()` (`telegram.constants.FileSizeLimit` `method`), 741
- `__and__()` (`telegram.constants.FloodLimit` `method`), 746
- `__and__()` (`telegram.constants.ForumIconColor` `method`), 750
- `__and__()` (`telegram.constants.ForumTopicLimit` `method`), 754
- `__and__()` (`telegram.constants.GiveawayLimit` `method`), 758
- `__and__()` (`telegram.constants.InlineKeyboardButtonLimit` `method`), 762
- `__and__()` (`telegram.constants.InlineKeyboardMarkupLimit` `method`), 767
- `__and__()` (`telegram.constants.InlineQueryLimit` `method`), 771
- `__and__()` (`telegram.constants.InlineQueryResultLimit` `method`), 775
- `__and__()` (`telegram.constants.InlineQueryResultsButtonLimit` `method`), 786
- `__and__()` (`telegram.constants.InvoiceLimit` `method`), 798
- `__and__()` (`telegram.constants.KeyboardButtonRequestUsersLimit` `method`), 802
- `__and__()` (`telegram.constants.LocationLimit` `method`), 808
- `__and__()` (`telegram.constants.MediaGroupLimit` `method`), 818
- `__and__()` (`telegram.constants.MessageLimit` `method`), 843
- `__and__()` (`telegram.constants.PollLimit` `method`), 870
- `__and__()` (`telegram.constants.PollingLimit` `method`), 880
- `__and__()` (`telegram.constants.ReplyLimit` `method`), 904
- `__and__()` (`telegram.constants.StickerLimit` `method`), 915
- `__and__()` (`telegram.constants.StickerSetLimit` `method`), 920
- `__and__()` (`telegram.constants.UserProfilePhotosLimit` `method`), 937
- `__and__()` (`telegram.constants.WebhookLimit` `method`), 941
- `__and__()` (`telegram.ext.filters.BaseFilter` `method`), 593
- `__bool__()` (`telegram.MaybeInaccessibleMessage` `method`), 289
- `__bool__()` (`telegram.Message` `method`), 309
- `__bool__()` (`telegram.constants.BotCommandLimit` `method`), 653
- `__bool__()` (`telegram.constants.BotDescriptionLimit` `method`), 664
- `__bool__()` (`telegram.constants.BotNameLimit` `method`), 668
- `__bool__()` (`telegram.constants.BulkRequestLimit` `method`), 672
- `__bool__()` (`telegram.constants.CallbackQueryLimit` `method`), 676
- `__bool__()` (`telegram.constants.ChatID` `method`), 693
- `__bool__()` (`telegram.constants.ChatInviteLinkLimit` `method`), 697
- `__bool__()` (`telegram.constants.ChatLimit` `method`), 702
- `__bool__()` (`telegram.constants.ChatPhotoSize` `method`), 712
- `__bool__()` (`telegram.constants.ContactLimit` `method`), 722
- `__bool__()` (`telegram.constants.CustomEmojiStickerLimit` `method`), 726
- `__bool__()` (`telegram.constants.DiceLimit` `method`), 737
- `__bool__()` (`telegram.constants.FileSizeLimit` `method`), 741
- `__bool__()` (`telegram.constants.FloodLimit` `method`), 746
- `__bool__()` (`telegram.constants.ForumIconColor` `method`), 750
- `__bool__()` (`telegram.constants.ForumTopicLimit` `method`), 754
- `__bool__()` (`telegram.constants.GiveawayLimit` `method`), 758
- `__bool__()` (`telegram.constants.InlineKeyboardButtonLimit` `method`), 762
- `__bool__()` (`telegram.constants.InlineKeyboardMarkupLimit` `method`), 767
- `__bool__()` (`telegram.constants.InlineQueryLimit` `method`), 771
- `__bool__()` (`telegram.constants.InlineQueryResultLimit` `method`), 775
- `__bool__()` (`telegram.constants.InlineQueryResultsButtonLimit` `method`), 786
- `__bool__()` (`telegram.constants.InvoiceLimit` `method`), 798
- `__bool__()` (`telegram.constants.KeyboardButtonRequestUsersLimit` `method`), 802
- `__bool__()` (`telegram.constants.LocationLimit` `method`), 808
- `__bool__()` (`telegram.constants.MediaGroupLimit` `method`), 818
- `__bool__()` (`telegram.constants.MessageLimit` `method`), 843
- `__bool__()` (`telegram.constants.PollLimit` `method`), 870
- `__bool__()` (`telegram.constants.PollingLimit` `method`), 880
- `__bool__()` (`telegram.constants.ReplyLimit` `method`), 904
- `__bool__()` (`telegram.constants.StickerLimit` `method`), 915
- `__bool__()` (`telegram.constants.StickerSetLimit` `method`), 920
- `__bool__()` (`telegram.constants.UserProfilePhotosLimit` `method`), 937

`__bool__()` (*telegram.constants.WebhookLimit method*), 941
`__bot_api_version__` (*in module telegram*), 21
`__bot_api_version_info__` (*in module telegram*), 21
`__ceil__()` (*telegram.constants.BotCommandLimit method*), 654
`__ceil__()` (*telegram.constants.BotDescriptionLimit method*), 664
`__ceil__()` (*telegram.constants.BotNameLimit method*), 668
`__ceil__()` (*telegram.constants.BulkRequestLimit method*), 672
`__ceil__()` (*telegram.constants.CallbackQueryLimit method*), 676
`__ceil__()` (*telegram.constants.ChatID method*), 693
`__ceil__()` (*telegram.constants.ChatInviteLinkLimit method*), 697
`__ceil__()` (*telegram.constants.ChatLimit method*), 702
`__ceil__()` (*telegram.constants.ChatPhotoSize method*), 712
`__ceil__()` (*telegram.constants.ContactLimit method*), 722
`__ceil__()` (*telegram.constants.CustomEmojiStickerLimit method*), 726
`__ceil__()` (*telegram.constants.DiceLimit method*), 737
`__ceil__()` (*telegram.constants.FileSizeLimit method*), 741
`__ceil__()` (*telegram.constants.FloodLimit method*), 746
`__ceil__()` (*telegram.constants.ForumIconColor method*), 750
`__ceil__()` (*telegram.constants.ForumTopicLimit method*), 755
`__ceil__()` (*telegram.constants.GiveawayLimit method*), 758
`__ceil__()` (*telegram.constants.InlineKeyboardButtonLimit method*), 763
`__ceil__()` (*telegram.constants.InlineKeyboardMarkupLimit method*), 767
`__ceil__()` (*telegram.constants.InlineQueryLimit method*), 771
`__ceil__()` (*telegram.constants.InlineQueryResultLimit method*), 775
`__ceil__()` (*telegram.constants.InlineQueryResultsButtonLimit method*), 787
`__ceil__()` (*telegram.constants.InvoiceLimit method*), 798
`__ceil__()` (*telegram.constants.KeyboardButtonRequestUsersLimit method*), 802
`__ceil__()` (*telegram.constants.LocationLimit method*), 808
`__ceil__()` (*telegram.constants.MediaGroupLimit method*), 818
`__ceil__()` (*telegram.constants.MessageLimit method*), 843
`__ceil__()` (*telegram.constants.PollLimit method*), 871
`__ceil__()` (*telegram.constants.PollingLimit method*), 880
`__ceil__()` (*telegram.constants.ReplyLimit method*), 904
`__ceil__()` (*telegram.constants.StickerLimit method*), 915
`__ceil__()` (*telegram.constants.StickerSetLimit method*), 920
`__ceil__()` (*telegram.constants.UserProfilePhotosLimit method*), 937
`__ceil__()` (*telegram.constants.WebhookLimit method*), 941
`__contains__()` (*telegram.constants.BotCommandScopeType method*), 658
`__contains__()` (*telegram.constants.ChatAction method*), 681
`__contains__()` (*telegram.constants.ChatBoostSources method*), 687
`__contains__()` (*telegram.constants.ChatMemberStatus method*), 706
`__contains__()` (*telegram.constants.ChatType method*), 716
`__contains__()` (*telegram.constants.DiceEmoji method*), 731
`__contains__()` (*telegram.constants.InlineQueryResultType method*), 781
`__contains__()` (*telegram.constants.InputMediaType method*), 791
`__contains__()` (*telegram.constants.MaskPosition method*), 812
`__contains__()` (*telegram.constants.MenuButtonType method*), 822
`__contains__()` (*telegram.constants.MessageAttachmentType method*), 829
`__contains__()` (*telegram.constants.MessageEntityType method*), 836
`__contains__()` (*telegram.constants.MessageOriginType method*), 847
`__contains__()` (*telegram.constants.MessageType method*), 858
`__contains__()` (*telegram.constants.ParseMode method*), 864
`__contains__()` (*telegram.constants.PollType method*), 874
`__contains__()` (*telegram.constants.ReactionEmoji method*), 892
`__contains__()` (*telegram.constants.ReactionType*

- `method`), 898
- `__contains__()` (*telegram.constants.StickerFormat* method), 908
- `__contains__()` (*telegram.constants.StickerType* method), 924
- `__contains__()` (*telegram.constants.UpdateType* method), 931
- `__deepcopy__()` (*telegram.Bot* method), 27
- `__deepcopy__()` (*telegram.TelegramObject* method), 373
- `__delattr__()` (*telegram.TelegramObject* method), 373
- `__divmod__()` (*telegram.constants.BotCommandLimit* method), 654
- `__divmod__()` (*telegram.constants.BotDescriptionLimit* method), 664
- `__divmod__()` (*telegram.constants.BotNameLimit* method), 668
- `__divmod__()` (*telegram.constants.BulkRequestLimit* method), 672
- `__divmod__()` (*telegram.constants.CallbackQueryLimit* method), 676
- `__divmod__()` (*telegram.constants.ChatID* method), 693
- `__divmod__()` (*telegram.constants.ChatInviteLinkLimit* method), 698
- `__divmod__()` (*telegram.constants.ChatLimit* method), 702
- `__divmod__()` (*telegram.constants.ChatPhotoSize* method), 712
- `__divmod__()` (*telegram.constants.ContactLimit* method), 722
- `__divmod__()` (*telegram.constants.CustomEmojiStickerLimit* method), 726
- `__divmod__()` (*telegram.constants.DiceLimit* method), 737
- `__divmod__()` (*telegram.constants.FileSizeLimit* method), 741
- `__divmod__()` (*telegram.constants.FloodLimit* method), 746
- `__divmod__()` (*telegram.constants.ForumIconColor* method), 750
- `__divmod__()` (*telegram.constants.ForumTopicLimit* method), 755
- `__divmod__()` (*telegram.constants.GiveawayLimit* method), 759
- `__divmod__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 763
- `__divmod__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 767
- `__divmod__()` (*telegram.constants.InlineQueryLimit* method), 771
- `__divmod__()` (*telegram.constants.InlineQueryResultLimit* method), 775
- `__divmod__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 787
- `__divmod__()` (*telegram.constants.InvoiceLimit* method), 798
- `__divmod__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 802
- `__divmod__()` (*telegram.constants.LocationLimit* method), 808
- `__divmod__()` (*telegram.constants.MediaGroupLimit* method), 818
- `__divmod__()` (*telegram.constants.MessageLimit* method), 843
- `__divmod__()` (*telegram.constants.PollLimit* method), 871
- `__divmod__()` (*telegram.constants.PollingLimit* method), 880
- `__divmod__()` (*telegram.constants.ReplyLimit* method), 904
- `__divmod__()` (*telegram.constants.StickerLimit* method), 915
- `__divmod__()` (*telegram.constants.StickerSetLimit* method), 920
- `__divmod__()` (*telegram.constants.UserProfilePhotosLimit* method), 937
- `__divmod__()` (*telegram.constants.WebhookLimit* method), 941
- `__eq__()` (*telegram.Bot* method), 27
- `__eq__()` (*telegram.TelegramObject* method), 373
- `__eq__()` (*telegram.constants.BotCommandLimit* method), 654
- `__eq__()` (*telegram.constants.BotCommandScopeType* method), 658
- `__eq__()` (*telegram.constants.BotDescriptionLimit* method), 664
- `__eq__()` (*telegram.constants.BotNameLimit* method), 668
- `__eq__()` (*telegram.constants.BulkRequestLimit* method), 672
- `__eq__()` (*telegram.constants.CallbackQueryLimit* method), 676
- `__eq__()` (*telegram.constants.ChatAction* method), 681
- `__eq__()` (*telegram.constants.ChatBoostSources* method), 687
- `__eq__()` (*telegram.constants.ChatID* method), 693
- `__eq__()` (*telegram.constants.ChatInviteLinkLimit* method), 698
- `__eq__()` (*telegram.constants.ChatLimit* method), 702
- `__eq__()` (*telegram.constants.ChatMemberStatus* method), 706
- `__eq__()` (*telegram.constants.ChatPhotoSize* method), 712

[712](#)
__eq__() (telegram.constants.ChatType method), [716](#)
__eq__() (telegram.constants.ContactLimit method), [722](#)
__eq__() (telegram.constants.CustomEmojiStickerLimit method), [726](#)
__eq__() (telegram.constants.DiceEmoji method), [731](#)
__eq__() (telegram.constants.DiceLimit method), [737](#)
__eq__() (telegram.constants.FileSizeLimit method), [742](#)
__eq__() (telegram.constants.FloodLimit method), [746](#)
__eq__() (telegram.constants.ForumIconColor method), [750](#)
__eq__() (telegram.constants.ForumTopicLimit method), [755](#)
__eq__() (telegram.constants.GiveawayLimit method), [759](#)
__eq__() (telegram.constants.InlineKeyboardButtonLimit method), [763](#)
__eq__() (telegram.constants.InlineKeyboardMarkupLimit method), [767](#)
__eq__() (telegram.constants.InlineQueryLimit method), [771](#)
__eq__() (telegram.constants.InlineQueryResultLimit method), [776](#)
__eq__() (telegram.constants.InlineQueryResultType method), [781](#)
__eq__() (telegram.constants.InlineQueryResultsButtonLimit method), [787](#)
__eq__() (telegram.constants.InputMediaType method), [791](#)
__eq__() (telegram.constants.InvoiceLimit method), [798](#)
__eq__() (telegram.constants.KeyboardButtonRequestUserData method), [802](#)
__eq__() (telegram.constants.LocationLimit method), [808](#)
__eq__() (telegram.constants.MaskPosition method), [812](#)
__eq__() (telegram.constants.MediaGroupLimit method), [818](#)
__eq__() (telegram.constants.MenuButtonType method), [822](#)
__eq__() (telegram.constants.MessageAttachmentType method), [829](#)
__eq__() (telegram.constants.MessageEntityType method), [836](#)
__eq__() (telegram.constants.MessageLimit method), [843](#)
__eq__() (telegram.constants.MessageOriginType method), [847](#)
__eq__() (telegram.constants.MessageType method), [858](#)
__eq__() (telegram.constants.ParseMode method), [864](#)
__eq__() (telegram.constants.PollLimit method), [871](#)
__eq__() (telegram.constants.PollType method), [874](#)
__eq__() (telegram.constants.PollingLimit method), [880](#)
__eq__() (telegram.constants.ReactionEmoji method), [892](#)
__eq__() (telegram.constants.ReactionType method), [898](#)
__eq__() (telegram.constants.ReplyLimit method), [904](#)
__eq__() (telegram.constants.StickerFormat method), [908](#)
__eq__() (telegram.constants.StickerLimit method), [915](#)
__eq__() (telegram.constants.StickerSetLimit method), [920](#)
__eq__() (telegram.constants.StickerType method), [924](#)
__eq__() (telegram.constants.UpdateType method), [931](#)
__eq__() (telegram.constants.UserProfilePhotosLimit method), [937](#)
__eq__() (telegram.constants.WebhookLimit method), [941](#)
__eq__() (telegram.ext.Defaults method), [555](#)
__eq__() (telegram.ext.Job method), [560](#)
__float__() (telegram.constants.BotCommandLimit method), [654](#)
__float__() (telegram.constants.BotDescriptionLimit method), [664](#)
__float__() (telegram.constants.BotNameLimit method), [668](#)
__float__() (telegram.constants.BulkRequestLimit method), [672](#)
__float__() (telegram.constants.CallbackQueryLimit method), [676](#)
__float__() (telegram.constants.ChatID method), [693](#)
__float__() (telegram.constants.ChatInviteLinkLimit method), [698](#)
__float__() (telegram.constants.ChatLimit method), [702](#)
__float__() (telegram.constants.ChatPhotoSize method), [712](#)
__float__() (telegram.constants.ContactLimit method), [722](#)
__float__() (telegram.constants.CustomEmojiStickerLimit method), [726](#)
__float__() (telegram.constants.DiceLimit method), [737](#)
__float__() (telegram.constants.FileSizeLimit method), [742](#)
__float__() (telegram.constants.FloodLimit method), [746](#)
__float__() (telegram.constants.ForumIconColor method), [750](#)
__float__() (telegram.constants.ForumTopicLimit method), [755](#)
__float__() (telegram.constants.GiveawayLimit method), [759](#)

<code>__float__()</code> (<code>telegram.constants.InlineKeyboardButtonLimit</code> method), 763	<code>__float__()</code> (<code>telegram.constants.FileSizeLimit</code> method), 742
<code>__float__()</code> (<code>telegram.constants.InlineKeyboardMarkupLimit</code> method), 767	<code>__float__()</code> (<code>telegram.constants.FloodLimit</code> method), 746
<code>__float__()</code> (<code>telegram.constants.InlineQueryLimit</code> method), 771	<code>__float__()</code> (<code>telegram.constants.ForumIconColor</code> method), 750
<code>__float__()</code> (<code>telegram.constants.InlineQueryResultLimit</code> method), 776	<code>__float__()</code> (<code>telegram.constants.ForumTopicLimit</code> method), 755
<code>__float__()</code> (<code>telegram.constants.InlineQueryResultsButtonLimit</code> method), 787	<code>__float__()</code> (<code>telegram.constants.GiveawayLimit</code> method), 759
<code>__float__()</code> (<code>telegram.constants.InvoiceLimit</code> method), 798	<code>__float__()</code> (<code>telegram.constants.InlineKeyboardButtonLimit</code> method), 763
<code>__float__()</code> (<code>telegram.constants.KeyboardButtonRequestUsersLimit</code> method), 802	<code>__float__()</code> (<code>telegram.constants.InlineKeyboardMarkupLimit</code> method), 767
<code>__float__()</code> (<code>telegram.constants.LocationLimit</code> method), 808	<code>__float__()</code> (<code>telegram.constants.InlineQueryLimit</code> method), 771
<code>__float__()</code> (<code>telegram.constants.MediaGroupLimit</code> method), 818	<code>__float__()</code> (<code>telegram.constants.InlineQueryResultLimit</code> method), 776
<code>__float__()</code> (<code>telegram.constants.MessageLimit</code> method), 843	<code>__float__()</code> (<code>telegram.constants.InlineQueryResultsButtonLimit</code> method), 787
<code>__float__()</code> (<code>telegram.constants.PollLimit</code> method), 871	<code>__float__()</code> (<code>telegram.constants.InvoiceLimit</code> method), 798
<code>__float__()</code> (<code>telegram.constants.PollingLimit</code> method), 880	<code>__float__()</code> (<code>telegram.constants.KeyboardButtonRequestUsersLimit</code> method), 802
<code>__float__()</code> (<code>telegram.constants.ReplyLimit</code> method), 904	<code>__float__()</code> (<code>telegram.constants.LocationLimit</code> method), 808
<code>__float__()</code> (<code>telegram.constants.StickerLimit</code> method), 915	<code>__float__()</code> (<code>telegram.constants.MediaGroupLimit</code> method), 818
<code>__float__()</code> (<code>telegram.constants.StickerSetLimit</code> method), 920	<code>__float__()</code> (<code>telegram.constants.MessageLimit</code> method), 843
<code>__float__()</code> (<code>telegram.constants.UserProfilePhotosLimit</code> method), 937	<code>__float__()</code> (<code>telegram.constants.PollLimit</code> method), 871
<code>__float__()</code> (<code>telegram.constants.WebhookLimit</code> method), 941	<code>__float__()</code> (<code>telegram.constants.PollingLimit</code> method), 881
<code>__floor__()</code> (<code>telegram.constants.BotCommandLimit</code> method), 654	<code>__float__()</code> (<code>telegram.constants.ReplyLimit</code> method), 904
<code>__floor__()</code> (<code>telegram.constants.BotDescriptionLimit</code> method), 664	<code>__float__()</code> (<code>telegram.constants.StickerLimit</code> method), 915
<code>__floor__()</code> (<code>telegram.constants.BotNameLimit</code> method), 668	<code>__float__()</code> (<code>telegram.constants.StickerSetLimit</code> method), 920
<code>__floor__()</code> (<code>telegram.constants.BulkRequestLimit</code> method), 672	<code>__float__()</code> (<code>telegram.constants.UserProfilePhotosLimit</code> method), 937
<code>__floor__()</code> (<code>telegram.constants.CallbackQueryLimit</code> method), 676	<code>__float__()</code> (<code>telegram.constants.WebhookLimit</code> method), 942
<code>__floor__()</code> (<code>telegram.constants.ChatID</code> method), 693	<code>__floordiv__()</code> (<code>telegram.constants.BotCommandLimit</code> method), 654
<code>__floor__()</code> (<code>telegram.constants.ChatInviteLinkLimit</code> method), 698	<code>__floordiv__()</code> (<code>telegram.constants.BotDescriptionLimit</code> method), 664
<code>__floor__()</code> (<code>telegram.constants.ChatLimit</code> method), 702	<code>__floordiv__()</code> (<code>telegram.constants.BotNameLimit</code> method), 668
<code>__floor__()</code> (<code>telegram.constants.ChatPhotoSize</code> method), 712	<code>__floordiv__()</code> (<code>telegram.constants.BulkRequestLimit</code> method), 672
<code>__floor__()</code> (<code>telegram.constants.ContactLimit</code> method), 722	<code>__floordiv__()</code> (<code>telegram.constants.CallbackQueryLimit</code> method), 676
<code>__floor__()</code> (<code>telegram.constants.CustomEmojiStickerLimit</code> method), 726	
<code>__floor__()</code> (<code>telegram.constants.DiceLimit</code> method), 737	

`__floordiv__()` (*telegram.constants.ChatID* method), 693

`__floordiv__()` (*telegram.constants.ChatInviteLinkLimit* method), 698

`__floordiv__()` (*telegram.constants.ChatLimit* method), 702

`__floordiv__()` (*telegram.constants.ChatPhotoSize* method), 712

`__floordiv__()` (*telegram.constants.ContactLimit* method), 722

`__floordiv__()` (*telegram.constants.CustomEmojiStickerLimit* method), 726

`__floordiv__()` (*telegram.constants.DiceLimit* method), 737

`__floordiv__()` (*telegram.constants.FileSizeLimit* method), 742

`__floordiv__()` (*telegram.constants.FloodLimit* method), 746

`__floordiv__()` (*telegram.constants.ForumIconColor* method), 750

`__floordiv__()` (*telegram.constants.ForumTopicLimit* method), 755

`__floordiv__()` (*telegram.constants.GiveawayLimit* method), 759

`__floordiv__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 763

`__floordiv__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 767

`__floordiv__()` (*telegram.constants.InlineQueryLimit* method), 772

`__floordiv__()` (*telegram.constants.InlineQueryResultLimit* method), 776

`__floordiv__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 787

`__floordiv__()` (*telegram.constants.InvoiceLimit* method), 798

`__floordiv__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 802

`__floordiv__()` (*telegram.constants.LocationLimit* method), 808

`__floordiv__()` (*telegram.constants.MediaGroupLimit* method), 818

`__floordiv__()` (*telegram.constants.MessageLimit* method), 843

`__floordiv__()` (*telegram.constants.PollLimit* method), 871

`__floordiv__()` (*telegram.constants.PollingLimit* method), 881

`__floordiv__()` (*telegram.constants.ReplyLimit* method), 904

`__floordiv__()` (*telegram.constants.StickerLimit* method), 915

`__floordiv__()` (*telegram.constants.StickerSetLimit* method), 920

`__floordiv__()` (*telegram.constants.UserProfilePhotosLimit* method), 937

`__floordiv__()` (*telegram.constants.WebhookLimit* method), 942

`__format__()` (*telegram.constants.BotCommandLimit* method), 654

`__format__()` (*telegram.constants.BotCommandScopeType* method), 658

`__format__()` (*telegram.constants.BotDescriptionLimit* method), 664

`__format__()` (*telegram.constants.BotNameLimit* method), 668

`__format__()` (*telegram.constants.BulkRequestLimit* method), 672

`__format__()` (*telegram.constants.CallbackQueryLimit* method), 676

`__format__()` (*telegram.constants.ChatAction* method), 681

`__format__()` (*telegram.constants.ChatBoostSources* method), 687

`__format__()` (*telegram.constants.ChatID* method), 693

`__format__()` (*telegram.constants.ChatInviteLinkLimit* method), 698

`__format__()` (*telegram.constants.ChatLimit* method), 702

`__format__()` (*telegram.constants.ChatMemberStatus* method), 706

`__format__()` (*telegram.constants.ChatPhotoSize* method), 712

`__format__()` (*telegram.constants.ChatType* method), 716

`__format__()` (*telegram.constants.ContactLimit* method), 722

`__format__()` (*telegram.constants.CustomEmojiStickerLimit* method), 726

`__format__()` (*telegram.constants.DiceEmoji* method), 731

`__format__()` (*telegram.constants.DiceLimit* method), 737

`__format__()` (*telegram.constants.FileSizeLimit* method), 742

`__format__()` (*telegram.constants.FloodLimit* method), 746

- `method`), 746
- `__format__()` (`telegram.constants.ForumIconColor` method), 750
- `__format__()` (`telegram.constants.ForumTopicLimit` method), 755
- `__format__()` (`telegram.constants.GiveawayLimit` method), 759
- `__format__()` (`telegram.constants.InlineKeyboardButtonLimit` method), 763
- `__format__()` (`telegram.constants.InlineKeyboardMarkupLimit` method), 767
- `__format__()` (`telegram.constants.InlineQueryLimit` method), 772
- `__format__()` (`telegram.constants.InlineQueryResultLimit` method), 776
- `__format__()` (`telegram.constants.InlineQueryResultType` method), 781
- `__format__()` (`telegram.constants.InlineQueryResultsButtonLimit` method), 787
- `__format__()` (`telegram.constants.InputMediaType` method), 791
- `__format__()` (`telegram.constants.InvoiceLimit` method), 798
- `__format__()` (`telegram.constants.KeyboardButtonRequestUsersLimit` method), 802
- `__format__()` (`telegram.constants.LocationLimit` method), 808
- `__format__()` (`telegram.constants.MaskPosition` method), 812
- `__format__()` (`telegram.constants.MediaGroupLimit` method), 818
- `__format__()` (`telegram.constants.MenuButtonType` method), 822
- `__format__()` (`telegram.constants.MessageAttachmentType` method), 829
- `__format__()` (`telegram.constants.MessageEntityType` method), 836
- `__format__()` (`telegram.constants.MessageLimit` method), 843
- `__format__()` (`telegram.constants.MessageOriginType` method), 847
- `__format__()` (`telegram.constants.MessageType` method), 858
- `__format__()` (`telegram.constants.ParseMode` method), 864
- `__format__()` (`telegram.constants.PollLimit` method), 871
- `__format__()` (`telegram.constants.PollType` method), 875
- `__format__()` (`telegram.constants.PollingLimit` method), 881
- `__format__()` (`telegram.constants.ReactionEmoji` method), 892
- `__format__()` (`telegram.constants.ReactionType` method), 898
- `__format__()` (`telegram.constants.ReplyLimit` method), 904
- `__format__()` (`telegram.constants.StickerFormat` method), 908
- `__format__()` (`telegram.constants.StickerLimit` method), 915
- `__format__()` (`telegram.constants.StickerSetLimit` method), 920
- `__format__()` (`telegram.constants.StickerType` method), 924
- `__format__()` (`telegram.constants.UpdateType` method), 931
- `__format__()` (`telegram.constants.UserProfilePhotosLimit` method), 937
- `__format__()` (`telegram.constants.WebhookLimit` method), 942
- `__ge__()` (`telegram.constants.BotCommandLimit` method), 654
- `__ge__()` (`telegram.constants.BotCommandScopeType` method), 658
- `__ge__()` (`telegram.constants.BotDescriptionLimit` method), 664
- `__ge__()` (`telegram.constants.BotNameLimit` method), 668
- `__ge__()` (`telegram.constants.BulkRequestLimit` method), 672
- `__ge__()` (`telegram.constants.CallbackQueryLimit` method), 676
- `__ge__()` (`telegram.constants.ChatAction` method), 681
- `__ge__()` (`telegram.constants.ChatBoostSources` method), 687
- `__ge__()` (`telegram.constants.ChatID` method), 693
- `__ge__()` (`telegram.constants.ChatInviteLinkLimit` method), 698
- `__ge__()` (`telegram.constants.ChatLimit` method), 702
- `__ge__()` (`telegram.constants.ChatMemberStatus` method), 706
- `__ge__()` (`telegram.constants.ChatPhotoSize` method), 712
- `__ge__()` (`telegram.constants.ChatType` method), 716
- `__ge__()` (`telegram.constants.ContactLimit` method), 722
- `__ge__()` (`telegram.constants.CustomEmojiStickerLimit` method), 727
- `__ge__()` (`telegram.constants.DiceEmoji` method), 731
- `__ge__()` (`telegram.constants.DiceLimit` method), 737
- `__ge__()` (`telegram.constants.FileSizeLimit` method), 742
- `__ge__()` (`telegram.constants.FloodLimit` method), 746

<code>__ge__()</code> (<i>telegram.constants.ForumIconColor</i> method), 751	<code>__ge__()</code> (<i>telegram.constants.StickerSetLimit</i> method), 920
<code>__ge__()</code> (<i>telegram.constants.ForumTopicLimit</i> method), 755	<code>__ge__()</code> (<i>telegram.constants.StickerType</i> method), 924
<code>__ge__()</code> (<i>telegram.constants.GiveawayLimit</i> method), 759	<code>__ge__()</code> (<i>telegram.constants.UpdateType</i> method), 931
<code>__ge__()</code> (<i>telegram.constants.InlineKeyboardButtonLimit</i> method), 763	<code>__ge__()</code> (<i>telegram.constants.UserProfilePhotosLimit</i> method), 937
<code>__ge__()</code> (<i>telegram.constants.InlineKeyboardMarkupLimit</i> method), 767	<code>__ge__()</code> (<i>telegram.constants.WebhookLimit</i> method), 942
<code>__ge__()</code> (<i>telegram.constants.InlineQueryLimit</i> method), 772	<code>__getattr__()</code> (<i>telegram.ext.Job</i> method), 560
<code>__ge__()</code> (<i>telegram.constants.InlineQueryResultLimit</i> method), 776	<code>__getattribute__()</code> (<i>telegram.constants.BotCommandLimit</i> method), 654
<code>__ge__()</code> (<i>telegram.constants.InlineQueryResultType</i> method), 781	<code>__getattribute__()</code> (<i>telegram.constants.BotCommandScopeType</i> method), 658
<code>__ge__()</code> (<i>telegram.constants.InlineQueryResultsButtonLimit</i> method), 787	<code>__getattribute__()</code> (<i>telegram.constants.BotDescriptionLimit</i> method), 664
<code>__ge__()</code> (<i>telegram.constants.InputMediaType</i> method), 791	<code>__getattribute__()</code> (<i>telegram.constants.BotNameLimit</i> method), 668
<code>__ge__()</code> (<i>telegram.constants.InvoiceLimit</i> method), 798	<code>__getattribute__()</code> (<i>telegram.constants.BulkRequestLimit</i> method), 672
<code>__ge__()</code> (<i>telegram.constants.KeyboardButtonRequestUsersLimit</i> method), 802	<code>__getattribute__()</code> (<i>telegram.constants.CallbackQueryLimit</i> method), 676
<code>__ge__()</code> (<i>telegram.constants.LocationLimit</i> method), 808	<code>__getattribute__()</code> (<i>telegram.constants.ChatAction</i> method), 681
<code>__ge__()</code> (<i>telegram.constants.MaskPosition</i> method), 812	<code>__getattribute__()</code> (<i>telegram.constants.ChatBoostSources</i> method), 687
<code>__ge__()</code> (<i>telegram.constants.MediaGroupLimit</i> method), 818	<code>__getattribute__()</code> (<i>telegram.constants.ChatID</i> method), 694
<code>__ge__()</code> (<i>telegram.constants.MenuButtonType</i> method), 822	<code>__getattribute__()</code> (<i>telegram.constants.ChatInviteLinkLimit</i> method), 698
<code>__ge__()</code> (<i>telegram.constants.MessageAttachmentType</i> method), 829	<code>__getattribute__()</code> (<i>telegram.constants.ChatLimit</i> method), 702
<code>__ge__()</code> (<i>telegram.constants.MessageEntityType</i> method), 836	<code>__getattribute__()</code> (<i>telegram.constants.ChatMemberStatus</i> method), 706
<code>__ge__()</code> (<i>telegram.constants.MessageLimit</i> method), 843	<code>__getattribute__()</code> (<i>telegram.constants.ChatPhotoSize</i> method), 712
<code>__ge__()</code> (<i>telegram.constants.MessageOriginType</i> method), 847	<code>__getattribute__()</code> (<i>telegram.constants.ChatType</i> method), 716
<code>__ge__()</code> (<i>telegram.constants.MessageType</i> method), 858	<code>__getattribute__()</code> (<i>telegram.constants.ContactLimit</i> method), 722
<code>__ge__()</code> (<i>telegram.constants.ParseMode</i> method), 864	<code>__getattribute__()</code> (<i>telegram.constants.CustomEmojiStickerLimit</i> method), 727
<code>__ge__()</code> (<i>telegram.constants.PollLimit</i> method), 871	<code>__getattribute__()</code> (<i>telegram.constants.DiceEmoji</i> method), 731
<code>__ge__()</code> (<i>telegram.constants.PollType</i> method), 875	<code>__getattribute__()</code> (<i>telegram.constants.DiceLimit</i> method), 731
<code>__ge__()</code> (<i>telegram.constants.PollingLimit</i> method), 881	
<code>__ge__()</code> (<i>telegram.constants.ReactionEmoji</i> method), 892	
<code>__ge__()</code> (<i>telegram.constants.ReactionType</i> method), 898	
<code>__ge__()</code> (<i>telegram.constants.ReplyLimit</i> method), 904	
<code>__ge__()</code> (<i>telegram.constants.StickerFormat</i> method), 908	
<code>__ge__()</code> (<i>telegram.constants.StickerLimit</i> method), 915	

method), 737

`__getattr__()` (*telegram.constants.FileSizeLimit* method), 742

`__getattr__()` (*telegram.constants.FloodLimit* method), 746

`__getattr__()` (*telegram.constants.ForumIconColor* method), 751

`__getattr__()` (*telegram.constants.ForumTopicLimit* method), 755

`__getattr__()` (*telegram.constants.GiveawayLimit* method), 759

`__getattr__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 763

`__getattr__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 767

`__getattr__()` (*telegram.constants.InlineQueryLimit* method), 772

`__getattr__()` (*telegram.constants.InlineQueryResultLimit* method), 776

`__getattr__()` (*telegram.constants.InlineQueryResultType* method), 781

`__getattr__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 787

`__getattr__()` (*telegram.constants.InputMediaType* method), 791

`__getattr__()` (*telegram.constants.InvoiceLimit* method), 798

`__getattr__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 802

`__getattr__()` (*telegram.constants.LocationLimit* method), 808

`__getattr__()` (*telegram.constants.MaskPosition* method), 812

`__getattr__()` (*telegram.constants.MediaGroupLimit* method), 818

`__getattr__()` (*telegram.constants.MenuButtonType* method), 822

`__getattr__()` (*telegram.constants.MessageAttachmentType* method), 829

`__getattr__()` (*telegram.constants.MessageEntityType* method), 837

`__getattr__()` (*telegram.constants.MessageLimit* method), 843

`__getattr__()` (*telegram.constants.MessageOriginType* method), 847

`__getattr__()` (*telegram.constants.MessageType* method), 858

`__getattr__()` (*telegram.constants.ParseMode* method), 864

`__getattr__()` (*telegram.constants.PollLimit* method), 871

`__getattr__()` (*telegram.constants.PollType* method), 875

`__getattr__()` (*telegram.constants.PollingLimit* method), 881

`__getattr__()` (*telegram.constants.ReactionEmoji* method), 893

`__getattr__()` (*telegram.constants.ReactionType* method), 898

`__getattr__()` (*telegram.constants.ReplyLimit* method), 904

`__getattr__()` (*telegram.constants.StickerFormat* method), 908

`__getattr__()` (*telegram.constants.StickerLimit* method), 915

`__getattr__()` (*telegram.constants.StickerSetLimit* method), 920

`__getattr__()` (*telegram.constants.StickerType* method), 924

`__getattr__()` (*telegram.constants.UpdateType* method), 931

`__getattr__()` (*telegram.constants.UserProfilePhotosLimit* method), 937

`__getattr__()` (*telegram.constants.WebhookLimit* method), 942

`__getitem__()` (*telegram.TelegramObject* method), 373

`__getitem__()` (*telegram.constants.BotCommandScopeType* method), 658

`__getitem__()` (*telegram.constants.ChatAction* method), 681

`__getitem__()` (*telegram.constants.ChatBoostSources* method), 687

`__getitem__()` (*telegram.constants.ChatMemberStatus* method),

[706](#)
`__getitem__()` (*telegram.constants.ChatType* method), [716](#)
`__getitem__()` (*telegram.constants.DiceEmoji* method), [731](#)
`__getitem__()` (*telegram.constants.InlineQueryResultType* method), [781](#)
`__getitem__()` (*telegram.constants.InputMediaType* method), [791](#)
`__getitem__()` (*telegram.constants.MaskPosition* method), [812](#)
`__getitem__()` (*telegram.constants.MenuButtonType* method), [822](#)
`__getitem__()` (*telegram.constants.MessageAttachmentType* method), [829](#)
`__getitem__()` (*telegram.constants.MessageEntityType* method), [837](#)
`__getitem__()` (*telegram.constants.MessageOriginType* method), [847](#)
`__getitem__()` (*telegram.constants.MessageType* method), [858](#)
`__getitem__()` (*telegram.constants.ParseMode* method), [864](#)
`__getitem__()` (*telegram.constants.PollType* method), [875](#)
`__getitem__()` (*telegram.constants.ReactionEmoji* method), [893](#)
`__getitem__()` (*telegram.constants.ReactionType* method), [898](#)
`__getitem__()` (*telegram.constants.StickerFormat* method), [909](#)
`__getitem__()` (*telegram.constants.StickerType* method), [924](#)
`__getitem__()` (*telegram.constants.UpdateType* method), [931](#)
`__getstate__()` (*telegram.TelegramObject* method), [374](#)
`__gt__()` (*telegram.constants.BotCommandLimit* method), [654](#)
`__gt__()` (*telegram.constants.BotCommandScopeType* method), [658](#)
`__gt__()` (*telegram.constants.BotDescriptionLimit* method), [664](#)
`__gt__()` (*telegram.constants.BotNameLimit* method), [668](#)
`__gt__()` (*telegram.constants.BulkRequestLimit* method), [672](#)
`__gt__()` (*telegram.constants.CallbackQueryLimit* method), [676](#)
`__gt__()` (*telegram.constants.ChatAction* method), [681](#)
`__gt__()` (*telegram.constants.ChatBoostSources* method), [687](#)
`__gt__()` (*telegram.constants.ChatID* method), [694](#)
`__gt__()` (*telegram.constants.ChatInviteLinkLimit* method), [698](#)
`__gt__()` (*telegram.constants.ChatLimit* method), [702](#)
`__gt__()` (*telegram.constants.ChatMemberStatus* method), [706](#)
`__gt__()` (*telegram.constants.ChatPhotoSize* method), [712](#)
`__gt__()` (*telegram.constants.ChatType* method), [717](#)
`__gt__()` (*telegram.constants.ContactLimit* method), [723](#)
`__gt__()` (*telegram.constants.CustomEmojiStickerLimit* method), [727](#)
`__gt__()` (*telegram.constants.DiceEmoji* method), [731](#)
`__gt__()` (*telegram.constants.DiceLimit* method), [737](#)
`__gt__()` (*telegram.constants.FileSizeLimit* method), [742](#)
`__gt__()` (*telegram.constants.FloodLimit* method), [746](#)
`__gt__()` (*telegram.constants.ForumIconColor* method), [751](#)
`__gt__()` (*telegram.constants.ForumTopicLimit* method), [755](#)
`__gt__()` (*telegram.constants.GiveawayLimit* method), [759](#)
`__gt__()` (*telegram.constants.InlineKeyboardButtonLimit* method), [763](#)
`__gt__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), [767](#)
`__gt__()` (*telegram.constants.InlineQueryLimit* method), [772](#)
`__gt__()` (*telegram.constants.InlineQueryResultLimit* method), [776](#)
`__gt__()` (*telegram.constants.InlineQueryResultType* method), [781](#)
`__gt__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), [787](#)
`__gt__()` (*telegram.constants.InputMediaType* method), [791](#)
`__gt__()` (*telegram.constants.InvoiceLimit* method), [798](#)
`__gt__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), [802](#)
`__gt__()` (*telegram.constants.LocationLimit* method), [808](#)
`__gt__()` (*telegram.constants.MaskPosition* method), [812](#)
`__gt__()` (*telegram.constants.MediaGroupLimit* method), [818](#)
`__gt__()` (*telegram.constants.MenuButtonType* method), [822](#)
`__gt__()` (*telegram.constants.MessageAttachmentType* method), [829](#)
`__gt__()` (*telegram.constants.MessageEntityType* method), [837](#)
`__gt__()` (*telegram.constants.MessageLimit* method), [843](#)
`__gt__()` (*telegram.constants.MessageOriginType* method), [847](#)

- `__gt__()` (*telegram.constants.MessageType* method), 858
- `__gt__()` (*telegram.constants.ParseMode* method), 864
- `__gt__()` (*telegram.constants.PollLimit* method), 871
- `__gt__()` (*telegram.constants.PollType* method), 875
- `__gt__()` (*telegram.constants.PollingLimit* method), 881
- `__gt__()` (*telegram.constants.ReactionEmoji* method), 893
- `__gt__()` (*telegram.constants.ReactionType* method), 898
- `__gt__()` (*telegram.constants.ReplyLimit* method), 905
- `__gt__()` (*telegram.constants.StickerFormat* method), 909
- `__gt__()` (*telegram.constants.StickerLimit* method), 915
- `__gt__()` (*telegram.constants.StickerSetLimit* method), 920
- `__gt__()` (*telegram.constants.StickerType* method), 924
- `__gt__()` (*telegram.constants.UpdateType* method), 931
- `__gt__()` (*telegram.constants.UserProfilePhotosLimit* method), 937
- `__gt__()` (*telegram.constants.WebhookLimit* method), 942
- `__hash__()` (*telegram.Bot* method), 27
- `__hash__()` (*telegram.TelegramObject* method), 374
- `__hash__()` (*telegram.constants.BotCommandLimit* method), 654
- `__hash__()` (*telegram.constants.BotCommandScopeType* method), 658
- `__hash__()` (*telegram.constants.BotDescriptionLimit* method), 664
- `__hash__()` (*telegram.constants.BotNameLimit* method), 668
- `__hash__()` (*telegram.constants.BulkRequestLimit* method), 672
- `__hash__()` (*telegram.constants.CallbackQueryLimit* method), 676
- `__hash__()` (*telegram.constants.ChatAction* method), 681
- `__hash__()` (*telegram.constants.ChatBoostSources* method), 687
- `__hash__()` (*telegram.constants.ChatID* method), 694
- `__hash__()` (*telegram.constants.ChatInviteLinkLimit* method), 698
- `__hash__()` (*telegram.constants.ChatLimit* method), 702
- `__hash__()` (*telegram.constants.ChatMemberStatus* method), 706
- `__hash__()` (*telegram.constants.ChatPhotoSize* method), 712
- `__hash__()` (*telegram.constants.ChatType* method), 717
- `__hash__()` (*telegram.constants.ContactLimit* method), 723
- `__hash__()` (*telegram.constants.CustomEmojiStickerLimit* method), 727
- `__hash__()` (*telegram.constants.DiceEmoji* method), 731
- `__hash__()` (*telegram.constants.DiceLimit* method), 737
- `__hash__()` (*telegram.constants.FileSizeLimit* method), 742
- `__hash__()` (*telegram.constants.FloodLimit* method), 746
- `__hash__()` (*telegram.constants.ForumIconColor* method), 751
- `__hash__()` (*telegram.constants.ForumTopicLimit* method), 755
- `__hash__()` (*telegram.constants.GiveawayLimit* method), 759
- `__hash__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 763
- `__hash__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 767
- `__hash__()` (*telegram.constants.InlineQueryLimit* method), 772
- `__hash__()` (*telegram.constants.InlineQueryResultLimit* method), 776
- `__hash__()` (*telegram.constants.InlineQueryResultType* method), 781
- `__hash__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 787
- `__hash__()` (*telegram.constants.InputMediaType* method), 791
- `__hash__()` (*telegram.constants.InvoiceLimit* method), 798
- `__hash__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 802
- `__hash__()` (*telegram.constants.LocationLimit* method), 808
- `__hash__()` (*telegram.constants.MaskPosition* method), 812
- `__hash__()` (*telegram.constants.MediaGroupLimit* method), 818
- `__hash__()` (*telegram.constants.MenuButtonType* method), 822
- `__hash__()` (*telegram.constants.MessageAttachmentType* method), 829
- `__hash__()` (*telegram.constants.MessageEntityType* method), 837
- `__hash__()` (*telegram.constants.MessageLimit* method), 844
- `__hash__()` (*telegram.constants.MessageOriginType* method), 847
- `__hash__()` (*telegram.constants.MessageType* method), 858
- `__hash__()` (*telegram.constants.ParseMode* method), 864
- `__hash__()` (*telegram.constants.PollLimit* method), 871
- `__hash__()` (*telegram.constants.PollType* method), 875

875
__hash__() (telegram.constants.PollingLimit method), 881
__hash__() (telegram.constants.ReactionEmoji method), 893
__hash__() (telegram.constants.ReactionType method), 899
__hash__() (telegram.constants.ReplyLimit method), 905
__hash__() (telegram.constants.StickerFormat method), 909
__hash__() (telegram.constants.StickerLimit method), 915
__hash__() (telegram.constants.StickerSetLimit method), 920
__hash__() (telegram.constants.StickerType method), 924
__hash__() (telegram.constants.UpdateType method), 931
__hash__() (telegram.constants.UserProfilePhotosLimit method), 937
__hash__() (telegram.constants.WebhookLimit method), 942
__hash__() (telegram.ext.Defaults method), 555
__hash__() (telegram.ext.Job method), 561
__index__() (telegram.constants.BotCommandLimit method), 654
__index__() (telegram.constants.BotDescriptionLimit method), 664
__index__() (telegram.constants.BotNameLimit method), 668
__index__() (telegram.constants.BulkRequestLimit method), 672
__index__() (telegram.constants.CallbackQueryLimit method), 676
__index__() (telegram.constants.ChatID method), 694
__index__() (telegram.constants.ChatInviteLinkLimit method), 698
__index__() (telegram.constants.ChatLimit method), 702
__index__() (telegram.constants.ChatPhotoSize method), 713
__index__() (telegram.constants.ContactLimit method), 723
__index__() (telegram.constants.CustomEmojiStickerLimit method), 727
__index__() (telegram.constants.DiceLimit method), 737
__index__() (telegram.constants.FileSizeLimit method), 742
__index__() (telegram.constants.FloodLimit method), 746
__index__() (telegram.constants.ForumIconColor method), 751
__index__() (telegram.constants.ForumTopicLimit method), 755
__index__() (telegram.constants.GiveawayLimit method), 759
__index__() (telegram.constants.InlineKeyboardButtonLimit method), 763
__index__() (telegram.constants.InlineKeyboardMarkupLimit method), 767
__index__() (telegram.constants.InlineQueryLimit method), 772
__index__() (telegram.constants.InlineQueryResultLimit method), 776
__index__() (telegram.constants.InlineQueryResultsButtonLimit method), 787
__index__() (telegram.constants.InvoiceLimit method), 798
__index__() (telegram.constants.KeyboardButtonRequestUsersLimit method), 802
__index__() (telegram.constants.LocationLimit method), 808
__index__() (telegram.constants.MediaGroupLimit method), 818
__index__() (telegram.constants.MessageLimit method), 844
__index__() (telegram.constants.PollLimit method), 871
__index__() (telegram.constants.PollingLimit method), 881
__index__() (telegram.constants.ReplyLimit method), 905
__index__() (telegram.constants.StickerLimit method), 915
__index__() (telegram.constants.StickerSetLimit method), 920
__index__() (telegram.constants.UserProfilePhotosLimit method), 938
__index__() (telegram.constants.WebhookLimit method), 942
__int__() (telegram.constants.BotCommandLimit method), 654
__int__() (telegram.constants.BotDescriptionLimit method), 664
__int__() (telegram.constants.BotNameLimit method), 668
__int__() (telegram.constants.BulkRequestLimit method), 672
__int__() (telegram.constants.CallbackQueryLimit method), 676
__int__() (telegram.constants.ChatID method), 694
__int__() (telegram.constants.ChatInviteLinkLimit method), 698
__int__() (telegram.constants.ChatLimit method), 702
__int__() (telegram.constants.ChatPhotoSize method), 713
__int__() (telegram.constants.ContactLimit method), 723
__int__() (telegram.constants.CustomEmojiStickerLimit method), 727
__int__() (telegram.constants.DiceLimit method), 738

- `__int__()` (*telegram.constants.FileSizeLimit* method), 742
- `__int__()` (*telegram.constants.FloodLimit* method), 746
- `__int__()` (*telegram.constants.ForumIconColor* method), 751
- `__int__()` (*telegram.constants.ForumTopicLimit* method), 755
- `__int__()` (*telegram.constants.GiveawayLimit* method), 759
- `__int__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 763
- `__int__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 767
- `__int__()` (*telegram.constants.InlineQueryLimit* method), 772
- `__int__()` (*telegram.constants.InlineQueryResultLimit* method), 776
- `__int__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 787
- `__int__()` (*telegram.constants.InvoiceLimit* method), 798
- `__int__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 802
- `__int__()` (*telegram.constants.LocationLimit* method), 808
- `__int__()` (*telegram.constants.MediaGroupLimit* method), 818
- `__int__()` (*telegram.constants.MessageLimit* method), 844
- `__int__()` (*telegram.constants.PollLimit* method), 871
- `__int__()` (*telegram.constants.PollingLimit* method), 881
- `__int__()` (*telegram.constants.ReplyLimit* method), 905
- `__int__()` (*telegram.constants.StickerLimit* method), 915
- `__int__()` (*telegram.constants.StickerSetLimit* method), 920
- `__int__()` (*telegram.constants.UserProfilePhotosLimit* method), 938
- `__int__()` (*telegram.constants.WebhookLimit* method), 942
- `__invert__()` (*telegram.constants.BotCommandLimit* method), 654
- `__invert__()` (*telegram.constants.BotDescriptionLimit* method), 664
- `__invert__()` (*telegram.constants.BotNameLimit* method), 668
- `__invert__()` (*telegram.constants.BulkRequestLimit* method), 672
- `__invert__()` (*telegram.constants.CallbackQueryLimit* method), 677
- `__invert__()` (*telegram.constants.ChatID* method), 694
- `__invert__()` (*telegram.constants.ChatInviteLinkLimit* method), 698
- `__invert__()` (*telegram.constants.ChatLimit* method), 702
- `__invert__()` (*telegram.constants.ChatPhotoSize* method), 713
- `__invert__()` (*telegram.constants.ContactLimit* method), 723
- `__invert__()` (*telegram.constants.CustomEmojiStickerLimit* method), 727
- `__invert__()` (*telegram.constants.DiceLimit* method), 738
- `__invert__()` (*telegram.constants.FileSizeLimit* method), 742
- `__invert__()` (*telegram.constants.FloodLimit* method), 746
- `__invert__()` (*telegram.constants.ForumIconColor* method), 751
- `__invert__()` (*telegram.constants.ForumTopicLimit* method), 755
- `__invert__()` (*telegram.constants.GiveawayLimit* method), 759
- `__invert__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 763
- `__invert__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 767
- `__invert__()` (*telegram.constants.InlineQueryLimit* method), 772
- `__invert__()` (*telegram.constants.InlineQueryResultLimit* method), 776
- `__invert__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 787
- `__invert__()` (*telegram.constants.InvoiceLimit* method), 798
- `__invert__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 803
- `__invert__()` (*telegram.constants.LocationLimit* method), 808
- `__invert__()` (*telegram.constants.MediaGroupLimit* method), 818
- `__invert__()` (*telegram.constants.MessageLimit* method), 844
- `__invert__()` (*telegram.constants.PollLimit* method), 871
- `__invert__()` (*telegram.constants.PollingLimit* method), 881
- `__invert__()` (*telegram.constants.ReplyLimit* method), 905
- `__invert__()` (*telegram.constants.StickerLimit* method), 915
- `__invert__()` (*telegram.constants.StickerSetLimit* method), 920

method), 920
__invert__() (telegram.constants.UserProfilePhotosLimit method), 938
__invert__() (telegram.constants.WebhookLimit method), 942
__invert__() (telegram.ext.filters.BaseFilter method), 593
__iter__() (telegram.constants.BotCommandScopeType method), 658
__iter__() (telegram.constants.ChatAction method), 681
__iter__() (telegram.constants.ChatBoostSources method), 687
__iter__() (telegram.constants.ChatMemberStatus method), 706
__iter__() (telegram.constants.ChatType method), 717
__iter__() (telegram.constants.DiceEmoji method), 731
__iter__() (telegram.constants.InlineQueryResultType method), 781
__iter__() (telegram.constants.InputMediaType method), 791
__iter__() (telegram.constants.MaskPosition method), 812
__iter__() (telegram.constants.MenuButtonType method), 822
__iter__() (telegram.constants.MessageAttachmentType method), 829
__iter__() (telegram.constants.MessageEntityType method), 837
__iter__() (telegram.constants.MessageOriginType method), 848
__iter__() (telegram.constants.MessageType method), 858
__iter__() (telegram.constants.ParseMode method), 864
__iter__() (telegram.constants.PollType method), 875
__iter__() (telegram.constants.ReactionEmoji method), 893
__iter__() (telegram.constants.ReactionType method), 899
__iter__() (telegram.constants.StickerFormat method), 909
__iter__() (telegram.constants.StickerType method), 924
__iter__() (telegram.constants.UpdateType method), 931
__le__() (telegram.constants.BotCommandLimit method), 654
__le__() (telegram.constants.BotCommandScopeType method), 658
__le__() (telegram.constants.BotDescriptionLimit method), 664
__le__() (telegram.constants.BotNameLimit method), 668
__le__() (telegram.constants.BulkRequestLimit method), 673
__le__() (telegram.constants.CallbackQueryLimit method), 677
__le__() (telegram.constants.ChatAction method), 681
__le__() (telegram.constants.ChatBoostSources method), 687
__le__() (telegram.constants.ChatID method), 694
__le__() (telegram.constants.ChatInviteLinkLimit method), 698
__le__() (telegram.constants.ChatLimit method), 702
__le__() (telegram.constants.ChatMemberStatus method), 706
__le__() (telegram.constants.ChatPhotoSize method), 713
__le__() (telegram.constants.ChatType method), 717
__le__() (telegram.constants.ContactLimit method), 723
__le__() (telegram.constants.CustomEmojiStickerLimit method), 727
__le__() (telegram.constants.DiceEmoji method), 731
__le__() (telegram.constants.DiceLimit method), 738
__le__() (telegram.constants.FileSizeLimit method), 742
__le__() (telegram.constants.FloodLimit method), 746
__le__() (telegram.constants.ForumIconColor method), 751
__le__() (telegram.constants.ForumTopicLimit method), 755
__le__() (telegram.constants.GiveawayLimit method), 759
__le__() (telegram.constants.InlineKeyboardButtonLimit method), 763
__le__() (telegram.constants.InlineKeyboardMarkupLimit method), 767
__le__() (telegram.constants.InlineQueryLimit method), 772
__le__() (telegram.constants.InlineQueryResultLimit method), 776
__le__() (telegram.constants.InlineQueryResultType method), 781
__le__() (telegram.constants.InlineQueryResultsButtonLimit method), 787
__le__() (telegram.constants.InputMediaType method), 791
__le__() (telegram.constants.InvoiceLimit method), 798
__le__() (telegram.constants.KeyboardButtonRequestUsersLimit method), 803
__le__() (telegram.constants.LocationLimit method), 808
__le__() (telegram.constants.MaskPosition method), 812
__le__() (telegram.constants.MediaGroupLimit method), 818
__le__() (telegram.constants.MenuButtonType

- [method\), 822](#)
- [__le__\(\) \(telegram.constants.MessageAttachmentType method\), 829](#)
- [__le__\(\) \(telegram.constants.MessageEntityType method\), 837](#)
- [__le__\(\) \(telegram.constants.MessageLimit method\), 844](#)
- [__le__\(\) \(telegram.constants.MessageOriginType method\), 848](#)
- [__le__\(\) \(telegram.constants.MessageType method\), 858](#)
- [__le__\(\) \(telegram.constants.ParseMode method\), 864](#)
- [__le__\(\) \(telegram.constants.PollLimit method\), 871](#)
- [__le__\(\) \(telegram.constants.PollType method\), 875](#)
- [__le__\(\) \(telegram.constants.PollingLimit method\), 881](#)
- [__le__\(\) \(telegram.constants.ReactionEmoji method\), 893](#)
- [__le__\(\) \(telegram.constants.ReactionType method\), 899](#)
- [__le__\(\) \(telegram.constants.ReplyLimit method\), 905](#)
- [__le__\(\) \(telegram.constants.StickerFormat method\), 909](#)
- [__le__\(\) \(telegram.constants.StickerLimit method\), 916](#)
- [__le__\(\) \(telegram.constants.StickerSetLimit method\), 920](#)
- [__le__\(\) \(telegram.constants.StickerType method\), 924](#)
- [__le__\(\) \(telegram.constants.UpdateType method\), 932](#)
- [__le__\(\) \(telegram.constants.UserProfilePhotosLimit method\), 938](#)
- [__le__\(\) \(telegram.constants.WebhookLimit method\), 942](#)
- [__len__\(\) \(telegram.constants.BotCommandScopeType method\), 658](#)
- [__len__\(\) \(telegram.constants.ChatAction method\), 681](#)
- [__len__\(\) \(telegram.constants.ChatBoostSources method\), 687](#)
- [__len__\(\) \(telegram.constants.ChatMemberStatus method\), 706](#)
- [__len__\(\) \(telegram.constants.ChatType method\), 717](#)
- [__len__\(\) \(telegram.constants.DiceEmoji method\), 731](#)
- [__len__\(\) \(telegram.constants.InlineQueryResultType method\), 781](#)
- [__len__\(\) \(telegram.constants.InputMediaType method\), 791](#)
- [__len__\(\) \(telegram.constants.MaskPosition method\), 812](#)
- [__len__\(\) \(telegram.constants.MenuButtonType method\), 822](#)
- [__len__\(\) \(telegram.constants.MessageAttachmentType method\), 829](#)
- [__len__\(\) \(telegram.constants.MessageEntityType method\), 837](#)
- [__len__\(\) \(telegram.constants.MessageOriginType method\), 848](#)
- [__len__\(\) \(telegram.constants.MessageType method\), 858](#)
- [__len__\(\) \(telegram.constants.ParseMode method\), 864](#)
- [__len__\(\) \(telegram.constants.PollType method\), 875](#)
- [__len__\(\) \(telegram.constants.ReactionEmoji method\), 893](#)
- [__len__\(\) \(telegram.constants.ReactionType method\), 899](#)
- [__len__\(\) \(telegram.constants.StickerFormat method\), 909](#)
- [__len__\(\) \(telegram.constants.StickerType method\), 924](#)
- [__len__\(\) \(telegram.constants.UpdateType method\), 932](#)
- [__lshift__\(\) \(telegram.constants.BotCommandLimit method\), 654](#)
- [__lshift__\(\) \(telegram.constants.BotDescriptionLimit method\), 664](#)
- [__lshift__\(\) \(telegram.constants.BotNameLimit method\), 668](#)
- [__lshift__\(\) \(telegram.constants.BulkRequestLimit method\), 673](#)
- [__lshift__\(\) \(telegram.constants.CallbackQueryLimit method\), 677](#)
- [__lshift__\(\) \(telegram.constants.ChatID method\), 694](#)
- [__lshift__\(\) \(telegram.constants.ChatInviteLinkLimit method\), 698](#)
- [__lshift__\(\) \(telegram.constants.ChatLimit method\), 702](#)
- [__lshift__\(\) \(telegram.constants.ChatPhotoSize method\), 713](#)
- [__lshift__\(\) \(telegram.constants.ContactLimit method\), 723](#)
- [__lshift__\(\) \(telegram.constants.CustomEmojiStickerLimit method\), 727](#)
- [__lshift__\(\) \(telegram.constants.DiceLimit method\), 738](#)
- [__lshift__\(\) \(telegram.constants.FileSizeLimit method\), 742](#)
- [__lshift__\(\) \(telegram.constants.FloodLimit method\), 746](#)
- [__lshift__\(\) \(telegram.constants.ForumIconColor method\), 751](#)
- [__lshift__\(\) \(telegram.constants.ForumTopicLimit method\), 755](#)
- [__lshift__\(\) \(telegram.constants.GiveawayLimit](#)

method), 759
__lshift__() (telegram.constants.InlineKeyboardButtonLimit method), 763
__lshift__() (telegram.constants.InlineKeyboardMarkupLimit method), 768
__lshift__() (telegram.constants.InlineQueryLimit method), 772
__lshift__() (telegram.constants.InlineQueryResultLimit method), 776
__lshift__() (telegram.constants.InlineQueryResultsButtonLimit method), 787
__lshift__() (telegram.constants.InvoiceLimit method), 798
__lshift__() (telegram.constants.KeyboardButtonRequestUsersLimit method), 803
__lshift__() (telegram.constants.LocationLimit method), 808
__lshift__() (telegram.constants.MediaGroupLimit method), 818
__lshift__() (telegram.constants.MessageLimit method), 844
__lshift__() (telegram.constants.PollLimit method), 871
__lshift__() (telegram.constants.PollingLimit method), 881
__lshift__() (telegram.constants.ReplyLimit method), 905
__lshift__() (telegram.constants.StickerLimit method), 916
__lshift__() (telegram.constants.StickerSetLimit method), 920
__lshift__() (telegram.constants.UserProfilePhotosLimit method), 938
__lshift__() (telegram.constants.WebhookLimit method), 942
__lt__() (telegram.constants.BotCommandLimit method), 654
__lt__() (telegram.constants.BotCommandScopeType method), 658
__lt__() (telegram.constants.BotDescriptionLimit method), 665
__lt__() (telegram.constants.BotNameLimit method), 668
__lt__() (telegram.constants.BulkRequestLimit method), 673
__lt__() (telegram.constants.CallbackQueryLimit method), 677
__lt__() (telegram.constants.ChatAction method), 681
__lt__() (telegram.constants.ChatBoostSources method), 687
__lt__() (telegram.constants.ChatID method), 694
__lt__() (telegram.constants.ChatInviteLinkLimit method), 698
__lt__() (telegram.constants.ChatLimit method), 703
__lt__() (telegram.constants.ChatMemberStatus method), 707
__lt__() (telegram.constants.ChatPhotoSize method), 713
__lt__() (telegram.constants.ChatType method), 717
__lt__() (telegram.constants.ContactLimit method), 723
__lt__() (telegram.constants.CustomEmojiStickerLimit method), 727
__lt__() (telegram.constants.DiceEmoji method), 731
__lt__() (telegram.constants.DiceLimit method), 738
__lt__() (telegram.constants.FileSizeLimit method), 742
__lt__() (telegram.constants.FloodLimit method), 746
__lt__() (telegram.constants.ForumIconColor method), 751
__lt__() (telegram.constants.ForumTopicLimit method), 755
__lt__() (telegram.constants.GiveawayLimit method), 759
__lt__() (telegram.constants.InlineKeyboardButtonLimit method), 763
__lt__() (telegram.constants.InlineKeyboardMarkupLimit method), 768
__lt__() (telegram.constants.InlineQueryLimit method), 772
__lt__() (telegram.constants.InlineQueryResultLimit method), 776
__lt__() (telegram.constants.InlineQueryResultType method), 781
__lt__() (telegram.constants.InlineQueryResultsButtonLimit method), 787
__lt__() (telegram.constants.InputMediaType method), 791
__lt__() (telegram.constants.InvoiceLimit method), 799
__lt__() (telegram.constants.KeyboardButtonRequestUsersLimit method), 803
__lt__() (telegram.constants.LocationLimit method), 808
__lt__() (telegram.constants.MaskPosition method), 812
__lt__() (telegram.constants.MediaGroupLimit method), 818
__lt__() (telegram.constants.MenuButtonType method), 822
__lt__() (telegram.constants.MessageAttachmentType method), 830
__lt__() (telegram.constants.MessageEntityType method), 837
__lt__() (telegram.constants.MessageLimit method), 844
__lt__() (telegram.constants.MessageOriginType method), 848

`__lt__()` (*telegram.constants.MessageType* method), 858
`__lt__()` (*telegram.constants.ParseMode* method), 864
`__lt__()` (*telegram.constants.PollLimit* method), 871
`__lt__()` (*telegram.constants.PollType* method), 875
`__lt__()` (*telegram.constants.PollingLimit* method), 881
`__lt__()` (*telegram.constants.ReactionEmoji* method), 893
`__lt__()` (*telegram.constants.ReactionType* method), 899
`__lt__()` (*telegram.constants.ReplyLimit* method), 905
`__lt__()` (*telegram.constants.StickerFormat* method), 909
`__lt__()` (*telegram.constants.StickerLimit* method), 916
`__lt__()` (*telegram.constants.StickerSetLimit* method), 920
`__lt__()` (*telegram.constants.StickerType* method), 924
`__lt__()` (*telegram.constants.UpdateType* method), 932
`__lt__()` (*telegram.constants.UserProfilePhotosLimit* method), 938
`__lt__()` (*telegram.constants.WebhookLimit* method), 942
`__mod__()` (*telegram.constants.BotCommandLimit* method), 654
`__mod__()` (*telegram.constants.BotCommandScopeType* method), 658
`__mod__()` (*telegram.constants.BotDescriptionLimit* method), 665
`__mod__()` (*telegram.constants.BotNameLimit* method), 669
`__mod__()` (*telegram.constants.BulkRequestLimit* method), 673
`__mod__()` (*telegram.constants.CallbackQueryLimit* method), 677
`__mod__()` (*telegram.constants.ChatAction* method), 681
`__mod__()` (*telegram.constants.ChatBoostSources* method), 687
`__mod__()` (*telegram.constants.ChatID* method), 694
`__mod__()` (*telegram.constants.ChatInviteLinkLimit* method), 698
`__mod__()` (*telegram.constants.ChatLimit* method), 703
`__mod__()` (*telegram.constants.ChatMemberStatus* method), 707
`__mod__()` (*telegram.constants.ChatPhotoSize* method), 713
`__mod__()` (*telegram.constants.ChatType* method), 717
`__mod__()` (*telegram.constants.ContactLimit* method), 723
`__mod__()` (*telegram.constants.CustomEmojiStickerLimit* method), 727
`__mod__()` (*telegram.constants.DiceEmoji* method), 731
`__mod__()` (*telegram.constants.DiceLimit* method), 738
`__mod__()` (*telegram.constants.FileSizeLimit* method), 742
`__mod__()` (*telegram.constants.FloodLimit* method), 746
`__mod__()` (*telegram.constants.ForumIconColor* method), 751
`__mod__()` (*telegram.constants.ForumTopicLimit* method), 755
`__mod__()` (*telegram.constants.GiveawayLimit* method), 759
`__mod__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 763
`__mod__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 768
`__mod__()` (*telegram.constants.InlineQueryLimit* method), 772
`__mod__()` (*telegram.constants.InlineQueryResultLimit* method), 776
`__mod__()` (*telegram.constants.InlineQueryResultType* method), 781
`__mod__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 787
`__mod__()` (*telegram.constants.InputMediaType* method), 791
`__mod__()` (*telegram.constants.InvoiceLimit* method), 799
`__mod__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 803
`__mod__()` (*telegram.constants.LocationLimit* method), 808
`__mod__()` (*telegram.constants.MaskPosition* method), 812
`__mod__()` (*telegram.constants.MediaGroupLimit* method), 818
`__mod__()` (*telegram.constants.MenuButtonType* method), 822
`__mod__()` (*telegram.constants.MessageAttachmentType* method), 830
`__mod__()` (*telegram.constants.MessageEntityType* method), 837
`__mod__()` (*telegram.constants.MessageLimit* method), 844
`__mod__()` (*telegram.constants.MessageOriginType* method), 848
`__mod__()` (*telegram.constants.MessageType* method), 858
`__mod__()` (*telegram.constants.ParseMode* method), 864
`__mod__()` (*telegram.constants.PollLimit* method), 871
`__mod__()` (*telegram.constants.PollType* method), 875
`__mod__()` (*telegram.constants.PollingLimit* method), 881
`__mod__()` (*telegram.constants.ReactionEmoji* method), 893

method), 893

`__mod__()` (*telegram.constants.ReactionType* method), 899

`__mod__()` (*telegram.constants.ReplyLimit* method), 905

`__mod__()` (*telegram.constants.StickerFormat* method), 909

`__mod__()` (*telegram.constants.StickerLimit* method), 916

`__mod__()` (*telegram.constants.StickerSetLimit* method), 920

`__mod__()` (*telegram.constants.StickerType* method), 924

`__mod__()` (*telegram.constants.UpdateType* method), 932

`__mod__()` (*telegram.constants.UserProfilePhotosLimit* method), 938

`__mod__()` (*telegram.constants.WebhookLimit* method), 942

`__mul__()` (*telegram.constants.BotCommandLimit* method), 654

`__mul__()` (*telegram.constants.BotCommandScopeType* method), 659

`__mul__()` (*telegram.constants.BotDescriptionLimit* method), 665

`__mul__()` (*telegram.constants.BotNameLimit* method), 669

`__mul__()` (*telegram.constants.BulkRequestLimit* method), 673

`__mul__()` (*telegram.constants.CallbackQueryLimit* method), 677

`__mul__()` (*telegram.constants.ChatAction* method), 681

`__mul__()` (*telegram.constants.ChatBoostSources* method), 687

`__mul__()` (*telegram.constants.ChatID* method), 694

`__mul__()` (*telegram.constants.ChatInviteLinkLimit* method), 698

`__mul__()` (*telegram.constants.ChatLimit* method), 703

`__mul__()` (*telegram.constants.ChatMemberStatus* method), 707

`__mul__()` (*telegram.constants.ChatPhotoSize* method), 713

`__mul__()` (*telegram.constants.ChatType* method), 717

`__mul__()` (*telegram.constants.ContactLimit* method), 723

`__mul__()` (*telegram.constants.CustomEmojiStickerLimit* method), 727

`__mul__()` (*telegram.constants.DiceEmoji* method), 731

`__mul__()` (*telegram.constants.DiceLimit* method), 738

`__mul__()` (*telegram.constants.FileSizeLimit* method), 742

`__mul__()` (*telegram.constants.FloodLimit* method), 746

`__mul__()` (*telegram.constants.ForumIconColor* method), 751

`__mul__()` (*telegram.constants.ForumTopicLimit* method), 755

`__mul__()` (*telegram.constants.GiveawayLimit* method), 759

`__mul__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 763

`__mul__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 768

`__mul__()` (*telegram.constants.InlineQueryLimit* method), 772

`__mul__()` (*telegram.constants.InlineQueryResultLimit* method), 776

`__mul__()` (*telegram.constants.InlineQueryResultType* method), 781

`__mul__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 787

`__mul__()` (*telegram.constants.InputMediaType* method), 791

`__mul__()` (*telegram.constants.InvoiceLimit* method), 799

`__mul__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 803

`__mul__()` (*telegram.constants.LocationLimit* method), 808

`__mul__()` (*telegram.constants.MaskPosition* method), 812

`__mul__()` (*telegram.constants.MediaGroupLimit* method), 819

`__mul__()` (*telegram.constants.MenuButtonType* method), 822

`__mul__()` (*telegram.constants.MessageAttachmentType* method), 830

`__mul__()` (*telegram.constants.MessageEntityType* method), 837

`__mul__()` (*telegram.constants.MessageLimit* method), 844

`__mul__()` (*telegram.constants.MessageOriginType* method), 848

`__mul__()` (*telegram.constants.MessageType* method), 858

`__mul__()` (*telegram.constants.ParseMode* method), 864

`__mul__()` (*telegram.constants.PollLimit* method), 871

`__mul__()` (*telegram.constants.PollType* method), 875

`__mul__()` (*telegram.constants.PollingLimit* method), 881

`__mul__()` (*telegram.constants.ReactionEmoji* method), 893

`__mul__()` (*telegram.constants.ReactionType* method), 899

`__mul__()` (*telegram.constants.ReplyLimit* method), 905

`__mul__()` (*telegram.constants.StickerFormat* method), 909

`__mul__()` (*telegram.constants.StickerLimit* method), 916

`__mul__()` (*telegram.constants.StickerSetLimit method*), 921
`__mul__()` (*telegram.constants.StickerType method*), 924
`__mul__()` (*telegram.constants.UpdateType method*), 932
`__mul__()` (*telegram.constants.UserProfilePhotosLimit method*), 938
`__mul__()` (*telegram.constants.WebhookLimit method*), 942
`__ne__()` (*telegram.constants.BotCommandLimit method*), 654
`__ne__()` (*telegram.constants.BotCommandScopeType method*), 659
`__ne__()` (*telegram.constants.BotDescriptionLimit method*), 665
`__ne__()` (*telegram.constants.BotNameLimit method*), 669
`__ne__()` (*telegram.constants.BulkRequestLimit method*), 673
`__ne__()` (*telegram.constants.CallbackQueryLimit method*), 677
`__ne__()` (*telegram.constants.ChatAction method*), 681
`__ne__()` (*telegram.constants.ChatBoostSources method*), 687
`__ne__()` (*telegram.constants.ChatID method*), 694
`__ne__()` (*telegram.constants.ChatInviteLinkLimit method*), 698
`__ne__()` (*telegram.constants.ChatLimit method*), 703
`__ne__()` (*telegram.constants.ChatMemberStatus method*), 707
`__ne__()` (*telegram.constants.ChatPhotoSize method*), 713
`__ne__()` (*telegram.constants.ChatType method*), 717
`__ne__()` (*telegram.constants.ContactLimit method*), 723
`__ne__()` (*telegram.constants.CustomEmojiStickerLimit method*), 727
`__ne__()` (*telegram.constants.DiceEmoji method*), 731
`__ne__()` (*telegram.constants.DiceLimit method*), 738
`__ne__()` (*telegram.constants.FileSizeLimit method*), 742
`__ne__()` (*telegram.constants.FloodLimit method*), 747
`__ne__()` (*telegram.constants.ForumIconColor method*), 751
`__ne__()` (*telegram.constants.ForumTopicLimit method*), 755
`__ne__()` (*telegram.constants.GiveawayLimit method*), 759
`__ne__()` (*telegram.constants.InlineKeyboardButtonLimit method*), 763
`__ne__()` (*telegram.constants.InlineKeyboardMarkupLimit method*), 768
`__ne__()` (*telegram.constants.InlineQueryLimit method*), 772
`__ne__()` (*telegram.constants.InlineQueryResultLimit method*), 776
`__ne__()` (*telegram.constants.InlineQueryResultType method*), 781
`__ne__()` (*telegram.constants.InlineQueryResultsButtonLimit method*), 787
`__ne__()` (*telegram.constants.InputMediaType method*), 791
`__ne__()` (*telegram.constants.InvoiceLimit method*), 799
`__ne__()` (*telegram.constants.KeyboardButtonRequestUsersLimit method*), 803
`__ne__()` (*telegram.constants.LocationLimit method*), 809
`__ne__()` (*telegram.constants.MaskPosition method*), 812
`__ne__()` (*telegram.constants.MediaGroupLimit method*), 819
`__ne__()` (*telegram.constants.MenuButtonType method*), 822
`__ne__()` (*telegram.constants.MessageAttachmentType method*), 830
`__ne__()` (*telegram.constants.MessageEntityType method*), 837
`__ne__()` (*telegram.constants.MessageLimit method*), 844
`__ne__()` (*telegram.constants.MessageOriginType method*), 848
`__ne__()` (*telegram.constants.MessageType method*), 858
`__ne__()` (*telegram.constants.ParseMode method*), 864
`__ne__()` (*telegram.constants.PollLimit method*), 871
`__ne__()` (*telegram.constants.PollType method*), 875
`__ne__()` (*telegram.constants.PollingLimit method*), 881
`__ne__()` (*telegram.constants.ReactionEmoji method*), 893
`__ne__()` (*telegram.constants.ReactionType method*), 899
`__ne__()` (*telegram.constants.ReplyLimit method*), 905
`__ne__()` (*telegram.constants.StickerFormat method*), 909
`__ne__()` (*telegram.constants.StickerLimit method*), 916
`__ne__()` (*telegram.constants.StickerSetLimit method*), 921
`__ne__()` (*telegram.constants.StickerType method*), 924
`__ne__()` (*telegram.constants.UpdateType method*), 932
`__ne__()` (*telegram.constants.UserProfilePhotosLimit method*), 938
`__ne__()` (*telegram.constants.WebhookLimit method*), 942
`__neg__()` (*telegram.constants.BotCommandLimit method*), 654
`__neg__()` (*telegram.constants.BotDescriptionLimit method*), 665

method), 665
__neg__() (telegram.constants.BotNameLimit method), 669
__neg__() (telegram.constants.BulkRequestLimit method), 673
__neg__() (telegram.constants.CallbackQueryLimit method), 677
__neg__() (telegram.constants.ChatID method), 694
__neg__() (telegram.constants.ChatInviteLinkLimit method), 698
__neg__() (telegram.constants.ChatLimit method), 703
__neg__() (telegram.constants.ChatPhotoSize method), 713
__neg__() (telegram.constants.ContactLimit method), 723
__neg__() (telegram.constants.CustomEmojiStickerLimit method), 727
__neg__() (telegram.constants.DiceLimit method), 738
__neg__() (telegram.constants.FileSizeLimit method), 742
__neg__() (telegram.constants.FloodLimit method), 747
__neg__() (telegram.constants.ForumIconColor method), 751
__neg__() (telegram.constants.ForumTopicLimit method), 755
__neg__() (telegram.constants.GiveawayLimit method), 759
__neg__() (telegram.constants.InlineKeyboardButtonLimit method), 763
__neg__() (telegram.constants.InlineKeyboardMarkupLimit method), 768
__neg__() (telegram.constants.InlineQueryLimit method), 772
__neg__() (telegram.constants.InlineQueryResultLimit method), 776
__neg__() (telegram.constants.InlineQueryResultsButtonLimit method), 787
__neg__() (telegram.constants.InvoiceLimit method), 799
__neg__() (telegram.constants.KeyboardButtonRequestUserLimit method), 803
__neg__() (telegram.constants.LocationLimit method), 809
__neg__() (telegram.constants.MediaGroupLimit method), 819
__neg__() (telegram.constants.MessageLimit method), 844
__neg__() (telegram.constants.PollLimit method), 871
__neg__() (telegram.constants.PollingLimit method), 881
__neg__() (telegram.constants.ReplyLimit method), 905
__neg__() (telegram.constants.StickerLimit method), 916
__neg__() (telegram.constants.StickerSetLimit method), 921
__neg__() (telegram.constants.UserProfilePhotosLimit method), 938
__neg__() (telegram.constants.WebhookLimit method), 942
__new__() (telegram.constants.BotCommandLimit method), 655
__new__() (telegram.constants.BotCommandScopeType method), 659
__new__() (telegram.constants.BotDescriptionLimit method), 665
__new__() (telegram.constants.BotNameLimit method), 669
__new__() (telegram.constants.BulkRequestLimit method), 673
__new__() (telegram.constants.CallbackQueryLimit method), 677
__new__() (telegram.constants.ChatAction method), 681
__new__() (telegram.constants.ChatBoostSources method), 687
__new__() (telegram.constants.ChatID method), 694
__new__() (telegram.constants.ChatInviteLinkLimit method), 698
__new__() (telegram.constants.ChatLimit method), 703
__new__() (telegram.constants.ChatMemberStatus method), 707
__new__() (telegram.constants.ChatPhotoSize method), 713
__new__() (telegram.constants.ChatType method), 717
__new__() (telegram.constants.ContactLimit method), 723
__new__() (telegram.constants.CustomEmojiStickerLimit method), 727
__new__() (telegram.constants.DiceEmoji method), 731
__new__() (telegram.constants.DiceLimit method), 738
__new__() (telegram.constants.FileSizeLimit method), 742
__new__() (telegram.constants.FloodLimit method), 747
__new__() (telegram.constants.ForumIconColor method), 751
__new__() (telegram.constants.ForumTopicLimit method), 756
__new__() (telegram.constants.GiveawayLimit method), 759
__new__() (telegram.constants.InlineKeyboardButtonLimit method), 764
__new__() (telegram.constants.InlineKeyboardMarkupLimit method), 768
__new__() (telegram.constants.InlineQueryLimit method), 772
__new__() (telegram.constants.InlineQueryResultLimit method), 776

<code>__new__()</code> (<i>telegram.constants.InlineQueryResultType</i> method), 781	<code>__or__()</code> (<i>telegram.constants.BotNameLimit</i> method), 669
<code>__new__()</code> (<i>telegram.constants.InlineQueryResultsButtonLimit</i> method), 788	<code>__or__()</code> (<i>telegram.constants.BulkRequestLimit</i> method), 673
<code>__new__()</code> (<i>telegram.constants.InputMediaType</i> method), 791	<code>__or__()</code> (<i>telegram.constants.CallbackQueryLimit</i> method), 677
<code>__new__()</code> (<i>telegram.constants.InvoiceLimit</i> method), 799	<code>__or__()</code> (<i>telegram.constants.ChatID</i> method), 694
<code>__new__()</code> (<i>telegram.constants.KeyboardButtonRequestUsersLimit</i> method), 803	<code>__or__()</code> (<i>telegram.constants.ChatInviteLinkLimit</i> method), 699
<code>__new__()</code> (<i>telegram.constants.LocationLimit</i> method), 809	<code>__or__()</code> (<i>telegram.constants.ChatLimit</i> method), 703
<code>__new__()</code> (<i>telegram.constants.MaskPosition</i> method), 812	<code>__or__()</code> (<i>telegram.constants.ChatPhotoSize</i> method), 713
<code>__new__()</code> (<i>telegram.constants.MediaGroupLimit</i> method), 819	<code>__or__()</code> (<i>telegram.constants.ContactLimit</i> method), 723
<code>__new__()</code> (<i>telegram.constants.MenuButtonType</i> method), 822	<code>__or__()</code> (<i>telegram.constants.CustomEmojiStickerLimit</i> method), 727
<code>__new__()</code> (<i>telegram.constants.MessageAttachmentType</i> method), 830	<code>__or__()</code> (<i>telegram.constants.DiceLimit</i> method), 738
<code>__new__()</code> (<i>telegram.constants.MessageEntityType</i> method), 837	<code>__or__()</code> (<i>telegram.constants.FileSizeLimit</i> method), 742
<code>__new__()</code> (<i>telegram.constants.MessageLimit</i> method), 844	<code>__or__()</code> (<i>telegram.constants.FloodLimit</i> method), 747
<code>__new__()</code> (<i>telegram.constants.MessageOriginType</i> method), 848	<code>__or__()</code> (<i>telegram.constants.ForumIconColor</i> method), 751
<code>__new__()</code> (<i>telegram.constants.MessageType</i> method), 858	<code>__or__()</code> (<i>telegram.constants.ForumTopicLimit</i> method), 756
<code>__new__()</code> (<i>telegram.constants.ParseMode</i> method), 864	<code>__or__()</code> (<i>telegram.constants.GiveawayLimit</i> method), 760
<code>__new__()</code> (<i>telegram.constants.PollLimit</i> method), 872	<code>__or__()</code> (<i>telegram.constants.InlineKeyboardButtonLimit</i> method), 764
<code>__new__()</code> (<i>telegram.constants.PollType</i> method), 875	<code>__or__()</code> (<i>telegram.constants.InlineKeyboardMarkupLimit</i> method), 768
<code>__new__()</code> (<i>telegram.constants.PollingLimit</i> method), 881	<code>__or__()</code> (<i>telegram.constants.InlineQueryLimit</i> method), 772
<code>__new__()</code> (<i>telegram.constants.ReactionEmoji</i> method), 893	<code>__or__()</code> (<i>telegram.constants.InlineQueryResultLimit</i> method), 776
<code>__new__()</code> (<i>telegram.constants.ReactionType</i> method), 899	<code>__or__()</code> (<i>telegram.constants.InlineQueryResultsButtonLimit</i> method), 788
<code>__new__()</code> (<i>telegram.constants.ReplyLimit</i> method), 905	<code>__or__()</code> (<i>telegram.constants.InvoiceLimit</i> method), 799
<code>__new__()</code> (<i>telegram.constants.StickerFormat</i> method), 909	<code>__or__()</code> (<i>telegram.constants.KeyboardButtonRequestUsersLimit</i> method), 803
<code>__new__()</code> (<i>telegram.constants.StickerLimit</i> method), 916	<code>__or__()</code> (<i>telegram.constants.LocationLimit</i> method), 809
<code>__new__()</code> (<i>telegram.constants.StickerSetLimit</i> method), 921	<code>__or__()</code> (<i>telegram.constants.MediaGroupLimit</i> method), 819
<code>__new__()</code> (<i>telegram.constants.StickerType</i> method), 924	<code>__or__()</code> (<i>telegram.constants.MessageLimit</i> method), 844
<code>__new__()</code> (<i>telegram.constants.UpdateType</i> method), 932	<code>__or__()</code> (<i>telegram.constants.PollLimit</i> method), 872
<code>__new__()</code> (<i>telegram.constants.UserProfilePhotosLimit</i> method), 938	<code>__or__()</code> (<i>telegram.constants.PollingLimit</i> method), 881
<code>__new__()</code> (<i>telegram.constants.WebhookLimit</i> method), 942	<code>__or__()</code> (<i>telegram.constants.ReplyLimit</i> method), 905
<code>__or__()</code> (<i>telegram.constants.BotCommandLimit</i> method), 655	<code>__or__()</code> (<i>telegram.constants.StickerLimit</i> method), 916
<code>__or__()</code> (<i>telegram.constants.BotDescriptionLimit</i> method), 665	<code>__or__()</code> (<i>telegram.constants.StickerSetLimit</i> method), 921
	<code>__or__()</code> (<i>telegram.constants.UserProfilePhotosLimit</i> method), 938

- `__or__()` (*telegram.constants.WebhookLimit* method), 942
- `__or__()` (*telegram.ext.filters.BaseFilter* method), 593
- `__pos__()` (*telegram.constants.BotCommandLimit* method), 655
- `__pos__()` (*telegram.constants.BotDescriptionLimit* method), 665
- `__pos__()` (*telegram.constants.BotNameLimit* method), 669
- `__pos__()` (*telegram.constants.BulkRequestLimit* method), 673
- `__pos__()` (*telegram.constants.CallbackQueryLimit* method), 677
- `__pos__()` (*telegram.constants.ChatID* method), 694
- `__pos__()` (*telegram.constants.ChatInviteLinkLimit* method), 699
- `__pos__()` (*telegram.constants.ChatLimit* method), 703
- `__pos__()` (*telegram.constants.ChatPhotoSize* method), 713
- `__pos__()` (*telegram.constants.ContactLimit* method), 723
- `__pos__()` (*telegram.constants.CustomEmojiStickerLimit* method), 727
- `__pos__()` (*telegram.constants.DiceLimit* method), 738
- `__pos__()` (*telegram.constants.FileSizeLimit* method), 743
- `__pos__()` (*telegram.constants.FloodLimit* method), 747
- `__pos__()` (*telegram.constants.ForumIconColor* method), 751
- `__pos__()` (*telegram.constants.ForumTopicLimit* method), 756
- `__pos__()` (*telegram.constants.GiveawayLimit* method), 760
- `__pos__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 764
- `__pos__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 768
- `__pos__()` (*telegram.constants.InlineQueryLimit* method), 772
- `__pos__()` (*telegram.constants.InlineQueryResultLimit* method), 777
- `__pos__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 788
- `__pos__()` (*telegram.constants.InvoiceLimit* method), 799
- `__pos__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 803
- `__pos__()` (*telegram.constants.LocationLimit* method), 809
- `__pos__()` (*telegram.constants.MediaGroupLimit* method), 819
- `__pos__()` (*telegram.constants.MessageLimit* method), 844
- `__pos__()` (*telegram.constants.PollLimit* method), 872
- `__pos__()` (*telegram.constants.PollingLimit* method), 881
- `__pos__()` (*telegram.constants.ReplyLimit* method), 905
- `__pos__()` (*telegram.constants.StickerLimit* method), 916
- `__pos__()` (*telegram.constants.StickerSetLimit* method), 921
- `__pos__()` (*telegram.constants.UserProfilePhotosLimit* method), 938
- `__pos__()` (*telegram.constants.WebhookLimit* method), 942
- `__pow__()` (*telegram.constants.BotCommandLimit* method), 655
- `__pow__()` (*telegram.constants.BotDescriptionLimit* method), 665
- `__pow__()` (*telegram.constants.BotNameLimit* method), 669
- `__pow__()` (*telegram.constants.BulkRequestLimit* method), 673
- `__pow__()` (*telegram.constants.CallbackQueryLimit* method), 677
- `__pow__()` (*telegram.constants.ChatID* method), 694
- `__pow__()` (*telegram.constants.ChatInviteLinkLimit* method), 699
- `__pow__()` (*telegram.constants.ChatLimit* method), 703
- `__pow__()` (*telegram.constants.ChatPhotoSize* method), 713
- `__pow__()` (*telegram.constants.ContactLimit* method), 723
- `__pow__()` (*telegram.constants.CustomEmojiStickerLimit* method), 727
- `__pow__()` (*telegram.constants.DiceLimit* method), 738
- `__pow__()` (*telegram.constants.FileSizeLimit* method), 743
- `__pow__()` (*telegram.constants.FloodLimit* method), 747
- `__pow__()` (*telegram.constants.ForumIconColor* method), 751
- `__pow__()` (*telegram.constants.ForumTopicLimit* method), 756
- `__pow__()` (*telegram.constants.GiveawayLimit* method), 760
- `__pow__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 764
- `__pow__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 768
- `__pow__()` (*telegram.constants.InlineQueryLimit* method), 772
- `__pow__()` (*telegram.constants.InlineQueryResultLimit* method), 777
- `__pow__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 788
- `__pow__()` (*telegram.constants.InvoiceLimit* method), 799
- `__pow__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 803

<code>__pow__()</code> (<i>telegram.constants.LocationLimit method</i>), 809	<code>__radd__()</code> (<i>telegram.constants.InlineQueryResultLimit method</i>), 777
<code>__pow__()</code> (<i>telegram.constants.MediaGroupLimit method</i>), 819	<code>__radd__()</code> (<i>telegram.constants.InlineQueryResultsButtonLimit method</i>), 788
<code>__pow__()</code> (<i>telegram.constants.MessageLimit method</i>), 844	<code>__radd__()</code> (<i>telegram.constants.InvoiceLimit method</i>), 799
<code>__pow__()</code> (<i>telegram.constants.PollLimit method</i>), 872	<code>__radd__()</code> (<i>telegram.constants.KeyboardButtonRequestUsersLimit method</i>), 803
<code>__pow__()</code> (<i>telegram.constants.PollingLimit method</i>), 881	<code>__radd__()</code> (<i>telegram.constants.LocationLimit method</i>), 809
<code>__pow__()</code> (<i>telegram.constants.ReplyLimit method</i>), 905	<code>__radd__()</code> (<i>telegram.constants.MediaGroupLimit method</i>), 819
<code>__pow__()</code> (<i>telegram.constants.StickerLimit method</i>), 916	<code>__radd__()</code> (<i>telegram.constants.MessageLimit method</i>), 844
<code>__pow__()</code> (<i>telegram.constants.StickerSetLimit method</i>), 921	<code>__radd__()</code> (<i>telegram.constants.PollLimit method</i>), 872
<code>__pow__()</code> (<i>telegram.constants.UserProfilePhotosLimit method</i>), 938	<code>__radd__()</code> (<i>telegram.constants.PollingLimit method</i>), 882
<code>__pow__()</code> (<i>telegram.constants.WebhookLimit method</i>), 942	<code>__radd__()</code> (<i>telegram.constants.ReplyLimit method</i>), 905
<code>__radd__()</code> (<i>telegram.constants.BotCommandLimit method</i>), 655	<code>__radd__()</code> (<i>telegram.constants.StickerLimit method</i>), 916
<code>__radd__()</code> (<i>telegram.constants.BotDescriptionLimit method</i>), 665	<code>__radd__()</code> (<i>telegram.constants.StickerSetLimit method</i>), 921
<code>__radd__()</code> (<i>telegram.constants.BotNameLimit method</i>), 669	<code>__radd__()</code> (<i>telegram.constants.UserProfilePhotosLimit method</i>), 938
<code>__radd__()</code> (<i>telegram.constants.BulkRequestLimit method</i>), 673	<code>__radd__()</code> (<i>telegram.constants.WebhookLimit method</i>), 943
<code>__radd__()</code> (<i>telegram.constants.CallbackQueryLimit method</i>), 677	<code>__rand__()</code> (<i>telegram.constants.BotCommandLimit method</i>), 655
<code>__radd__()</code> (<i>telegram.constants.ChatID method</i>), 694	<code>__rand__()</code> (<i>telegram.constants.BotDescriptionLimit method</i>), 665
<code>__radd__()</code> (<i>telegram.constants.ChatInviteLinkLimit method</i>), 699	<code>__rand__()</code> (<i>telegram.constants.BotNameLimit method</i>), 669
<code>__radd__()</code> (<i>telegram.constants.ChatLimit method</i>), 703	<code>__rand__()</code> (<i>telegram.constants.BulkRequestLimit method</i>), 673
<code>__radd__()</code> (<i>telegram.constants.ChatPhotoSize method</i>), 713	<code>__rand__()</code> (<i>telegram.constants.CallbackQueryLimit method</i>), 677
<code>__radd__()</code> (<i>telegram.constants.ContactLimit method</i>), 723	<code>__rand__()</code> (<i>telegram.constants.ChatID method</i>), 694
<code>__radd__()</code> (<i>telegram.constants.CustomEmojiStickerLimit method</i>), 727	<code>__rand__()</code> (<i>telegram.constants.ChatInviteLinkLimit method</i>), 699
<code>__radd__()</code> (<i>telegram.constants.DiceLimit method</i>), 738	<code>__rand__()</code> (<i>telegram.constants.ChatLimit method</i>), 703
<code>__radd__()</code> (<i>telegram.constants.FileSizeLimit method</i>), 743	<code>__rand__()</code> (<i>telegram.constants.ChatPhotoSize method</i>), 713
<code>__radd__()</code> (<i>telegram.constants.FloodLimit method</i>), 747	<code>__rand__()</code> (<i>telegram.constants.ContactLimit method</i>), 723
<code>__radd__()</code> (<i>telegram.constants.ForumIconColor method</i>), 751	<code>__rand__()</code> (<i>telegram.constants.CustomEmojiStickerLimit method</i>), 727
<code>__radd__()</code> (<i>telegram.constants.ForumTopicLimit method</i>), 756	<code>__rand__()</code> (<i>telegram.constants.DiceLimit method</i>), 738
<code>__radd__()</code> (<i>telegram.constants.GiveawayLimit method</i>), 760	<code>__rand__()</code> (<i>telegram.constants.FileSizeLimit method</i>), 743
<code>__radd__()</code> (<i>telegram.constants.InlineKeyboardButtonLimit method</i>), 764	<code>__rand__()</code> (<i>telegram.constants.FloodLimit method</i>), 747
<code>__radd__()</code> (<i>telegram.constants.InlineKeyboardMarkupLimit method</i>), 768	<code>__rand__()</code> (<i>telegram.constants.ForumIconColor method</i>), 751
<code>__radd__()</code> (<i>telegram.constants.InlineQueryLimit method</i>), 772	<code>__rand__()</code> (<i>telegram.constants.ForumTopicLimit method</i>), 756

method), 756
__rand__() (telegram.constants.GiveawayLimit method), 760
__rand__() (telegram.constants.InlineKeyboardButtonLimit method), 764
__rand__() (telegram.constants.InlineKeyboardMarkupLimit method), 768
__rand__() (telegram.constants.InlineQueryLimit method), 773
__rand__() (telegram.constants.InlineQueryResultLimit method), 777
__rand__() (telegram.constants.InlineQueryResultsButtonLimit method), 788
__rand__() (telegram.constants.InvoiceLimit method), 799
__rand__() (telegram.constants.KeyboardButtonRequestUsersLimit method), 803
__rand__() (telegram.constants.LocationLimit method), 809
__rand__() (telegram.constants.MediaGroupLimit method), 819
__rand__() (telegram.constants.MessageLimit method), 844
__rand__() (telegram.constants.PollLimit method), 872
__rand__() (telegram.constants.PollingLimit method), 882
__rand__() (telegram.constants.ReplyLimit method), 905
__rand__() (telegram.constants.StickerLimit method), 916
__rand__() (telegram.constants.StickerSetLimit method), 921
__rand__() (telegram.constants.UserProfilePhotosLimit method), 938
__rand__() (telegram.constants.WebhookLimit method), 943
__rdivmod__() (telegram.constants.BotCommandLimit method), 655
__rdivmod__() (telegram.constants.BotDescriptionLimit method), 665
__rdivmod__() (telegram.constants.BotNameLimit method), 669
__rdivmod__() (telegram.constants.BulkRequestLimit method), 673
__rdivmod__() (telegram.constants.CallbackQueryLimit method), 677
__rdivmod__() (telegram.constants.ChatID method), 694
__rdivmod__() (telegram.constants.ChatInviteLinkLimit method), 699
__rdivmod__() (telegram.constants.ChatLimit method), 703
__rdivmod__() (telegram.constants.ChatPhotoSize method), 713
__rdivmod__() (telegram.constants.ContactLimit method), 723
__rdivmod__() (telegram.constants.CustomEmojiStickerLimit method), 727
__rdivmod__() (telegram.constants.DiceLimit method), 738
__rdivmod__() (telegram.constants.FileSizeLimit method), 743
__rdivmod__() (telegram.constants.FloodLimit method), 747
__rdivmod__() (telegram.constants.ForumIconColor method), 751
__rdivmod__() (telegram.constants.ForumTopicLimit method), 756
__rdivmod__() (telegram.constants.GiveawayLimit method), 760
__rdivmod__() (telegram.constants.InlineKeyboardButtonLimit method), 764
__rdivmod__() (telegram.constants.InlineKeyboardMarkupLimit method), 768
__rdivmod__() (telegram.constants.InlineQueryLimit method), 773
__rdivmod__() (telegram.constants.InlineQueryResultLimit method), 777
__rdivmod__() (telegram.constants.InlineQueryResultsButtonLimit method), 788
__rdivmod__() (telegram.constants.InvoiceLimit method), 799
__rdivmod__() (telegram.constants.KeyboardButtonRequestUsersLimit method), 803
__rdivmod__() (telegram.constants.LocationLimit method), 809
__rdivmod__() (telegram.constants.MediaGroupLimit method), 819
__rdivmod__() (telegram.constants.MessageLimit method), 844
__rdivmod__() (telegram.constants.PollLimit method), 872
__rdivmod__() (telegram.constants.PollingLimit method), 882
__rdivmod__() (telegram.constants.ReplyLimit method), 905
__rdivmod__() (telegram.constants.StickerLimit method), 916
__rdivmod__() (telegram.constants.StickerSetLimit method), 921
__rdivmod__() (telegram.constants.UserProfilePhotosLimit method), 938

- `__rdivmod__()` (*telegram.constants.WebhookLimit* method), 943
- `__reduce__()` (*telegram.Bot* method), 27
- `__reduce__()` (*telegram.error.ChatMigrated* method), 945
- `__reduce__()` (*telegram.error.Conflict* method), 946
- `__reduce__()` (*telegram.error.PassportDecryptionError* method), 947
- `__reduce__()` (*telegram.error.RetryAfter* method), 947
- `__reduce__()` (*telegram.error.TelegramError* method), 948
- `__reduce__()` (*telegram.ext.InvalidCallbackData* method), 647
- `__repr__()` (*telegram.Bot* method), 27
- `__repr__()` (*telegram.TelegramObject* method), 374
- `__repr__()` (*telegram.constants.BotCommandLimit* method), 655
- `__repr__()` (*telegram.constants.BotCommandScopeType* method), 659
- `__repr__()` (*telegram.constants.BotDescriptionLimit* method), 665
- `__repr__()` (*telegram.constants.BotNameLimit* method), 669
- `__repr__()` (*telegram.constants.BulkRequestLimit* method), 673
- `__repr__()` (*telegram.constants.CallbackQueryLimit* method), 677
- `__repr__()` (*telegram.constants.ChatAction* method), 681
- `__repr__()` (*telegram.constants.ChatBoostSources* method), 687
- `__repr__()` (*telegram.constants.ChatID* method), 694
- `__repr__()` (*telegram.constants.ChatInviteLinkLimit* method), 699
- `__repr__()` (*telegram.constants.ChatLimit* method), 703
- `__repr__()` (*telegram.constants.ChatMemberStatus* method), 707
- `__repr__()` (*telegram.constants.ChatPhotoSize* method), 713
- `__repr__()` (*telegram.constants.ChatType* method), 717
- `__repr__()` (*telegram.constants.ContactLimit* method), 723
- `__repr__()` (*telegram.constants.CustomEmojiStickerLimit* method), 728
- `__repr__()` (*telegram.constants.DiceEmoji* method), 731
- `__repr__()` (*telegram.constants.DiceLimit* method), 738
- `__repr__()` (*telegram.constants.FileSizeLimit* method), 743
- `__repr__()` (*telegram.constants.FloodLimit* method), 747
- `__repr__()` (*telegram.constants.ForumIconColor* method), 752
- `__repr__()` (*telegram.constants.ForumTopicLimit* method), 756
- `__repr__()` (*telegram.constants.GiveawayLimit* method), 760
- `__repr__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 764
- `__repr__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 768
- `__repr__()` (*telegram.constants.InlineQueryLimit* method), 773
- `__repr__()` (*telegram.constants.InlineQueryResultLimit* method), 777
- `__repr__()` (*telegram.constants.InlineQueryResultType* method), 781
- `__repr__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 788
- `__repr__()` (*telegram.constants.InputMediaType* method), 791
- `__repr__()` (*telegram.constants.InvoiceLimit* method), 799
- `__repr__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 803
- `__repr__()` (*telegram.constants.LocationLimit* method), 809
- `__repr__()` (*telegram.constants.MaskPosition* method), 812
- `__repr__()` (*telegram.constants.MediaGroupLimit* method), 819
- `__repr__()` (*telegram.constants.MenuButtonType* method), 822
- `__repr__()` (*telegram.constants.MessageAttachmentType* method), 830
- `__repr__()` (*telegram.constants.MessageEntityType* method), 837
- `__repr__()` (*telegram.constants.MessageLimit* method), 844
- `__repr__()` (*telegram.constants.MessageOriginType* method), 848
- `__repr__()` (*telegram.constants.MessageType* method), 858
- `__repr__()` (*telegram.constants.ParseMode* method), 864
- `__repr__()` (*telegram.constants.PollLimit* method), 872
- `__repr__()` (*telegram.constants.PollType* method), 875
- `__repr__()` (*telegram.constants.PollingLimit* method), 882
- `__repr__()` (*telegram.constants.ReactionEmoji* method), 893
- `__repr__()` (*telegram.constants.ReactionType* method), 899
- `__repr__()` (*telegram.constants.ReplyLimit* method), 905
- `__repr__()` (*telegram.constants.StickerFormat* method), 909
- `__repr__()` (*telegram.constants.StickerLimit* method), 916

`__repr__()` (*telegram.constants.StickerSetLimit* method), 921

`__repr__()` (*telegram.constants.StickerType* method), 924

`__repr__()` (*telegram.constants.UpdateType* method), 932

`__repr__()` (*telegram.constants.UserProfilePhotosLimit* method), 938

`__repr__()` (*telegram.constants.WebhookLimit* method), 943

`__repr__()` (*telegram.error.TelegramError* method), 948

`__repr__()` (*telegram.ext.Application* method), 520

`__repr__()` (*telegram.ext.BaseHandler* method), 575

`__repr__()` (*telegram.ext.ConversationHandler* method), 588

`__repr__()` (*telegram.ext.Job* method), 561

`__repr__()` (*telegram.ext.JobQueue* method), 563

`__repr__()` (*telegram.ext.Updater* method), 570

`__repr__()` (*telegram.ext.filters.BaseFilter* method), 593

`__rfloordiv__()` (*telegram.constants.BotCommandLimit* method), 655

`__rfloordiv__()` (*telegram.constants.BotDescriptionLimit* method), 665

`__rfloordiv__()` (*telegram.constants.BotNameLimit* method), 669

`__rfloordiv__()` (*telegram.constants.BulkRequestLimit* method), 673

`__rfloordiv__()` (*telegram.constants.CallbackQueryLimit* method), 677

`__rfloordiv__()` (*telegram.constants.ChatID* method), 695

`__rfloordiv__()` (*telegram.constants.ChatInviteLinkLimit* method), 699

`__rfloordiv__()` (*telegram.constants.ChatLimit* method), 703

`__rfloordiv__()` (*telegram.constants.ChatPhotoSize* method), 713

`__rfloordiv__()` (*telegram.constants.ContactLimit* method), 723

`__rfloordiv__()` (*telegram.constants.CustomEmojiStickerLimit* method), 728

`__rfloordiv__()` (*telegram.constants.DiceLimit* method), 738

`__rfloordiv__()` (*telegram.constants.FileSizeLimit* method), 743

`__rfloordiv__()` (*telegram.constants.FloodLimit* method), 747

`__rfloordiv__()` (*telegram.constants.ForumIconColor* method), 752

`__rfloordiv__()` (*telegram.constants.ForumTopicLimit* method), 756

`__rfloordiv__()` (*telegram.constants.GiveawayLimit* method), 760

`__rfloordiv__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 764

`__rfloordiv__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 768

`__rfloordiv__()` (*telegram.constants.InlineQueryLimit* method), 773

`__rfloordiv__()` (*telegram.constants.InlineQueryResultLimit* method), 777

`__rfloordiv__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 788

`__rfloordiv__()` (*telegram.constants.InvoiceLimit* method), 799

`__rfloordiv__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 803

`__rfloordiv__()` (*telegram.constants.LocationLimit* method), 809

`__rfloordiv__()` (*telegram.constants.MediaGroupLimit* method), 819

`__rfloordiv__()` (*telegram.constants.MessageLimit* method), 844

`__rfloordiv__()` (*telegram.constants.PollLimit* method), 872

`__rfloordiv__()` (*telegram.constants.PollingLimit* method), 882

`__rfloordiv__()` (*telegram.constants.ReplyLimit* method), 905

`__rfloordiv__()` (*telegram.constants.StickerLimit* method), 916

`__rfloordiv__()` (*telegram.constants.StickerSetLimit* method), 921

`__rfloordiv__()` (*telegram.constants.UserProfilePhotosLimit* method), 938

`__rfloordiv__()` (*telegram.constants.WebhookLimit* method), 943

`__rlshift__()` (*telegram.constants.BotCommandLimit* method), 655

`__rlshift__()` (*telegram.constants.BotDescriptionLimit* method), 665

`__rlshift__()` (*telegram.constants.BotNameLimit* method), 669

`__rlshift__()` (*telegram.constants.BotCommandLimit* method), 655

`gram.constants.BulkRequestLimit` method), 673
`__rlshift__()` (`telegram.constants.CallbackQueryLimit` method), 677
`__rlshift__()` (`telegram.constants.ChatID` method), 695
`__rlshift__()` (`telegram.constants.ChatInviteLinkLimit` method), 699
`__rlshift__()` (`telegram.constants.ChatLimit` method), 703
`__rlshift__()` (`telegram.constants.ChatPhotoSize` method), 713
`__rlshift__()` (`telegram.constants.ContactLimit` method), 724
`__rlshift__()` (`telegram.constants.CustomEmojiStickerLimit` method), 728
`__rlshift__()` (`telegram.constants.DiceLimit` method), 738
`__rlshift__()` (`telegram.constants.FileSizeLimit` method), 743
`__rlshift__()` (`telegram.constants.FloodLimit` method), 747
`__rlshift__()` (`telegram.constants.ForumIconColor` method), 752
`__rlshift__()` (`telegram.constants.ForumTopicLimit` method), 756
`__rlshift__()` (`telegram.constants.GiveawayLimit` method), 760
`__rlshift__()` (`telegram.constants.InlineKeyboardButtonLimit` method), 764
`__rlshift__()` (`telegram.constants.InlineKeyboardMarkupLimit` method), 768
`__rlshift__()` (`telegram.constants.InlineQueryLimit` method), 773
`__rlshift__()` (`telegram.constants.InlineQueryResultLimit` method), 777
`__rlshift__()` (`telegram.constants.InlineQueryResultsButtonLimit` method), 788
`__rlshift__()` (`telegram.constants.InvoiceLimit` method), 799
`__rlshift__()` (`telegram.constants.KeyboardButtonRequestUsersLimit` method), 803
`__rlshift__()` (`telegram.constants.LocationLimit` method), 809
`__rlshift__()` (`telegram.constants.MediaGroupLimit` method), 819
`__rlshift__()` (`telegram.constants.MessageLimit` method), 844
`__rlshift__()` (`telegram.constants.PollLimit` method), 872
`__rlshift__()` (`telegram.constants.PollingLimit` method), 882
`__rlshift__()` (`telegram.constants.ReplyLimit` method), 906
`__rlshift__()` (`telegram.constants.StickerLimit` method), 916
`__rlshift__()` (`telegram.constants.StickerSetLimit` method), 921
`__rlshift__()` (`telegram.constants.UserProfilePhotosLimit` method), 938
`__rlshift__()` (`telegram.constants.WebhookLimit` method), 943
`__rmod__()` (`telegram.constants.BotCommandLimit` method), 655
`__rmod__()` (`telegram.constants.BotCommandScopeType` method), 659
`__rmod__()` (`telegram.constants.BotDescriptionLimit` method), 665
`__rmod__()` (`telegram.constants.BotNameLimit` method), 669
`__rmod__()` (`telegram.constants.BulkRequestLimit` method), 673
`__rmod__()` (`telegram.constants.CallbackQueryLimit` method), 677
`__rmod__()` (`telegram.constants.ChatAction` method), 681
`__rmod__()` (`telegram.constants.ChatBoostSources` method), 687
`__rmod__()` (`telegram.constants.ChatID` method), 695
`__rmod__()` (`telegram.constants.ChatInviteLinkLimit` method), 699
`__rmod__()` (`telegram.constants.ChatLimit` method), 703
`__rmod__()` (`telegram.constants.ChatMemberStatus` method), 707
`__rmod__()` (`telegram.constants.ChatPhotoSize` method), 713
`__rmod__()` (`telegram.constants.ChatType` method), 717
`__rmod__()` (`telegram.constants.ContactLimit` method), 724
`__rmod__()` (`telegram.constants.CustomEmojiStickerLimit` method), 728
`__rmod__()` (`telegram.constants.DiceEmoji` method), 731
`__rmod__()` (`telegram.constants.DiceLimit` method), 738
`__rmod__()` (`telegram.constants.FileSizeLimit` method), 743
`__rmod__()` (`telegram.constants.FloodLimit` method), 747
`__rmod__()` (`telegram.constants.ForumIconColor` method), 752
`__rmod__()` (`telegram.constants.ForumTopicLimit` method), 756
`__rmod__()` (`telegram.constants.GiveawayLimit` method), 760

<code>method</code>), 760	<code>__rmul__</code> () (telegram.constants.UpdateType method), 924
<code>__rmod__</code> () (telegram.constants.InlineKeyboardButtonLimit method), 764	<code>__rmul__</code> () (telegram.constants.UserProfilePhotosLimit method), 932
<code>__rmod__</code> () (telegram.constants.InlineKeyboardMarkupLimit method), 768	<code>__rmul__</code> () (telegram.constants.WebhookLimit method), 943
<code>__rmod__</code> () (telegram.constants.InlineQueryLimit method), 773	<code>__rmul__</code> () (telegram.constants.BotCommandLimit method), 655
<code>__rmod__</code> () (telegram.constants.InlineQueryResultLimit method), 777	<code>__rmul__</code> () (telegram.constants.BotCommandScopeType method), 659
<code>__rmod__</code> () (telegram.constants.InlineQueryResultType method), 781	<code>__rmul__</code> () (telegram.constants.BotDescriptionLimit method), 665
<code>__rmod__</code> () (telegram.constants.InlineQueryResultsButtonLimit method), 788	<code>__rmul__</code> () (telegram.constants.BotNameLimit method), 669
<code>__rmod__</code> () (telegram.constants.InputMediaType method), 791	<code>__rmul__</code> () (telegram.constants.BulkRequestLimit method), 673
<code>__rmod__</code> () (telegram.constants.InvoiceLimit method), 799	<code>__rmul__</code> () (telegram.constants.CallbackQueryLimit method), 677
<code>__rmod__</code> () (telegram.constants.KeyboardButtonRequestUserLimit method), 803	<code>__rmul__</code> () (telegram.constants.ChatAction method), 682
<code>__rmod__</code> () (telegram.constants.LocationLimit method), 809	<code>__rmul__</code> () (telegram.constants.ChatBoostSources method), 687
<code>__rmod__</code> () (telegram.constants.MaskPosition method), 813	<code>__rmul__</code> () (telegram.constants.ChatID method), 695
<code>__rmod__</code> () (telegram.constants.MediaGroupLimit method), 819	<code>__rmul__</code> () (telegram.constants.ChatInviteLinkLimit method), 699
<code>__rmod__</code> () (telegram.constants.MenuButtonType method), 822	<code>__rmul__</code> () (telegram.constants.ChatLimit method), 703
<code>__rmod__</code> () (telegram.constants.MessageAttachmentType method), 830	<code>__rmul__</code> () (telegram.constants.ChatMemberStatus method), 707
<code>__rmod__</code> () (telegram.constants.MessageEntityType method), 837	<code>__rmul__</code> () (telegram.constants.ChatPhotoSize method), 714
<code>__rmod__</code> () (telegram.constants.MessageLimit method), 845	<code>__rmul__</code> () (telegram.constants.ChatType method), 717
<code>__rmod__</code> () (telegram.constants.MessageOriginType method), 848	<code>__rmul__</code> () (telegram.constants.ContactLimit method), 724
<code>__rmod__</code> () (telegram.constants.MessageType method), 858	<code>__rmul__</code> () (telegram.constants.CustomEmojiStickerLimit method), 728
<code>__rmod__</code> () (telegram.constants.ParseMode method), 864	<code>__rmul__</code> () (telegram.constants.DiceEmoji method), 731
<code>__rmod__</code> () (telegram.constants.PollLimit method), 872	<code>__rmul__</code> () (telegram.constants.DiceLimit method), 738
<code>__rmod__</code> () (telegram.constants.PollType method), 875	<code>__rmul__</code> () (telegram.constants.FileSizeLimit method), 743
<code>__rmod__</code> () (telegram.constants.PollingLimit method), 882	<code>__rmul__</code> () (telegram.constants.FloodLimit method), 747
<code>__rmod__</code> () (telegram.constants.ReactionEmoji method), 893	<code>__rmul__</code> () (telegram.constants.ForumIconColor method), 752
<code>__rmod__</code> () (telegram.constants.ReactionType method), 899	<code>__rmul__</code> () (telegram.constants.ForumTopicLimit method), 756
<code>__rmod__</code> () (telegram.constants.ReplyLimit method), 906	<code>__rmul__</code> () (telegram.constants.GiveawayLimit method), 760
<code>__rmod__</code> () (telegram.constants.StickerFormat method), 909	<code>__rmul__</code> () (telegram.constants.InlineKeyboardButtonLimit method), 764
<code>__rmod__</code> () (telegram.constants.StickerLimit method), 916	<code>__rmul__</code> () (telegram.constants.InlineKeyboardMarkupLimit method), 768
<code>__rmod__</code> () (telegram.constants.StickerSetLimit method), 921	<code>__rmul__</code> () (telegram.constants.InlineQueryLimit method), 773
<code>__rmod__</code> () (telegram.constants.StickerType method),	

<code>__rmul__()</code> (<code>telegram.constants.InlineQueryResultLimit</code> method), 777	<code>__ror__()</code> (<code>telegram.constants.BotCommandLimit</code> method), 655
<code>__rmul__()</code> (<code>telegram.constants.InlineQueryResultType</code> method), 781	<code>__ror__()</code> (<code>telegram.constants.BotDescriptionLimit</code> method), 665
<code>__rmul__()</code> (<code>telegram.constants.InlineQueryResultsButtonLimit</code> method), 788	<code>__ror__()</code> (<code>telegram.constants.BotNameLimit</code> method), 669
<code>__rmul__()</code> (<code>telegram.constants.InputMediaType</code> method), 792	<code>__ror__()</code> (<code>telegram.constants.BulkRequestLimit</code> method), 673
<code>__rmul__()</code> (<code>telegram.constants.InvoiceLimit</code> method), 799	<code>__ror__()</code> (<code>telegram.constants.CallbackQueryLimit</code> method), 677
<code>__rmul__()</code> (<code>telegram.constants.KeyboardButtonRequestUsersLimit</code> method), 803	<code>__ror__()</code> (<code>telegram.constants.ChatID</code> method), 695
<code>__rmul__()</code> (<code>telegram.constants.LocationLimit</code> method), 809	<code>__ror__()</code> (<code>telegram.constants.ChatInviteLinkLimit</code> method), 699
<code>__rmul__()</code> (<code>telegram.constants.MaskPosition</code> method), 813	<code>__ror__()</code> (<code>telegram.constants.ChatLimit</code> method), 703
<code>__rmul__()</code> (<code>telegram.constants.MediaGroupLimit</code> method), 819	<code>__ror__()</code> (<code>telegram.constants.ChatPhotoSize</code> method), 714
<code>__rmul__()</code> (<code>telegram.constants.MenuButtonType</code> method), 823	<code>__ror__()</code> (<code>telegram.constants.ContactLimit</code> method), 724
<code>__rmul__()</code> (<code>telegram.constants.MessageAttachmentType</code> method), 830	<code>__ror__()</code> (<code>telegram.constants.CustomEmojiStickerLimit</code> method), 728
<code>__rmul__()</code> (<code>telegram.constants.MessageEntityType</code> method), 837	<code>__ror__()</code> (<code>telegram.constants.DiceLimit</code> method), 739
<code>__rmul__()</code> (<code>telegram.constants.MessageLimit</code> method), 845	<code>__ror__()</code> (<code>telegram.constants.FileSizeLimit</code> method), 743
<code>__rmul__()</code> (<code>telegram.constants.MessageOriginType</code> method), 848	<code>__ror__()</code> (<code>telegram.constants.FloodLimit</code> method), 747
<code>__rmul__()</code> (<code>telegram.constants.MessageType</code> method), 858	<code>__ror__()</code> (<code>telegram.constants.ForumIconColor</code> method), 752
<code>__rmul__()</code> (<code>telegram.constants.ParseMode</code> method), 865	<code>__ror__()</code> (<code>telegram.constants.ForumTopicLimit</code> method), 756
<code>__rmul__()</code> (<code>telegram.constants.PollLimit</code> method), 872	<code>__ror__()</code> (<code>telegram.constants.GiveawayLimit</code> method), 760
<code>__rmul__()</code> (<code>telegram.constants.PollType</code> method), 875	<code>__ror__()</code> (<code>telegram.constants.InlineKeyboardButtonLimit</code> method), 764
<code>__rmul__()</code> (<code>telegram.constants.PollingLimit</code> method), 882	<code>__ror__()</code> (<code>telegram.constants.InlineKeyboardMarkupLimit</code> method), 768
<code>__rmul__()</code> (<code>telegram.constants.ReactionEmoji</code> method), 893	<code>__ror__()</code> (<code>telegram.constants.InlineQueryLimit</code> method), 773
<code>__rmul__()</code> (<code>telegram.constants.ReactionType</code> method), 899	<code>__ror__()</code> (<code>telegram.constants.InlineQueryResultLimit</code> method), 777
<code>__rmul__()</code> (<code>telegram.constants.ReplyLimit</code> method), 906	<code>__ror__()</code> (<code>telegram.constants.InlineQueryResultsButtonLimit</code> method), 788
<code>__rmul__()</code> (<code>telegram.constants.StickerFormat</code> method), 909	<code>__ror__()</code> (<code>telegram.constants.InvoiceLimit</code> method), 799
<code>__rmul__()</code> (<code>telegram.constants.StickerLimit</code> method), 916	<code>__ror__()</code> (<code>telegram.constants.KeyboardButtonRequestUsersLimit</code> method), 803
<code>__rmul__()</code> (<code>telegram.constants.StickerSetLimit</code> method), 921	<code>__ror__()</code> (<code>telegram.constants.LocationLimit</code> method), 809
<code>__rmul__()</code> (<code>telegram.constants.StickerType</code> method), 925	<code>__ror__()</code> (<code>telegram.constants.MediaGroupLimit</code> method), 819
<code>__rmul__()</code> (<code>telegram.constants.UpdateType</code> method), 932	<code>__ror__()</code> (<code>telegram.constants.MessageLimit</code> method), 845
<code>__rmul__()</code> (<code>telegram.constants.UserProfilePhotosLimit</code> method), 939	<code>__ror__()</code> (<code>telegram.constants.PollLimit</code> method), 872
<code>__rmul__()</code> (<code>telegram.constants.WebhookLimit</code> method), 943	<code>__ror__()</code> (<code>telegram.constants.PollingLimit</code> method), 882
	<code>__ror__()</code> (<code>telegram.constants.ReplyLimit</code> method), 906

<code>__ror__()</code> (<i>telegram.constants.StickerLimit</i> method), 916	<code>__round__()</code> (<i>telegram.constants.MediaGroupLimit</i> method), 819
<code>__ror__()</code> (<i>telegram.constants.StickerSetLimit</i> method), 921	<code>__round__()</code> (<i>telegram.constants.MessageLimit</i> method), 845
<code>__ror__()</code> (<i>telegram.constants.UserProfilePhotosLimit</i> method), 939	<code>__round__()</code> (<i>telegram.constants.PollLimit</i> method), 872
<code>__ror__()</code> (<i>telegram.constants.WebhookLimit</i> method), 943	<code>__round__()</code> (<i>telegram.constants.PollingLimit</i> method), 882
<code>__round__()</code> (<i>telegram.constants.BotCommandLimit</i> method), 655	<code>__round__()</code> (<i>telegram.constants.ReplyLimit</i> method), 906
<code>__round__()</code> (<i>telegram.constants.BotDescriptionLimit</i> method), 665	<code>__round__()</code> (<i>telegram.constants.StickerLimit</i> method), 916
<code>__round__()</code> (<i>telegram.constants.BotNameLimit</i> method), 669	<code>__round__()</code> (<i>telegram.constants.StickerSetLimit</i> method), 921
<code>__round__()</code> (<i>telegram.constants.BulkRequestLimit</i> method), 673	<code>__round__()</code> (<i>telegram.constants.UserProfilePhotosLimit</i> method), 939
<code>__round__()</code> (<i>telegram.constants.CallbackQueryLimit</i> method), 678	<code>__round__()</code> (<i>telegram.constants.WebhookLimit</i> method), 943
<code>__round__()</code> (<i>telegram.constants.ChatID</i> method), 695	<code>__rpow__()</code> (<i>telegram.constants.BotCommandLimit</i> method), 655
<code>__round__()</code> (<i>telegram.constants.ChatInviteLinkLimit</i> method), 699	<code>__rpow__()</code> (<i>telegram.constants.BotDescriptionLimit</i> method), 665
<code>__round__()</code> (<i>telegram.constants.ChatLimit</i> method), 703	<code>__rpow__()</code> (<i>telegram.constants.BotNameLimit</i> method), 669
<code>__round__()</code> (<i>telegram.constants.ChatPhotoSize</i> method), 714	<code>__rpow__()</code> (<i>telegram.constants.BulkRequestLimit</i> method), 674
<code>__round__()</code> (<i>telegram.constants.ContactLimit</i> method), 724	<code>__rpow__()</code> (<i>telegram.constants.CallbackQueryLimit</i> method), 678
<code>__round__()</code> (<i>telegram.constants.CustomEmojiStickerLimit</i> method), 728	<code>__rpow__()</code> (<i>telegram.constants.ChatID</i> method), 695
<code>__round__()</code> (<i>telegram.constants.DiceLimit</i> method), 739	<code>__rpow__()</code> (<i>telegram.constants.ChatInviteLinkLimit</i> method), 699
<code>__round__()</code> (<i>telegram.constants.FileSizeLimit</i> method), 743	<code>__rpow__()</code> (<i>telegram.constants.ChatLimit</i> method), 703
<code>__round__()</code> (<i>telegram.constants.FloodLimit</i> method), 747	<code>__rpow__()</code> (<i>telegram.constants.ChatPhotoSize</i> method), 714
<code>__round__()</code> (<i>telegram.constants.ForumIconColor</i> method), 752	<code>__rpow__()</code> (<i>telegram.constants.ContactLimit</i> method), 724
<code>__round__()</code> (<i>telegram.constants.ForumTopicLimit</i> method), 756	<code>__rpow__()</code> (<i>telegram.constants.CustomEmojiStickerLimit</i> method), 728
<code>__round__()</code> (<i>telegram.constants.GiveawayLimit</i> method), 760	<code>__rpow__()</code> (<i>telegram.constants.DiceLimit</i> method), 739
<code>__round__()</code> (<i>telegram.constants.InlineKeyboardButtonLimit</i> method), 764	<code>__rpow__()</code> (<i>telegram.constants.FileSizeLimit</i> method), 743
<code>__round__()</code> (<i>telegram.constants.InlineKeyboardMarkupLimit</i> method), 768	<code>__rpow__()</code> (<i>telegram.constants.FloodLimit</i> method), 747
<code>__round__()</code> (<i>telegram.constants.InlineQueryLimit</i> method), 773	<code>__rpow__()</code> (<i>telegram.constants.ForumIconColor</i> method), 752
<code>__round__()</code> (<i>telegram.constants.InlineQueryResultLimit</i> method), 777	<code>__rpow__()</code> (<i>telegram.constants.ForumTopicLimit</i> method), 756
<code>__round__()</code> (<i>telegram.constants.InlineQueryResultsButtonLimit</i> method), 788	<code>__rpow__()</code> (<i>telegram.constants.GiveawayLimit</i> method), 760
<code>__round__()</code> (<i>telegram.constants.InvoiceLimit</i> method), 799	<code>__rpow__()</code> (<i>telegram.constants.InlineKeyboardButtonLimit</i> method), 764
<code>__round__()</code> (<i>telegram.constants.KeyboardButtonRequestUsersLimit</i> method), 804	<code>__rpow__()</code> (<i>telegram.constants.InlineKeyboardMarkupLimit</i> method), 768
<code>__round__()</code> (<i>telegram.constants.LocationLimit</i> method), 809	<code>__rpow__()</code> (<i>telegram.constants.InlineQueryLimit</i> method), 773
	<code>__rpow__()</code> (<i>telegram.constants.InlineQueryResultLimit</i> method), 777

method), 777

`__rpow__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 788

`__rpow__()` (*telegram.constants.InvoiceLimit* method), 799

`__rpow__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 804

`__rpow__()` (*telegram.constants.LocationLimit* method), 809

`__rpow__()` (*telegram.constants.MediaGroupLimit* method), 819

`__rpow__()` (*telegram.constants.MessageLimit* method), 845

`__rpow__()` (*telegram.constants.PollLimit* method), 872

`__rpow__()` (*telegram.constants.PollingLimit* method), 882

`__rpow__()` (*telegram.constants.ReplyLimit* method), 906

`__rpow__()` (*telegram.constants.StickerLimit* method), 917

`__rpow__()` (*telegram.constants.StickerSetLimit* method), 921

`__rpow__()` (*telegram.constants.UserProfilePhotosLimit* method), 939

`__rpow__()` (*telegram.constants.WebhookLimit* method), 943

`__rrshift__()` (*telegram.constants.BotCommandLimit* method), 655

`__rrshift__()` (*telegram.constants.BotDescriptionLimit* method), 665

`__rrshift__()` (*telegram.constants.BotNameLimit* method), 669

`__rrshift__()` (*telegram.constants.BulkRequestLimit* method), 674

`__rrshift__()` (*telegram.constants.CallbackQueryLimit* method), 678

`__rrshift__()` (*telegram.constants.ChatID* method), 695

`__rrshift__()` (*telegram.constants.ChatInviteLinkLimit* method), 699

`__rrshift__()` (*telegram.constants.ChatLimit* method), 703

`__rrshift__()` (*telegram.constants.ChatPhotoSize* method), 714

`__rrshift__()` (*telegram.constants.ContactLimit* method), 724

`__rrshift__()` (*telegram.constants.CustomEmojiStickerLimit* method), 728

`__rrshift__()` (*telegram.constants.DiceLimit* method), 739

`__rrshift__()` (*telegram.constants.FileSizeLimit* method), 743

`__rrshift__()` (*telegram.constants.FloodLimit* method), 747

`__rrshift__()` (*telegram.constants.ForumIconColor* method), 752

`__rrshift__()` (*telegram.constants.ForumTopicLimit* method), 756

`__rrshift__()` (*telegram.constants.GiveawayLimit* method), 760

`__rrshift__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 764

`__rrshift__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 769

`__rrshift__()` (*telegram.constants.InlineQueryLimit* method), 773

`__rrshift__()` (*telegram.constants.InlineQueryResultLimit* method), 777

`__rrshift__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 788

`__rrshift__()` (*telegram.constants.InvoiceLimit* method), 799

`__rrshift__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 804

`__rrshift__()` (*telegram.constants.LocationLimit* method), 809

`__rrshift__()` (*telegram.constants.MediaGroupLimit* method), 819

`__rrshift__()` (*telegram.constants.MessageLimit* method), 845

`__rrshift__()` (*telegram.constants.PollLimit* method), 872

`__rrshift__()` (*telegram.constants.PollingLimit* method), 882

`__rrshift__()` (*telegram.constants.ReplyLimit* method), 906

`__rrshift__()` (*telegram.constants.StickerLimit* method), 917

`__rrshift__()` (*telegram.constants.StickerSetLimit* method), 921

`__rrshift__()` (*telegram.constants.UserProfilePhotosLimit* method), 939

`__rrshift__()` (*telegram.constants.WebhookLimit* method), 943

`__rshift__()` (*telegram.constants.BotCommandLimit* method), 655

`__rshift__()` (*telegram.constants.BotDescriptionLimit* method), 666

`__rshift__()` (*telegram.constants.BotNameLimit* method), 669

<code>__rshift__()</code> (<i>telegram.constants.BulkRequestLimit</i> method), 674	<code>__rshift__()</code> (<i>telegram.constants.PollingLimit</i> method), 882
<code>__rshift__()</code> (<i>telegram.constants.CallbackQueryLimit</i> method), 678	<code>__rshift__()</code> (<i>telegram.constants.ReplyLimit</i> method), 906
<code>__rshift__()</code> (<i>telegram.constants.ChatID</i> method), 695	<code>__rshift__()</code> (<i>telegram.constants.StickerLimit</i> method), 917
<code>__rshift__()</code> (<i>telegram.constants.ChatInviteLinkLimit</i> method), 699	<code>__rshift__()</code> (<i>telegram.constants.StickerSetLimit</i> method), 921
<code>__rshift__()</code> (<i>telegram.constants.ChatLimit</i> method), 704	<code>__rshift__()</code> (<i>telegram.constants.UserProfilePhotosLimit</i> method), 939
<code>__rshift__()</code> (<i>telegram.constants.ChatPhotoSize</i> method), 714	<code>__rshift__()</code> (<i>telegram.constants.WebhookLimit</i> method), 943
<code>__rshift__()</code> (<i>telegram.constants.ContactLimit</i> method), 724	<code>__rsub__()</code> (<i>telegram.constants.BotCommandLimit</i> method), 655
<code>__rshift__()</code> (<i>telegram.constants.CustomEmojiStickerLimit</i> method), 728	<code>__rsub__()</code> (<i>telegram.constants.BotDescriptionLimit</i> method), 666
<code>__rshift__()</code> (<i>telegram.constants.DiceLimit</i> method), 739	<code>__rsub__()</code> (<i>telegram.constants.BotNameLimit</i> method), 670
<code>__rshift__()</code> (<i>telegram.constants.FileSizeLimit</i> method), 743	<code>__rsub__()</code> (<i>telegram.constants.BulkRequestLimit</i> method), 674
<code>__rshift__()</code> (<i>telegram.constants.FloodLimit</i> method), 747	<code>__rsub__()</code> (<i>telegram.constants.CallbackQueryLimit</i> method), 678
<code>__rshift__()</code> (<i>telegram.constants.ForumIconColor</i> method), 752	<code>__rsub__()</code> (<i>telegram.constants.ChatID</i> method), 695
<code>__rshift__()</code> (<i>telegram.constants.ForumTopicLimit</i> method), 756	<code>__rsub__()</code> (<i>telegram.constants.ChatInviteLinkLimit</i> method), 699
<code>__rshift__()</code> (<i>telegram.constants.GiveawayLimit</i> method), 760	<code>__rsub__()</code> (<i>telegram.constants.ChatLimit</i> method), 704
<code>__rshift__()</code> (<i>telegram.constants.InlineKeyboardButtonLimit</i> method), 764	<code>__rsub__()</code> (<i>telegram.constants.ChatPhotoSize</i> method), 714
<code>__rshift__()</code> (<i>telegram.constants.InlineKeyboardMarkupLimit</i> method), 769	<code>__rsub__()</code> (<i>telegram.constants.ContactLimit</i> method), 724
<code>__rshift__()</code> (<i>telegram.constants.InlineQueryLimit</i> method), 773	<code>__rsub__()</code> (<i>telegram.constants.CustomEmojiStickerLimit</i> method), 728
<code>__rshift__()</code> (<i>telegram.constants.InlineQueryResultLimit</i> method), 777	<code>__rsub__()</code> (<i>telegram.constants.DiceLimit</i> method), 739
<code>__rshift__()</code> (<i>telegram.constants.InlineQueryResultsButtonLimit</i> method), 788	<code>__rsub__()</code> (<i>telegram.constants.FileSizeLimit</i> method), 743
<code>__rshift__()</code> (<i>telegram.constants.InvoiceLimit</i> method), 800	<code>__rsub__()</code> (<i>telegram.constants.FloodLimit</i> method), 747
<code>__rshift__()</code> (<i>telegram.constants.KeyboardButtonRequestUsersLimit</i> method), 804	<code>__rsub__()</code> (<i>telegram.constants.ForumIconColor</i> method), 752
<code>__rshift__()</code> (<i>telegram.constants.LocationLimit</i> method), 809	<code>__rsub__()</code> (<i>telegram.constants.ForumTopicLimit</i> method), 756
<code>__rshift__()</code> (<i>telegram.constants.MediaGroupLimit</i> method), 819	<code>__rsub__()</code> (<i>telegram.constants.GiveawayLimit</i> method), 760
<code>__rshift__()</code> (<i>telegram.constants.MessageLimit</i> method), 845	<code>__rsub__()</code> (<i>telegram.constants.InlineKeyboardButtonLimit</i> method), 764
<code>__rshift__()</code> (<i>telegram.constants.PollLimit</i> method), 872	<code>__rsub__()</code> (<i>telegram.constants.InlineKeyboardMarkupLimit</i> method), 769
	<code>__rsub__()</code> (<i>telegram.constants.InlineQueryLimit</i> method), 773
	<code>__rsub__()</code> (<i>telegram.constants.InlineQueryResultLimit</i> method), 777
	<code>__rsub__()</code> (<i>telegram.constants.InlineQueryResultsButtonLimit</i> method), 788
	<code>__rsub__()</code> (<i>telegram.constants.InvoiceLimit</i> method), 800

`__rsub__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 804
`__rsub__()` (*telegram.constants.LocationLimit* method), 809
`__rsub__()` (*telegram.constants.MediaGroupLimit* method), 819
`__rsub__()` (*telegram.constants.MessageLimit* method), 845
`__rsub__()` (*telegram.constants.PollLimit* method), 872
`__rsub__()` (*telegram.constants.PollingLimit* method), 882
`__rsub__()` (*telegram.constants.ReplyLimit* method), 906
`__rsub__()` (*telegram.constants.StickerLimit* method), 917
`__rsub__()` (*telegram.constants.StickerSetLimit* method), 921
`__rsub__()` (*telegram.constants.UserProfilePhotosLimit* method), 939
`__rsub__()` (*telegram.constants.WebhookLimit* method), 943
`__rtruediv__()` (*telegram.constants.BotCommandLimit* method), 655
`__rtruediv__()` (*telegram.constants.BotDescriptionLimit* method), 666
`__rtruediv__()` (*telegram.constants.BotNameLimit* method), 670
`__rtruediv__()` (*telegram.constants.BulkRequestLimit* method), 674
`__rtruediv__()` (*telegram.constants.CallbackQueryLimit* method), 678
`__rtruediv__()` (*telegram.constants.ChatID* method), 695
`__rtruediv__()` (*telegram.constants.ChatInviteLinkLimit* method), 699
`__rtruediv__()` (*telegram.constants.ChatLimit* method), 704
`__rtruediv__()` (*telegram.constants.ChatPhotoSize* method), 714
`__rtruediv__()` (*telegram.constants.ContactLimit* method), 724
`__rtruediv__()` (*telegram.constants.CustomEmojiStickerLimit* method), 728
`__rtruediv__()` (*telegram.constants.DiceLimit* method), 739
`__rtruediv__()` (*telegram.constants.FileSizeLimit* method), 743
`__rtruediv__()` (*telegram.constants.FloodLimit* method), 747
`__rtruediv__()` (*telegram.constants.ForumIconColor* method),
`__rtruediv__()` (*telegram.constants.ForumTopicLimit* method), 756
`__rtruediv__()` (*telegram.constants.GiveawayLimit* method), 760
`__rtruediv__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 764
`__rtruediv__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 769
`__rtruediv__()` (*telegram.constants.InlineQueryLimit* method), 773
`__rtruediv__()` (*telegram.constants.InlineQueryResultLimit* method), 777
`__rtruediv__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 788
`__rtruediv__()` (*telegram.constants.InvoiceLimit* method), 800
`__rtruediv__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 804
`__rtruediv__()` (*telegram.constants.LocationLimit* method), 809
`__rtruediv__()` (*telegram.constants.MediaGroupLimit* method), 820
`__rtruediv__()` (*telegram.constants.MessageLimit* method), 845
`__rtruediv__()` (*telegram.constants.PollLimit* method), 872
`__rtruediv__()` (*telegram.constants.PollingLimit* method), 882
`__rtruediv__()` (*telegram.constants.ReplyLimit* method), 906
`__rtruediv__()` (*telegram.constants.StickerLimit* method), 917
`__rtruediv__()` (*telegram.constants.StickerSetLimit* method), 922
`__rtruediv__()` (*telegram.constants.UserProfilePhotosLimit* method), 939
`__rtruediv__()` (*telegram.constants.WebhookLimit* method), 943
`__rxor__()` (*telegram.constants.BotCommandLimit* method), 655
`__rxor__()` (*telegram.constants.BotDescriptionLimit* method), 666
`__rxor__()` (*telegram.constants.BotNameLimit* method), 670
`__rxor__()` (*telegram.constants.BulkRequestLimit* method), 674
`__rxor__()` (*telegram.constants.CallbackQueryLimit* method), 678

`__rxor__()` (*telegram.constants.ChatID* method), 695
`__rxor__()` (*telegram.constants.ChatInviteLinkLimit* method), 699
`__rxor__()` (*telegram.constants.ChatLimit* method), 704
`__rxor__()` (*telegram.constants.ChatPhotoSize* method), 714
`__rxor__()` (*telegram.constants.ContactLimit* method), 724
`__rxor__()` (*telegram.constants.CustomEmojiStickerLimit* method), 728
`__rxor__()` (*telegram.constants.DiceLimit* method), 739
`__rxor__()` (*telegram.constants.FileSizeLimit* method), 743
`__rxor__()` (*telegram.constants.FloodLimit* method), 748
`__rxor__()` (*telegram.constants.ForumIconColor* method), 752
`__rxor__()` (*telegram.constants.ForumTopicLimit* method), 756
`__rxor__()` (*telegram.constants.GiveawayLimit* method), 760
`__rxor__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 764
`__rxor__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 769
`__rxor__()` (*telegram.constants.InlineQueryLimit* method), 773
`__rxor__()` (*telegram.constants.InlineQueryResultLimit* method), 777
`__rxor__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 788
`__rxor__()` (*telegram.constants.InvoiceLimit* method), 800
`__rxor__()` (*telegram.constants.KeyboardButtonRequestUser* method), 804
`__rxor__()` (*telegram.constants.LocationLimit* method), 810
`__rxor__()` (*telegram.constants.MediaGroupLimit* method), 820
`__rxor__()` (*telegram.constants.MessageLimit* method), 845
`__rxor__()` (*telegram.constants.PollLimit* method), 872
`__rxor__()` (*telegram.constants.PollingLimit* method), 882
`__rxor__()` (*telegram.constants.ReplyLimit* method), 906
`__rxor__()` (*telegram.constants.StickerLimit* method), 917
`__rxor__()` (*telegram.constants.StickerSetLimit* method), 922
`__rxor__()` (*telegram.constants.UserProfilePhotosLimit* method), 939
`__rxor__()` (*telegram.constants.WebhookLimit* method), 943
`__setattr__()` (*telegram.TelegramObject* method), 374
`__setstate__()` (*telegram.TelegramObject* method), 374
`__sizeof__()` (*telegram.constants.BotCommandLimit* method), 655
`__sizeof__()` (*telegram.constants.BotCommandScopeType* method), 659
`__sizeof__()` (*telegram.constants.BotDescriptionLimit* method), 666
`__sizeof__()` (*telegram.constants.BotNameLimit* method), 670
`__sizeof__()` (*telegram.constants.BulkRequestLimit* method), 674
`__sizeof__()` (*telegram.constants.CallbackQueryLimit* method), 678
`__sizeof__()` (*telegram.constants.ChatAction* method), 682
`__sizeof__()` (*telegram.constants.ChatBoostSources* method), 688
`__sizeof__()` (*telegram.constants.ChatID* method), 695
`__sizeof__()` (*telegram.constants.ChatInviteLinkLimit* method), 699
`__sizeof__()` (*telegram.constants.ChatLimit* method), 704
`__sizeof__()` (*telegram.constants.ChatMemberStatus* method), 707
`__sizeof__()` (*telegram.constants.ChatPhotoSize* method), 714
`__sizeof__()` (*telegram.constants.ChatType* method), 717
`__sizeof__()` (*telegram.constants.ContactLimit* method), 724
`__sizeof__()` (*telegram.constants.CustomEmojiStickerLimit* method), 728
`__sizeof__()` (*telegram.constants.DiceEmoji* method), 731
`__sizeof__()` (*telegram.constants.DiceLimit* method), 739
`__sizeof__()` (*telegram.constants.FileSizeLimit* method), 743
`__sizeof__()` (*telegram.constants.FloodLimit* method), 748
`__sizeof__()` (*telegram.constants.ForumIconColor* method), 752
`__sizeof__()` (*telegram.constants.ForumTopicLimit* method), 756
`__sizeof__()` (*telegram.constants.GiveawayLimit* method), 760
`__sizeof__()` (*telegram.constants.InlineKeyboardButtonLimit*

method), 764

`__sizeof__()` (telegram.constants.InlineKeyboardMarkupLimit method), 769

`__sizeof__()` (telegram.constants.InlineQueryLimit method), 773

`__sizeof__()` (telegram.constants.InlineQueryResultLimit method), 777

`__sizeof__()` (telegram.constants.InlineQueryResultType method), 781

`__sizeof__()` (telegram.constants.InlineQueryResultsButtonLimit method), 788

`__sizeof__()` (telegram.constants.InputMediaType method), 792

`__sizeof__()` (telegram.constants.InvoiceLimit method), 800

`__sizeof__()` (telegram.constants.KeyboardButtonRequestUsersLimit method), 804

`__sizeof__()` (telegram.constants.LocationLimit method), 810

`__sizeof__()` (telegram.constants.MaskPosition method), 813

`__sizeof__()` (telegram.constants.MediaGroupLimit method), 820

`__sizeof__()` (telegram.constants.MenuButtonType method), 823

`__sizeof__()` (telegram.constants.MessageAttachmentType method), 830

`__sizeof__()` (telegram.constants.MessageEntityType method), 837

`__sizeof__()` (telegram.constants.MessageLimit method), 845

`__sizeof__()` (telegram.constants.MessageOriginType method), 848

`__sizeof__()` (telegram.constants.MessageType method), 859

`__sizeof__()` (telegram.constants.ParseMode method), 865

`__sizeof__()` (telegram.constants.PollLimit method), 872

`__sizeof__()` (telegram.constants.PollType method), 875

`__sizeof__()` (telegram.constants.PollingLimit method), 882

`__sizeof__()` (telegram.constants.ReactionEmoji method), 893

`__sizeof__()` (telegram.constants.ReactionType method), 899

`__sizeof__()` (telegram.constants.ReplyLimit method), 906

`__sizeof__()` (telegram.constants.StickerFormat method), 909

`__sizeof__()` (telegram.constants.StickerLimit method), 917

`__sizeof__()` (telegram.constants.StickerSetLimit method), 922

`__sizeof__()` (telegram.constants.StickerType method), 925

`__sizeof__()` (telegram.constants.UpdateType method), 932

`__sizeof__()` (telegram.constants.UserProfilePhotosLimit method), 939

`__sizeof__()` (telegram.constants.WebhookLimit method), 943

`__str__()` (telegram.constants.BotCommandLimit method), 656

`__str__()` (telegram.constants.BotCommandScopeType method), 659

`__str__()` (telegram.constants.BotDescriptionLimit method), 666

`__str__()` (telegram.constants.BotNameLimit method), 670

`__str__()` (telegram.constants.BulkRequestLimit method), 674

`__str__()` (telegram.constants.CallbackQueryLimit method), 678

`__str__()` (telegram.constants.ChatAction method), 682

`__str__()` (telegram.constants.ChatBoostSources method), 688

`__str__()` (telegram.constants.ChatID method), 695

`__str__()` (telegram.constants.ChatInviteLinkLimit method), 699

`__str__()` (telegram.constants.ChatLimit method), 704

`__str__()` (telegram.constants.ChatMemberStatus method), 707

`__str__()` (telegram.constants.ChatPhotoSize method), 714

`__str__()` (telegram.constants.ChatType method), 717

`__str__()` (telegram.constants.ContactLimit method), 724

`__str__()` (telegram.constants.CustomEmojiStickerLimit method), 728

`__str__()` (telegram.constants.DiceEmoji method), 731

`__str__()` (telegram.constants.DiceLimit method), 739

`__str__()` (telegram.constants.FileSizeLimit method), 743

`__str__()` (telegram.constants.FloodLimit method), 748

`__str__()` (telegram.constants.ForumIconColor method), 752

`__str__()` (telegram.constants.ForumTopicLimit method), 757

`__str__()` (telegram.constants.GiveawayLimit method), 764

method), 760

`__str__()` (`telegram.constants.InlineKeyboardButtonLimit` method), 765

`__str__()` (`telegram.constants.InlineKeyboardMarkupLimit` method), 769

`__str__()` (`telegram.constants.InlineQueryLimit` method), 773

`__str__()` (`telegram.constants.InlineQueryResultLimit` method), 777

`__str__()` (`telegram.constants.InlineQueryResultType` method), 781

`__str__()` (`telegram.constants.InlineQueryResultsButtonLimit` method), 789

`__str__()` (`telegram.constants.InputMediaType` method), 792

`__str__()` (`telegram.constants.InvoiceLimit` method), 800

`__str__()` (`telegram.constants.KeyboardButtonRequestUsersLimit` method), 804

`__str__()` (`telegram.constants.LocationLimit` method), 810

`__str__()` (`telegram.constants.MaskPosition` method), 813

`__str__()` (`telegram.constants.MediaGroupLimit` method), 820

`__str__()` (`telegram.constants.MenuButtonType` method), 823

`__str__()` (`telegram.constants.MessageAttachmentType` method), 830

`__str__()` (`telegram.constants.MessageEntityType` method), 837

`__str__()` (`telegram.constants.MessageLimit` method), 845

`__str__()` (`telegram.constants.MessageOriginType` method), 848

`__str__()` (`telegram.constants.MessageType` method), 859

`__str__()` (`telegram.constants.ParseMode` method), 865

`__str__()` (`telegram.constants.PollLimit` method), 873

`__str__()` (`telegram.constants.PollType` method), 875

`__str__()` (`telegram.constants.PollingLimit` method), 882

`__str__()` (`telegram.constants.ReactionEmoji` method), 893

`__str__()` (`telegram.constants.ReactionType` method), 899

`__str__()` (`telegram.constants.ReplyLimit` method), 906

`__str__()` (`telegram.constants.StickerFormat` method), 909

`__str__()` (`telegram.constants.StickerLimit` method), 917

`__str__()` (`telegram.constants.StickerSetLimit` method), 922

`__str__()` (`telegram.constants.StickerType` method), 925

`__str__()` (`telegram.constants.UpdateType` method), 932

`__str__()` (`telegram.constants.UserProfilePhotosLimit` method), 939

`__str__()` (`telegram.constants.WebhookLimit` method), 943

`__str__()` (`telegram.error.TelegramError` method), 948

`__sub__()` (`telegram.constants.BotCommandLimit` method), 656

`__sub__()` (`telegram.constants.BotDescriptionLimit` method), 666

`__sub__()` (`telegram.constants.BotNameLimit` method), 670

`__sub__()` (`telegram.constants.BulkRequestLimit` method), 674

`__sub__()` (`telegram.constants.CallbackQueryLimit` method), 678

`__sub__()` (`telegram.constants.ChatID` method), 695

`__sub__()` (`telegram.constants.ChatInviteLinkLimit` method), 700

`__sub__()` (`telegram.constants.ChatLimit` method), 704

`__sub__()` (`telegram.constants.ChatPhotoSize` method), 714

`__sub__()` (`telegram.constants.ContactLimit` method), 724

`__sub__()` (`telegram.constants.CustomEmojiStickerLimit` method), 728

`__sub__()` (`telegram.constants.DiceLimit` method), 739

`__sub__()` (`telegram.constants.FileSizeLimit` method), 744

`__sub__()` (`telegram.constants.FloodLimit` method), 748

`__sub__()` (`telegram.constants.ForumIconColor` method), 752

`__sub__()` (`telegram.constants.ForumTopicLimit` method), 757

`__sub__()` (`telegram.constants.GiveawayLimit` method), 761

`__sub__()` (`telegram.constants.InlineKeyboardButtonLimit` method), 765

`__sub__()` (`telegram.constants.InlineKeyboardMarkupLimit` method), 769

`__sub__()` (`telegram.constants.InlineQueryLimit` method), 773

`__sub__()` (`telegram.constants.InlineQueryResultLimit` method), 778

`__sub__()` (`telegram.constants.InlineQueryResultsButtonLimit` method), 789

`__sub__()` (`telegram.constants.InvoiceLimit` method), 800

`__sub__()` (`telegram.constants.KeyboardButtonRequestUsersLimit` method), 804

`__sub__()` (`telegram.constants.LocationLimit` method), 810

`__sub__()` (`telegram.constants.MediaGroupLimit` method), 820

<code>__sub__()</code> (<i>telegram.constants.MessageLimit</i> method), 845	<code>__truediv__()</code> (<i>telegram.constants.InlineKeyboardMarkupLimit</i> method), 769
<code>__sub__()</code> (<i>telegram.constants.PollLimit</i> method), 873	<code>__truediv__()</code> (<i>telegram.constants.InlineQueryLimit</i> method), 773
<code>__sub__()</code> (<i>telegram.constants.PollingLimit</i> method), 882	<code>__truediv__()</code> (<i>telegram.constants.InlineQueryResultLimit</i> method), 778
<code>__sub__()</code> (<i>telegram.constants.ReplyLimit</i> method), 906	<code>__truediv__()</code> (<i>telegram.constants.InlineQueryResultsButtonLimit</i> method), 789
<code>__sub__()</code> (<i>telegram.constants.StickerLimit</i> method), 917	<code>__truediv__()</code> (<i>telegram.constants.InvoiceLimit</i> method), 800
<code>__sub__()</code> (<i>telegram.constants.StickerSetLimit</i> method), 922	<code>__truediv__()</code> (<i>telegram.constants.KeyboardButtonRequestUsersLimit</i> method), 804
<code>__sub__()</code> (<i>telegram.constants.UserProfilePhotosLimit</i> method), 939	<code>__truediv__()</code> (<i>telegram.constants.LocationLimit</i> method), 810
<code>__sub__()</code> (<i>telegram.constants.WebhookLimit</i> method), 943	<code>__truediv__()</code> (<i>telegram.constants.MediaGroupLimit</i> method), 820
<code>__truediv__()</code> (<i>telegram.constants.BotCommandLimit</i> method), 656	<code>__truediv__()</code> (<i>telegram.constants.MessageLimit</i> method), 845
<code>__truediv__()</code> (<i>telegram.constants.BotDescriptionLimit</i> method), 666	<code>__truediv__()</code> (<i>telegram.constants.PollLimit</i> method), 873
<code>__truediv__()</code> (<i>telegram.constants.BotNameLimit</i> method), 670	<code>__truediv__()</code> (<i>telegram.constants.PollingLimit</i> method), 882
<code>__truediv__()</code> (<i>telegram.constants.BulkRequestLimit</i> method), 674	<code>__truediv__()</code> (<i>telegram.constants.ReplyLimit</i> method), 906
<code>__truediv__()</code> (<i>telegram.constants.CallbackQueryLimit</i> method), 678	<code>__truediv__()</code> (<i>telegram.constants.StickerLimit</i> method), 917
<code>__truediv__()</code> (<i>telegram.constants.ChatID</i> method), 695	<code>__truediv__()</code> (<i>telegram.constants.StickerSetLimit</i> method), 922
<code>__truediv__()</code> (<i>telegram.constants.ChatInviteLinkLimit</i> method), 700	<code>__truediv__()</code> (<i>telegram.constants.UserProfilePhotosLimit</i> method), 939
<code>__truediv__()</code> (<i>telegram.constants.ChatLimit</i> method), 704	<code>__truediv__()</code> (<i>telegram.constants.WebhookLimit</i> method), 943
<code>__truediv__()</code> (<i>telegram.constants.ChatPhotoSize</i> method), 714	<code>__trunc__()</code> (<i>telegram.constants.BotCommandLimit</i> method), 656
<code>__truediv__()</code> (<i>telegram.constants.ContactLimit</i> method), 724	<code>__trunc__()</code> (<i>telegram.constants.BotDescriptionLimit</i> method), 666
<code>__truediv__()</code> (<i>telegram.constants.CustomEmojiStickerLimit</i> method), 728	<code>__trunc__()</code> (<i>telegram.constants.BotNameLimit</i> method), 670
<code>__truediv__()</code> (<i>telegram.constants.DiceLimit</i> method), 739	<code>__trunc__()</code> (<i>telegram.constants.BulkRequestLimit</i> method), 674
<code>__truediv__()</code> (<i>telegram.constants.FileSizeLimit</i> method), 744	<code>__trunc__()</code> (<i>telegram.constants.CallbackQueryLimit</i> method), 678
<code>__truediv__()</code> (<i>telegram.constants.FloodLimit</i> method), 748	<code>__trunc__()</code> (<i>telegram.constants.ChatID</i> method), 695
<code>__truediv__()</code> (<i>telegram.constants.ForumIconColor</i> method), 752	<code>__trunc__()</code> (<i>telegram.constants.ChatInviteLinkLimit</i> method), 700
<code>__truediv__()</code> (<i>telegram.constants.ForumTopicLimit</i> method), 757	<code>__trunc__()</code> (<i>telegram.constants.ChatLimit</i> method), 704
<code>__truediv__()</code> (<i>telegram.constants.GiveawayLimit</i> method), 761	<code>__trunc__()</code> (<i>telegram.constants.ChatPhotoSize</i> method), 714
<code>__truediv__()</code> (<i>telegram.constants.InlineKeyboardButtonLimit</i> method), 765	<code>__trunc__()</code> (<i>telegram.constants.ContactLimit</i> method), 724

`__trunc__()` (*telegram.constants.CustomEmojiStickerLimit* method), 728

`__trunc__()` (*telegram.constants.DiceLimit* method), 739

`__trunc__()` (*telegram.constants.FileSizeLimit* method), 744

`__trunc__()` (*telegram.constants.FloodLimit* method), 748

`__trunc__()` (*telegram.constants.ForumIconColor* method), 752

`__trunc__()` (*telegram.constants.ForumTopicLimit* method), 757

`__trunc__()` (*telegram.constants.GiveawayLimit* method), 761

`__trunc__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 765

`__trunc__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 769

`__trunc__()` (*telegram.constants.InlineQueryLimit* method), 773

`__trunc__()` (*telegram.constants.InlineQueryResultLimit* method), 778

`__trunc__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 789

`__trunc__()` (*telegram.constants.InvoiceLimit* method), 800

`__trunc__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 804

`__trunc__()` (*telegram.constants.LocationLimit* method), 810

`__trunc__()` (*telegram.constants.MediaGroupLimit* method), 820

`__trunc__()` (*telegram.constants.MessageLimit* method), 845

`__trunc__()` (*telegram.constants.PollLimit* method), 873

`__trunc__()` (*telegram.constants.PollingLimit* method), 883

`__trunc__()` (*telegram.constants.ReplyLimit* method), 906

`__trunc__()` (*telegram.constants.StickerLimit* method), 917

`__trunc__()` (*telegram.constants.StickerSetLimit* method), 922

`__trunc__()` (*telegram.constants.UserProfilePhotosLimit* method), 939

`__trunc__()` (*telegram.constants.WebhookLimit* method), 944

`__version__` (in module *telegram*), 21

`__version_info__` (in module *telegram*), 21

`__xor__()` (*telegram.constants.BotCommandLimit* method), 656

`__xor__()` (*telegram.constants.BotDescriptionLimit* method), 666

`__xor__()` (*telegram.constants.BotNameLimit* method), 670

`__xor__()` (*telegram.constants.BulkRequestLimit* method), 674

`__xor__()` (*telegram.constants.CallbackQueryLimit* method), 678

`__xor__()` (*telegram.constants.ChatID* method), 695

`__xor__()` (*telegram.constants.ChatInviteLinkLimit* method), 700

`__xor__()` (*telegram.constants.ChatLimit* method), 704

`__xor__()` (*telegram.constants.ChatPhotoSize* method), 714

`__xor__()` (*telegram.constants.ContactLimit* method), 724

`__xor__()` (*telegram.constants.CustomEmojiStickerLimit* method), 728

`__xor__()` (*telegram.constants.DiceLimit* method), 739

`__xor__()` (*telegram.constants.FileSizeLimit* method), 744

`__xor__()` (*telegram.constants.FloodLimit* method), 748

`__xor__()` (*telegram.constants.ForumIconColor* method), 752

`__xor__()` (*telegram.constants.ForumTopicLimit* method), 757

`__xor__()` (*telegram.constants.GiveawayLimit* method), 761

`__xor__()` (*telegram.constants.InlineKeyboardButtonLimit* method), 765

`__xor__()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 769

`__xor__()` (*telegram.constants.InlineQueryLimit* method), 774

`__xor__()` (*telegram.constants.InlineQueryResultLimit* method), 778

`__xor__()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 789

`__xor__()` (*telegram.constants.InvoiceLimit* method), 800

`__xor__()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 804

`__xor__()` (*telegram.constants.LocationLimit* method), 810

`__xor__()` (*telegram.constants.MediaGroupLimit* method), 820

`__xor__()` (*telegram.constants.MessageLimit* method), 845

`__xor__()` (*telegram.constants.PollLimit* method), 873

`__xor__()` (*telegram.constants.PollingLimit* method), 883

`__xor__()` (*telegram.constants.ReplyLimit* method), 906

`__xor__()` (*telegram.constants.StickerLimit* method), 917

`__xor__()` (*telegram.constants.StickerSetLimit* method), 922

`__xor__()` (*telegram.constants.UserProfilePhotosLimit* method), 939

`__xor__()` (*telegram.constants.WebhookLimit* method), 944

`__xor__()` (*telegram.ext.filters.BaseFilter* method), 593

A

`accent_color_id` (*telegram.Chat* attribute), 179

`AccentColor` (class in *telegram.constants*), 651

`active_usernames` (*telegram.Chat* attribute), 179

`actor_chat` (*telegram.MessageReactionUpdated* attribute), 350

`add_bot_ids()` (*telegram.ext.filters.ViaBot* method), 614

`add_chat_ids()` (*telegram.ext.filters.Chat* method), 595

`add_chat_ids()` (*telegram.ext.filters.ForwardedFrom* method), 602

`add_chat_ids()` (*telegram.ext.filters.SenderChat* method), 606

`add_date` (*telegram.ChatBoost* attribute), 206

`add_error_handler()` (*telegram.ext.Application* method), 520

`add_handler()` (*telegram.ext.Application* method), 520

`add_handlers()` (*telegram.ext.Application* method), 521

`add_sticker_to_set()` (*telegram.Bot* method), 28

`add_user_ids()` (*telegram.ext.filters.User* method), 613

`add_usernames()` (*telegram.ext.filters.Chat* method), 596

`add_usernames()` (*telegram.ext.filters.ForwardedFrom* method), 602

`add_usernames()` (*telegram.ext.filters.SenderChat* method), 606

`add_usernames()` (*telegram.ext.filters.User* method), 612

`add_usernames()` (*telegram.ext.filters.ViaBot* method), 614

`added_to_attachment_menu` (*telegram.User* attribute), 386

`additional_chat_count` (*telegram.GiveawayWinners* attribute), 255

`address` (*telegram.ChatLocation* attribute), 216

`address` (*telegram.InlineQueryResultVenue* attribute), 463

`address` (*telegram.InputVenueMessageContent* attribute), 475

`address` (*telegram.SecureData* attribute), 514

`address` (*telegram.Venue* attribute), 404

`addStickerToSet()` (*telegram.Bot* method), 28

`ADMINISTRATOR` (*telegram.ChatMember* attribute), 218

`ADMINISTRATOR` (*telegram.constants.ChatMemberStatus* attribute), 705

`AIORateLimiter` (class in *telegram.ext*), 649

`ALIEN_MONSTER` (*telegram.constants.ReactionEmoji* attribute), 886

`ALL` (in module *telegram.ext.filters*), 590

`ALL` (*telegram.ext.filters.Dice* attribute), 597

`ALL` (*telegram.ext.filters.Document* attribute), 599

`ALL` (*telegram.ext.filters.SenderChat* attribute), 606

`ALL` (*telegram.ext.filters.StatusUpdate* attribute), 607

`ALL` (*telegram.ext.filters.Sticker* attribute), 609

`ALL_CHAT_ADMINISTRATORS` (*telegram.BotCommandScope* attribute), 160

`ALL_CHAT_ADMINISTRATORS` (*telegram.constants.BotCommandScopeType* attribute), 657

`ALL_EMOJI` (*telegram.Dice* attribute), 237

`ALL_GROUP_CHATS` (*telegram.BotCommandScope* attribute), 160

`ALL_GROUP_CHATS` (*telegram.constants.BotCommandScopeType* attribute), 657

`all_permissions()` (*telegram.ChatPermissions* class method), 233

`ALL_PRIVATE_CHATS` (*telegram.BotCommandScope* attribute), 160

`ALL_PRIVATE_CHATS` (*telegram.constants.BotCommandScopeType* attribute), 657

`all_rights()` (*telegram.ChatAdministratorRights* class method), 206

`ALL_TYPES` (*telegram.MessageEntity* attribute), 342

`ALL_TYPES` (*telegram.Update* attribute), 381

`allow_bot_chats` (*telegram.SwitchInlineQueryChosenChat* attribute), 372

`allow_channel_chats` (*telegram.SwitchInlineQueryChosenChat* attribute), 372

`allow_empty` (*telegram.ext.filters.Chat* attribute), 595

`allow_empty` (*telegram.ext.filters.ForwardedFrom* attribute), 602

`allow_empty` (*telegram.ext.filters.SenderChat* attribute), 606

`allow_empty` (*telegram.ext.filters.User* attribute), 612

`allow_empty` (*telegram.ext.filters.ViaBot* attribute), 613

`allow_group_chats` (*telegram.SwitchInlineQueryChosenChat* attribute), 372

`allow_reentry` (*telegram.ext.ConversationHandler* property), 588

`allow_sending_without_reply` (*telegram.ext.Defaults* property), 555

`allow_sending_without_reply` (*telegram.ReplyParameters* attribute), 369

`allow_user_chats` (*telegram.SwitchInlineQueryChosenChat* attribute), 371

`allowed_updates` (*telegram.WebhookInfo* attribute), 415

`allows_multiple_answers` (*telegram.Poll* attribute), 353

- amount (*telegram.LabeledPrice* attribute), 483
- ANIMATED (*telegram.constants.StickerFormat* attribute), 908
- ANIMATED (*telegram.ext.filters.Sticker* attribute), 609
- Animation (class in *telegram*), 155
- ANIMATION (in module *telegram.ext.filters*), 590
- ANIMATION (*telegram.constants.InputMediaType* attribute), 790
- ANIMATION (*telegram.constants.MessageAttachmentType* attribute), 827
- ANIMATION (*telegram.constants.MessageType* attribute), 853
- animation (*telegram.ExternalReplyInfo* attribute), 241
- animation (*telegram.Game* attribute), 492
- animation (*telegram.Message* attribute), 302
- ANONYMOUS_ADMIN (*telegram.constants.ChatID* attribute), 692
- answer() (*telegram.CallbackQuery* method), 167
- answer() (*telegram.InlineQuery* method), 426
- answer() (*telegram.PreCheckoutQuery* method), 486
- answer() (*telegram.ShippingQuery* method), 489
- answer_callback_query() (*telegram.Bot* method), 29
- ANSWER_CALLBACK_QUERY_TEXT_LENGTH (*telegram.constants.CallbackQueryLimit* attribute), 676
- answer_inline_query() (*telegram.Bot* method), 30
- answer_pre_checkout_query() (*telegram.Bot* method), 31
- answer_shipping_query() (*telegram.Bot* method), 32
- answer_web_app_query() (*telegram.Bot* method), 33
- answerCallbackQuery() (*telegram.Bot* method), 28
- answerInlineQuery() (*telegram.Bot* method), 28
- answerPreCheckoutQuery() (*telegram.Bot* method), 29
- answerShippingQuery() (*telegram.Bot* method), 29
- answerWebAppQuery() (*telegram.Bot* method), 29
- ANY_CHAT_BOOST (*telegram.ext.ChatBoostHandler* attribute), 578
- ANY_CHAT_MEMBER (*telegram.ext.ChatMemberHandler* attribute), 581
- api_kwargs (*telegram.TelegramObject* attribute), 372
- APK (*telegram.ext.filters.Document* attribute), 600
- Application (class in *telegram.ext*), 517
- application (*telegram.ext.CallbackContext* property), 549
- APPLICATION (*telegram.ext.filters.Document* attribute), 599
- application (*telegram.ext.JobQueue* property), 563
- application_class() (*telegram.ext.ApplicationBuilder* method), 531
- ApplicationBuilder (class in *telegram.ext*), 531
- ApplicationHandlerStop (class in *telegram.ext*), 546
- approve() (*telegram.ChatJoinRequest* method), 215
- approve_chat_join_request() (*telegram.Bot* method), 33
- approve_join_request() (*telegram.Chat* method), 181
- approve_join_request() (*telegram.User* method), 386
- approveChatJoinRequest() (*telegram.Bot* method), 33
- arbitrary_callback_data() (*telegram.ext.ApplicationBuilder* method), 532
- args (*telegram.ext.CallbackContext* attribute), 549
- ARTICLE (*telegram.constants.InlineQueryResultType* attribute), 779
- as_integer_ratio() (*telegram.constants.BotCommandLimit* method), 656
- as_integer_ratio() (*telegram.constants.BotDescriptionLimit* method), 666
- as_integer_ratio() (*telegram.constants.BotNameLimit* method), 670
- as_integer_ratio() (*telegram.constants.BulkRequestLimit* method), 674
- as_integer_ratio() (*telegram.constants.CallbackQueryLimit* method), 678
- as_integer_ratio() (*telegram.constants.ChatID* method), 695
- as_integer_ratio() (*telegram.constants.ChatInviteLinkLimit* method), 700
- as_integer_ratio() (*telegram.constants.ChatLimit* method), 704
- as_integer_ratio() (*telegram.constants.ChatPhotoSize* method), 714
- as_integer_ratio() (*telegram.constants.ContactLimit* method), 724
- as_integer_ratio() (*telegram.constants.CustomEmojiStickerLimit* method), 728
- as_integer_ratio() (*telegram.constants.DiceLimit* method), 739
- as_integer_ratio() (*telegram.constants.FileSizeLimit* method), 744
- as_integer_ratio() (*telegram.constants.FloodLimit* method), 748
- as_integer_ratio() (*telegram.constants.ForumIconColor* method), 752
- as_integer_ratio() (*telegram.constants.ForumTopicLimit* method), 757
- as_integer_ratio() (tele-

- `gram.constants.GiveawayLimit` (method), 761
- `as_integer_ratio()` (`telegram.constants.InlineKeyboardButtonLimit` method), 765
- `as_integer_ratio()` (`telegram.constants.InlineKeyboardMarkupLimit` method), 769
- `as_integer_ratio()` (`telegram.constants.InlineQueryLimit` method), 774
- `as_integer_ratio()` (`telegram.constants.InlineQueryResultLimit` method), 778
- `as_integer_ratio()` (`telegram.constants.InlineQueryResultsButtonLimit` method), 789
- `as_integer_ratio()` (`telegram.constants.InvoiceLimit` method), 800
- `as_integer_ratio()` (`telegram.constants.KeyboardButtonRequestUsersLimit` method), 804
- `as_integer_ratio()` (`telegram.constants.LocationLimit` method), 810
- `as_integer_ratio()` (`telegram.constants.MediaGroupLimit` method), 820
- `as_integer_ratio()` (`telegram.constants.MessageLimit` method), 845
- `as_integer_ratio()` (`telegram.constants.PollingLimit` method), 883
- `as_integer_ratio()` (`telegram.constants.PollLimit` method), 873
- `as_integer_ratio()` (`telegram.constants.ReplyLimit` method), 906
- `as_integer_ratio()` (`telegram.constants.StickerLimit` method), 917
- `as_integer_ratio()` (`telegram.constants.StickerSetLimit` method), 922
- `as_integer_ratio()` (`telegram.constants.UserProfilePhotosLimit` method), 939
- `as_integer_ratio()` (`telegram.constants.WebhookLimit` method), 944
- `attach_name` (`telegram.InputFile` attribute), 265
- `attach_uri` (`telegram.InputFile` property), 265
- ATTACHMENT (in module `telegram.ext.filters`), 590
- Audio (class in `telegram`), 157
- AUDIO (in module `telegram.ext.filters`), 590
- AUDIO (`telegram.constants.InlineQueryResultType` attribute), 779
- AUDIO (`telegram.constants.InputMediaType` attribute), 790
- AUDIO (`telegram.constants.MessageAttachmentType` attribute), 827
- AUDIO (`telegram.constants.MessageType` attribute), 853
- AUDIO (`telegram.ext.filters.Document` attribute), 599
- audio (`telegram.ExternalReplyInfo` attribute), 241
- audio (`telegram.Message` attribute), 301
- audio_duration (`telegram.InlineQueryResultAudio` attribute), 430
- audio_file_id (`telegram.InlineQueryResultCachedAudio` attribute), 432
- audio_url (`telegram.InlineQueryResultAudio` attribute), 430
- author_signature (`telegram.Message` attribute), 305
- author_signature (`telegram.MessageOriginChannel` attribute), 346
- author_signature (`telegram.MessageOriginChat` attribute), 346
- available_reactions (`telegram.Chat` attribute), 179
- B**
- background_custom_emoji_id (`telegram.Chat` attribute), 179
- BadRequest, 945
- ban_chat() (`telegram.Chat` method), 181
- ban_chat_member() (`telegram.Bot` method), 34
- ban_chat_sender_chat() (`telegram.Bot` method), 35
- ban_member() (`telegram.Chat` method), 181
- ban_sender_chat() (`telegram.Chat` method), 181
- BANANA (`telegram.constants.ReactionEmoji` attribute), 886
- banChatMember() (`telegram.Bot` method), 34
- banChatSenderChat() (`telegram.Bot` method), 34
- bank_statement (`telegram.SecureData` attribute), 514
- BANNED (`telegram.ChatMember` attribute), 218
- BANNED (`telegram.constants.ChatMemberStatus` attribute), 706
- base_file_url (`telegram.Bot` property), 36
- base_file_url() (`telegram.ext.ApplicationBuilder` method), 532
- base_url (`telegram.Bot` property), 36
- base_url() (`telegram.ext.ApplicationBuilder` method), 532
- BaseFilter (class in `telegram.ext.filters`), 592
- BaseHandler (class in `telegram.ext`), 574
- BasePersistence (class in `telegram.ext`), 630
- BaseRateLimiter (class in `telegram.ext`), 647
- BaseRequest (class in `telegram.request`), 950
- BaseUpdateProcessor (class in `telegram.ext`), 546
- BASKETBALL (`telegram.constants.DiceEmoji` attribute), 730
- BASKETBALL (`telegram.Dice` attribute), 237
- BASKETBALL (`telegram.ext.filters.Dice` attribute), 598
- BIG (`telegram.constants.ChatPhotoSize` attribute), 712
- big_file_id (`telegram.ChatPhoto` attribute), 233

`big_file_unique_id` (*telegram.ChatPhoto* attribute), 233

`bio` (*telegram.Chat* attribute), 177

`bio` (*telegram.ChatJoinRequest* attribute), 215

`birth_date` (*telegram.PersonalDetails* attribute), 511

`bit_count()` (*telegram.constants.BotCommandLimit* method), 656

`bit_count()` (*telegram.constants.BotDescriptionLimit* method), 666

`bit_count()` (*telegram.constants.BotNameLimit* method), 670

`bit_count()` (*telegram.constants.BulkRequestLimit* method), 674

`bit_count()` (*telegram.constants.CallbackQueryLimit* method), 678

`bit_count()` (*telegram.constants.ChatID* method), 696

`bit_count()` (*telegram.constants.ChatInviteLinkLimit* method), 700

`bit_count()` (*telegram.constants.ChatLimit* method), 704

`bit_count()` (*telegram.constants.ChatPhotoSize* method), 714

`bit_count()` (*telegram.constants.ContactLimit* method), 725

`bit_count()` (*telegram.constants.CustomEmojiStickerLimit* method), 729

`bit_count()` (*telegram.constants.DiceLimit* method), 739

`bit_count()` (*telegram.constants.FileSizeLimit* method), 744

`bit_count()` (*telegram.constants.FloodLimit* method), 748

`bit_count()` (*telegram.constants.ForumIconColor* method), 753

`bit_count()` (*telegram.constants.ForumTopicLimit* method), 757

`bit_count()` (*telegram.constants.GiveawayLimit* method), 761

`bit_count()` (*telegram.constants.InlineKeyboardButtonLimit* method), 765

`bit_count()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 769

`bit_count()` (*telegram.constants.InlineQueryLimit* method), 774

`bit_count()` (*telegram.constants.InlineQueryResultLimit* method), 778

`bit_count()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 789

`bit_count()` (*telegram.constants.InvoiceLimit* method), 800

`bit_count()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 804

`bit_count()` (*telegram.constants.LocationLimit* method), 810

`bit_count()` (*telegram.constants.MediaGroupLimit* method), 820

`bit_count()` (*telegram.constants.MessageLimit* method), 845

`bit_count()` (*telegram.constants.PollingLimit* method), 883

`bit_count()` (*telegram.constants.PollLimit* method), 873

`bit_count()` (*telegram.constants.ReplyLimit* method), 907

`bit_count()` (*telegram.constants.StickerLimit* method), 917

`bit_count()` (*telegram.constants.StickerSetLimit* method), 922

`bit_count()` (*telegram.constants.UserProfilePhotosLimit* method), 939

`bit_count()` (*telegram.constants.WebhookLimit* method), 944

`bit_length()` (*telegram.constants.BotCommandLimit* method), 656

`bit_length()` (*telegram.constants.BotDescriptionLimit* method), 666

`bit_length()` (*telegram.constants.BotNameLimit* method), 670

`bit_length()` (*telegram.constants.BulkRequestLimit* method), 674

`bit_length()` (*telegram.constants.CallbackQueryLimit* method), 679

`bit_length()` (*telegram.constants.ChatID* method), 696

`bit_length()` (*telegram.constants.ChatInviteLinkLimit* method), 700

`bit_length()` (*telegram.constants.ChatLimit* method), 704

`bit_length()` (*telegram.constants.ChatPhotoSize* method), 715

`bit_length()` (*telegram.constants.ContactLimit* method), 725

`bit_length()` (*telegram.constants.CustomEmojiStickerLimit* method), 729

`bit_length()` (*telegram.constants.DiceLimit* method), 740

`bit_length()` (*telegram.constants.FileSizeLimit* method), 744

`bit_length()` (*telegram.constants.FloodLimit* method), 748

`bit_length()` (*telegram.constants.ForumIconColor* method), 753

`bit_length()` (*telegram.constants.ForumTopicLimit* method), 757

`bit_length()` (*telegram.constants.GiveawayLimit* method), 761

`bit_length()` (*telegram.constants.InlineKeyboardButtonLimit* method), 765

`bit_length()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 769

`bit_length()` (*telegram.constants.InlineQueryLimit* method), 774

`bit_length()` (*telegram.constants.InlineQueryResultLimit* method), 778

`bit_length()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 789

`bit_length()` (*telegram.constants.InvoiceLimit* method), 800

`bit_length()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 804

`bit_length()` (*telegram.constants.LocationLimit* method), 810

`bit_length()` (*telegram.constants.MediaGroupLimit* method), 820

`bit_length()` (*telegram.constants.MessageLimit* method), 845

`bit_length()` (*telegram.constants.PollingLimit* method), 883

`bit_length()` (*telegram.constants.PollLimit* method), 873

`bit_length()` (*telegram.constants.ReplyLimit* method), 907

`bit_length()` (*telegram.constants.StickerLimit* method), 917

`bit_length()` (*telegram.constants.StickerSetLimit* method), 922

`bit_length()` (*telegram.constants.UserProfilePhotosLimit* method), 939

`bit_length()` (*telegram.constants.WebhookLimit* method), 944

- `gram.constants.InlineKeyboardMarkupLimit` (method), 769
- `bit_length()` (`telegram.constants.InlineQueryLimit` method), 774
- `bit_length()` (`telegram.constants.InlineQueryResultLimit` method), 778
- `bit_length()` (`telegram.constants.InlineQueryResultsButtonLimit` method), 789
- `bit_length()` (`telegram.constants.InvoiceLimit` method), 800
- `bit_length()` (`telegram.constants.KeyboardButtonRequestUsersLimit` method), 805
- `bit_length()` (`telegram.constants.LocationLimit` method), 810
- `bit_length()` (`telegram.constants.MediaGroupLimit` method), 820
- `bit_length()` (`telegram.constants.MessageLimit` method), 846
- `bit_length()` (`telegram.constants.PollingLimit` method), 883
- `bit_length()` (`telegram.constants.PollLimit` method), 873
- `bit_length()` (`telegram.constants.ReplyLimit` method), 907
- `bit_length()` (`telegram.constants.StickerLimit` method), 917
- `bit_length()` (`telegram.constants.StickerSetLimit` method), 922
- `bit_length()` (`telegram.constants.UserProfilePhotosLimit` method), 940
- `bit_length()` (`telegram.constants.WebhookLimit` method), 944
- `block` (`telegram.ext.BaseHandler` attribute), 575
- `block` (`telegram.ext.CallbackQueryHandler` attribute), 577
- `block` (`telegram.ext.ChatBoostHandler` attribute), 578
- `block` (`telegram.ext.ChatJoinRequestHandler` attribute), 580
- `block` (`telegram.ext.ChatMemberHandler` attribute), 581
- `block` (`telegram.ext.ChosenInlineResultHandler` attribute), 582
- `block` (`telegram.ext.CommandHandler` attribute), 584
- `block` (`telegram.ext.ConversationHandler` attribute), 587
- `block` (`telegram.ext.Defaults` property), 555
- `block` (`telegram.ext.InlineQueryHandler` attribute), 616
- `block` (`telegram.ext.MessageHandler` attribute), 617
- `block` (`telegram.ext.MessageReactionHandler` attribute), 619
- `block` (`telegram.ext.PollAnswerHandler` attribute), 620
- `block` (`telegram.ext.PollHandler` attribute), 621
- `block` (`telegram.ext.PreCheckoutQueryHandler` attribute), 622
- `block` (`telegram.ext.PrefixHandler` attribute), 624
- `block` (`telegram.ext.ShippingQueryHandler` attribute), 625
- `block` (`telegram.ext.StringCommandHandler` attribute), 626
- `block` (`telegram.ext.StringRegexHandler` attribute), 628
- `block` (`telegram.ext.TypeHandler` attribute), 629
- `BLOCKQUOTE` (`telegram.constants.MessageEntityType` attribute), 835
- `BLOCKQUOTE` (`telegram.MessageEntity` attribute), 342
- `BLUE` (`telegram.constants.ForumIconColor` attribute), 749
- `BOLD` (`telegram.constants.MessageEntityType` attribute), 835
- `BOLD` (`telegram.MessageEntity` attribute), 342
- `boost` (`telegram.ChatBoostUpdated` attribute), 211
- `boost_id` (`telegram.ChatBoost` attribute), 206
- `boost_id` (`telegram.ChatBoostRemoved` attribute), 207
- `boosts` (`telegram.UserChatBoosts` attribute), 402
- `Bot` (class in `telegram`), 22
- `bot` (`telegram.Bot` property), 36
- `bot` (`telegram.ext.Application` attribute), 518
- `bot` (`telegram.ext.BasePersistence` attribute), 631
- `bot` (`telegram.ext.CallbackContext` property), 549
- `bot` (`telegram.ext.CallbackDataCache` attribute), 644
- `bot` (`telegram.ext.Updater` attribute), 570
- `bot()` (`telegram.ext.ApplicationBuilder` method), 533
- `bot_administrator_rights` (`telegram.KeyboardButtonRequestChat` attribute), 283
- `BOT_API_VERSION` (in module `telegram.constants`), 652
- `BOT_API_VERSION_INFO` (in module `telegram.constants`), 653
- `BOT_COMMAND` (`telegram.constants.MessageEntityType` attribute), 835
- `BOT_COMMAND` (`telegram.MessageEntity` attribute), 342
- `bot_data` (`telegram.ext.Application` attribute), 518
- `bot_data` (`telegram.ext.CallbackContext` property), 549
- `bot_data` (`telegram.ext.ContextTypes` property), 553
- `bot_data` (`telegram.ext.DictPersistence` property), 636
- `bot_data` (`telegram.ext.PersistenceInput` attribute), 639
- `bot_data_json` (`telegram.ext.DictPersistence` property), 636
- `bot_ids` (`telegram.ext.filters.ViaBot` property), 614
- `bot_is_member` (`telegram.KeyboardButtonRequestChat` attribute), 283
- `bot_username` (`telegram.LoginUrl` attribute), 288
- `BotCommand` (class in `telegram`), 159
- `BotCommandLimit` (class in `telegram.constants`), 653
- `BotCommandScope` (class in `telegram`), 160
- `BotCommandScopeAllChatAdministrators` (class in `telegram`), 161

- `BotCommandScopeAllGroupChats` (class in `telegram`), 161
- `BotCommandScopeAllPrivateChats` (class in `telegram`), 162
- `BotCommandScopeChat` (class in `telegram`), 162
- `BotCommandScopeChatAdministrators` (class in `telegram`), 163
- `BotCommandScopeChatMember` (class in `telegram`), 163
- `BotCommandScopeDefault` (class in `telegram`), 164
- `BotCommandScopeType` (class in `telegram.constants`), 657
- `BotDescription` (class in `telegram`), 164
- `BotDescriptionLimit` (class in `telegram.constants`), 663
- `BotName` (class in `telegram`), 165
- `BotNameLimit` (class in `telegram.constants`), 667
- `BotShortDescription` (class in `telegram`), 165
- `BOTTLE_WITH_POPPING_CORK` (`telegram.constants.ReactionEmoji` attribute), 886
- `BOWLING` (`telegram.constants.DiceEmoji` attribute), 730
- `BOWLING` (`telegram.Dice` attribute), 237
- `BOWLING` (`telegram.ext.filters.Dice` attribute), 598
- `BROKEN_HEART` (`telegram.constants.ReactionEmoji` attribute), 886
- `build()` (`telegram.ext.ApplicationBuilder` method), 533
- `build_reply_arguments()` (`telegram.Message` method), 309
- `builder()` (`telegram.ext.Application` static method), 521
- `BulkRequestLimit` (class in `telegram.constants`), 671
- `button_text` (`telegram.WebAppData` attribute), 412
- `BUTTONS_PER_ROW` (`telegram.constants.InlineKeyboardMarkupLimit` attribute), 766
- C**
- `callback` (`telegram.ext.BaseHandler` attribute), 575
- `callback` (`telegram.ext.CallbackQueryHandler` attribute), 577
- `callback` (`telegram.ext.ChatBoostHandler` attribute), 578
- `callback` (`telegram.ext.ChatJoinRequestHandler` attribute), 580
- `callback` (`telegram.ext.ChatMemberHandler` attribute), 581
- `callback` (`telegram.ext.ChosenInlineResultHandler` attribute), 582
- `callback` (`telegram.ext.CommandHandler` attribute), 584
- `callback` (`telegram.ext.InlineQueryHandler` attribute), 615
- `callback` (`telegram.ext.Job` attribute), 560
- `callback` (`telegram.ext.MessageHandler` attribute), 617
- `callback` (`telegram.ext.MessageReactionHandler` attribute), 619
- `callback` (`telegram.ext.PollAnswerHandler` attribute), 620
- `callback` (`telegram.ext.PollHandler` attribute), 621
- `callback` (`telegram.ext.PreCheckoutQueryHandler` attribute), 622
- `callback` (`telegram.ext.PrefixHandler` attribute), 624
- `callback` (`telegram.ext.ShippingQueryHandler` attribute), 625
- `callback` (`telegram.ext.StringCommandHandler` attribute), 626
- `callback` (`telegram.ext.StringRegexHandler` attribute), 628
- `callback` (`telegram.ext.TypeHandler` attribute), 629
- `callback_data` (`telegram.ext.DictPersistence` property), 636
- `callback_data` (`telegram.ext.InvalidCallbackData` attribute), 647
- `callback_data` (`telegram.ext.PersistenceInput` attribute), 640
- `callback_data` (`telegram.InlineKeyboardButton` attribute), 259
- `callback_data_cache` (`telegram.ext.ExtBot` property), 558
- `callback_data_json` (`telegram.ext.DictPersistence` property), 636
- `callback_game` (`telegram.InlineKeyboardButton` attribute), 259
- `CALLBACK_QUERY` (`telegram.constants.UpdateType` attribute), 929
- `CALLBACK_QUERY` (`telegram.Update` attribute), 381
- `callback_query` (`telegram.Update` attribute), 379
- `CallbackContext` (class in `telegram.ext`), 548
- `CallbackDataCache` (class in `telegram.ext`), 644
- `CallbackGame` (class in `telegram`), 490
- `CallbackQuery` (class in `telegram`), 166
- `CallbackQueryHandler` (class in `telegram.ext`), 576
- `CallbackQueryLimit` (class in `telegram.constants`), 675
- `can_add_web_page_previews` (`telegram.ChatMemberRestricted` attribute), 226
- `can_add_web_page_previews` (`telegram.ChatPermissions` attribute), 231
- `can_be_edited` (`telegram.ChatMemberAdministrator` attribute), 220
- `can_change_info` (`telegram.ChatAdministratorRights` attribute), 205
- `can_change_info` (`telegram.ChatMemberAdministrator` attribute), 220
- `can_change_info` (`telegram.ChatMemberRestricted` attribute), 226
- `can_change_info` (`telegram.ChatPermissions` attribute), 231
- `can_delete_messages` (`tele-`

<code>gram.ChatAdministratorRights</code> attribute), 204	<code>can_pin_messages</code> (<code>telegram.ChatAdministratorRights</code> attribute), 205
<code>can_delete_messages</code> (<code>telegram.ChatMemberAdministrator</code> attribute), 220	<code>can_pin_messages</code> (<code>telegram.ChatMemberAdministrator</code> attribute), 221
<code>can_delete_stories</code> (<code>telegram.ChatAdministratorRights</code> attribute), 205	<code>can_pin_messages</code> (<code>telegram.ChatMemberRestricted</code> attribute), 226
<code>can_delete_stories</code> (<code>telegram.ChatMemberAdministrator</code> attribute), 221	<code>can_pin_messages</code> (<code>telegram.ChatPermissions</code> attribute), 232
<code>can_edit_messages</code> (<code>telegram.ChatAdministratorRights</code> attribute), 205	<code>can_post_messages</code> (<code>telegram.ChatAdministratorRights</code> attribute), 205
<code>can_edit_messages</code> (<code>telegram.ChatMemberAdministrator</code> attribute), 221	<code>can_post_messages</code> (<code>telegram.ChatMemberAdministrator</code> attribute), 221
<code>can_edit_stories</code> (<code>telegram.ChatAdministratorRights</code> attribute), 205	<code>can_post_stories</code> (<code>telegram.ChatAdministratorRights</code> attribute), 205
<code>can_edit_stories</code> (<code>telegram.ChatMemberAdministrator</code> attribute), 221	<code>can_post_stories</code> (<code>telegram.ChatMemberAdministrator</code> attribute), 221
<code>can_invite_users</code> (<code>telegram.ChatAdministratorRights</code> attribute), 205	<code>can_promote_members</code> (<code>telegram.ChatAdministratorRights</code> attribute), 205
<code>can_invite_users</code> (<code>telegram.ChatMemberAdministrator</code> attribute), 220	<code>can_promote_members</code> (<code>telegram.ChatMemberAdministrator</code> attribute), 220
<code>can_invite_users</code> (<code>telegram.ChatMemberRestricted</code> attribute), 226	<code>can_read_all_group_messages</code> (<code>telegram.Bot</code> property), 36
<code>can_invite_users</code> (<code>telegram.ChatPermissions</code> attribute), 232	<code>can_read_all_group_messages</code> (<code>telegram.User</code> attribute), 386
<code>can_join_groups</code> (<code>telegram.Bot</code> property), 36	<code>can_restrict_members</code> (<code>telegram.ChatAdministratorRights</code> attribute), 204
<code>can_join_groups</code> (<code>telegram.User</code> attribute), 386	<code>can_restrict_members</code> (<code>telegram.ChatMemberAdministrator</code> attribute), 220
<code>can_manage_chat</code> (<code>telegram.ChatAdministratorRights</code> attribute), 204	<code>can_send_audios</code> (<code>telegram.ChatMemberRestricted</code> attribute), 227
<code>can_manage_chat</code> (<code>telegram.ChatMemberAdministrator</code> attribute), 220	<code>can_send_audios</code> (<code>telegram.ChatPermissions</code> attribute), 232
<code>can_manage_topics</code> (<code>telegram.ChatAdministratorRights</code> attribute), 205	<code>can_send_documents</code> (<code>telegram.ChatMemberRestricted</code> attribute), 227
<code>can_manage_topics</code> (<code>telegram.ChatMemberAdministrator</code> attribute), 221	<code>can_send_documents</code> (<code>telegram.ChatPermissions</code> attribute), 232
<code>can_manage_topics</code> (<code>telegram.ChatMemberRestricted</code> attribute), 227	<code>can_send_messages</code> (<code>telegram.ChatMemberRestricted</code> attribute), 226
<code>can_manage_topics</code> (<code>telegram.ChatPermissions</code> attribute), 232	<code>can_send_messages</code> (<code>telegram.ChatPermissions</code> attribute), 231
<code>can_manage_video_chats</code> (<code>telegram.ChatAdministratorRights</code> attribute), 204	<code>can_send_other_messages</code> (<code>telegram.ChatMemberRestricted</code> attribute), 226
<code>can_manage_video_chats</code> (<code>telegram.ChatMemberAdministrator</code> attribute), 220	<code>can_send_other_messages</code> (<code>telegram.ChatPermissions</code> attribute), 231
	<code>can_send_photos</code> (<code>telegram.ChatMemberRestricted</code>

- attribute*), 227
- `can_send_photos` (*telegram.ChatPermissions attribute*), 232
- `can_send_polls` (*telegram.ChatMemberRestricted attribute*), 226
- `can_send_polls` (*telegram.ChatPermissions attribute*), 231
- `can_send_video_notes` (*telegram.ChatMemberRestricted attribute*), 227
- `can_send_video_notes` (*telegram.ChatPermissions attribute*), 232
- `can_send_videos` (*telegram.ChatMemberRestricted attribute*), 227
- `can_send_videos` (*telegram.ChatPermissions attribute*), 232
- `can_send_voice_notes` (*telegram.ChatMemberRestricted attribute*), 227
- `can_send_voice_notes` (*telegram.ChatPermissions attribute*), 232
- `can_set_sticker_set` (*telegram.Chat attribute*), 178
- `capitalize()` (*telegram.constants.BotCommandScopeType method*), 659
- `capitalize()` (*telegram.constants.ChatAction method*), 682
- `capitalize()` (*telegram.constants.ChatBoostSources method*), 688
- `capitalize()` (*telegram.constants.ChatMemberStatus method*), 707
- `capitalize()` (*telegram.constants.ChatType method*), 717
- `capitalize()` (*telegram.constants.DiceEmoji method*), 732
- `capitalize()` (*telegram.constants.InlineQueryResultType method*), 782
- `capitalize()` (*telegram.constants.InputMediaType method*), 792
- `capitalize()` (*telegram.constants.MaskPosition method*), 813
- `capitalize()` (*telegram.constants.MenuButtonType method*), 823
- `capitalize()` (*telegram.constants.MessageAttachmentType method*), 830
- `capitalize()` (*telegram.constants.MessageEntityType method*), 837
- `capitalize()` (*telegram.constants.MessageOriginType method*), 848
- `capitalize()` (*telegram.constants.MessageType method*), 859
- `capitalize()` (*telegram.constants.ParseMode method*), 865
- `capitalize()` (*telegram.constants.PollType method*), 875
- `capitalize()` (*telegram.constants.ReactionEmoji method*), 893
- `capitalize()` (*telegram.constants.ReactionType method*), 899
- `capitalize()` (*telegram.constants.StickerFormat method*), 909
- `capitalize()` (*telegram.constants.StickerType method*), 925
- `capitalize()` (*telegram.constants.UpdateType method*), 932
- `Caption` (*class in telegram.ext.filters*), 594
- `CAPTION` (*in module telegram.ext.filters*), 590
- `caption` (*telegram.InlineQueryResultAudio attribute*), 430
- `caption` (*telegram.InlineQueryResultCachedAudio attribute*), 432
- `caption` (*telegram.InlineQueryResultCachedDocument attribute*), 434
- `caption` (*telegram.InlineQueryResultCachedGif attribute*), 436
- `caption` (*telegram.InlineQueryResultCachedMpeg4Gif attribute*), 438
- `caption` (*telegram.InlineQueryResultCachedPhoto attribute*), 439
- `caption` (*telegram.InlineQueryResultCachedVideo attribute*), 442
- `caption` (*telegram.InlineQueryResultCachedVoice attribute*), 444
- `caption` (*telegram.InlineQueryResultDocument attribute*), 448
- `caption` (*telegram.InlineQueryResultGif attribute*), 452
- `caption` (*telegram.InlineQueryResultMpeg4Gif attribute*), 457
- `caption` (*telegram.InlineQueryResultPhoto attribute*), 460
- `caption` (*telegram.InlineQueryResultVideo attribute*), 466
- `caption` (*telegram.InlineQueryResultVoice attribute*), 468
- `caption` (*telegram.InputMedia attribute*), 266
- `caption` (*telegram.InputMediaAnimation attribute*), 268
- `caption` (*telegram.InputMediaAudio attribute*), 270
- `caption` (*telegram.InputMediaDocument attribute*), 272
- `caption` (*telegram.InputMediaPhoto attribute*), 274
- `caption` (*telegram.InputMediaVideo attribute*), 276
- `caption` (*telegram.Message attribute*), 303
- `caption_entities` (*telegram.InlineQueryResultAudio attribute*), 431
- `caption_entities` (*telegram.InlineQueryResultCachedAudio attribute*), 432
- `caption_entities` (*tele-*

- gram.InlineQueryResultCachedDocument* attribute), 434
- caption_entities* (*telegram.InlineQueryResultCachedGif* attribute), 436
- caption_entities* (*telegram.InlineQueryResultCachedMpeg4Gif* attribute), 438
- caption_entities* (*telegram.InlineQueryResultCachedPhoto* attribute), 440
- caption_entities* (*telegram.InlineQueryResultCachedVideo* attribute), 442
- caption_entities* (*telegram.InlineQueryResultCachedVoice* attribute), 444
- caption_entities* (*telegram.InlineQueryResultDocument* attribute), 448
- caption_entities* (*telegram.InlineQueryResultGif* attribute), 452
- caption_entities* (*telegram.InlineQueryResultMpeg4Gif* attribute), 458
- caption_entities* (*telegram.InlineQueryResultPhoto* attribute), 460
- caption_entities* (*telegram.InlineQueryResultVideo* attribute), 466
- caption_entities* (*telegram.InlineQueryResultVoice* attribute), 468
- caption_entities* (*telegram.InputMedia* attribute), 267
- caption_entities* (*telegram.InputMediaAnimation* attribute), 268
- caption_entities* (*telegram.InputMediaAudio* attribute), 271
- caption_entities* (*telegram.InputMediaDocument* attribute), 273
- caption_entities* (*telegram.InputMediaPhoto* attribute), 274
- caption_entities* (*telegram.InputMediaVideo* attribute), 276
- caption_entities* (*telegram.Message* attribute), 301
- caption_html* (*telegram.Message* property), 310
- caption_html_urled* (*telegram.Message* property), 310
- CAPTION_LENGTH* (*telegram.constants.MessageLimit* attribute), 842
- caption_markdown* (*telegram.Message* property), 310
- caption_markdown_urled* (*telegram.Message* property), 311
- caption_markdown_v2* (*telegram.Message* property), 312
- caption_markdown_v2_urled* (*telegram.Message* property), 312
- CaptionEntity* (class in *telegram.ext.filters*), 594
- CaptionRegex* (class in *telegram.ext.filters*), 594
- casefold()* (*telegram.constants.BotCommandScopeType* method), 659
- casefold()* (*telegram.constants.ChatAction* method), 682
- casefold()* (*telegram.constants.ChatBoostSources* method), 688
- casefold()* (*telegram.constants.ChatMemberStatus* method), 707
- casefold()* (*telegram.constants.ChatType* method), 717
- casefold()* (*telegram.constants.DiceEmoji* method), 732
- casefold()* (*telegram.constants.InlineQueryResultType* method), 782
- casefold()* (*telegram.constants.InputMediaType* method), 792
- casefold()* (*telegram.constants.MaskPosition* method), 813
- casefold()* (*telegram.constants.MenuButtonType* method), 823
- casefold()* (*telegram.constants.MessageAttachmentType* method), 830
- casefold()* (*telegram.constants.MessageEntityType* method), 837
- casefold()* (*telegram.constants.MessageOriginType* method), 848
- casefold()* (*telegram.constants.MessageType* method), 859
- casefold()* (*telegram.constants.ParseMode* method), 865
- casefold()* (*telegram.constants.PollType* method), 875
- casefold()* (*telegram.constants.ReactionEmoji* method), 893
- casefold()* (*telegram.constants.ReactionType* method), 899
- casefold()* (*telegram.constants.StickerFormat* method), 909
- casefold()* (*telegram.constants.StickerType* method), 925
- casefold()* (*telegram.constants.UpdateType* method), 932
- CASHTAG* (*telegram.constants.MessageEntityType* attribute), 835
- CASHTAG* (*telegram.MessageEntity* attribute), 343
- center()* (*telegram.constants.BotCommandScopeType* method), 659
- center()* (*telegram.constants.ChatAction* method), 682
- center()* (*telegram.constants.ChatBoostSources* method), 688
- center()* (*telegram.constants.ChatMemberStatus* method), 707
- center()* (*telegram.constants.ChatType* method), 717
- center()* (*telegram.constants.DiceEmoji* method), 732

- `center()` (*telegram.constants.InlineQueryResultType* method), 782
- `center()` (*telegram.constants.InputMediaType* method), 792
- `center()` (*telegram.constants.MaskPosition* method), 813
- `center()` (*telegram.constants.MenuButtonType* method), 823
- `center()` (*telegram.constants.MessageAttachmentType* method), 830
- `center()` (*telegram.constants.MessageEntityType* method), 837
- `center()` (*telegram.constants.MessageOriginType* method), 848
- `center()` (*telegram.constants.MessageType* method), 859
- `center()` (*telegram.constants.ParseMode* method), 865
- `center()` (*telegram.constants.PollType* method), 876
- `center()` (*telegram.constants.ReactionEmoji* method), 893
- `center()` (*telegram.constants.ReactionType* method), 899
- `center()` (*telegram.constants.StickerFormat* method), 909
- `center()` (*telegram.constants.StickerType* method), 925
- `center()` (*telegram.constants.UpdateType* method), 932
- `CHANNEL` (*telegram.Chat* attribute), 180
- `CHANNEL` (*telegram.constants.ChatType* attribute), 716
- `CHANNEL` (*telegram.constants.MessageOriginType* attribute), 847
- `CHANNEL` (*telegram.ext.filters.ChatType* attribute), 596
- `CHANNEL` (*telegram.ext.filters.SenderChat* attribute), 606
- `CHANNEL` (*telegram.MessageOrigin* attribute), 345
- `CHANNEL_CHAT_CREATED` (*telegram.constants.MessageType* attribute), 853
- `channel_chat_created` (*telegram.Message* attribute), 304
- `CHANNEL_POST` (*telegram.constants.UpdateType* attribute), 929
- `CHANNEL_POST` (*telegram.ext.filters.UpdateType* attribute), 611
- `CHANNEL_POST` (*telegram.Update* attribute), 381
- `channel_post` (*telegram.Update* attribute), 379
- `CHANNEL_POSTS` (*telegram.ext.filters.UpdateType* attribute), 611
- `Chat` (class in *telegram*), 173
- `Chat` (class in *telegram.ext.filters*), 595
- `CHAT` (in module *telegram.ext.filters*), 590
- `CHAT` (*telegram.BotCommandScope* attribute), 160
- `chat` (*telegram.ChatBoostRemoved* attribute), 207
- `chat` (*telegram.ChatBoostUpdated* attribute), 211
- `chat` (*telegram.ChatJoinRequest* attribute), 214
- `chat` (*telegram.ChatMemberUpdated* attribute), 228
- `CHAT` (*telegram.constants.BotCommandScopeType* attribute), 657
- `CHAT` (*telegram.constants.MessageOriginType* attribute), 847
- `chat` (*telegram.ExternalReplyInfo* attribute), 241
- `chat` (*telegram.GiveawayWinners* attribute), 254
- `chat` (*telegram.InaccessibleMessage* attribute), 256
- `chat` (*telegram.MaybeInaccessibleMessage* attribute), 289
- `chat` (*telegram.Message* attribute), 300
- `CHAT` (*telegram.MessageOrigin* attribute), 345
- `chat` (*telegram.MessageOriginChannel* attribute), 345
- `chat` (*telegram.MessageReactionCountUpdated* attribute), 348
- `chat` (*telegram.MessageReactionUpdated* attribute), 349
- `CHAT_ADMINISTRATOR_CUSTOM_TITLE_LENGTH` (*telegram.constants.ChatLimit* attribute), 701
- `CHAT_ADMINISTRATORS` (*telegram.BotCommandScope* attribute), 160
- `CHAT_ADMINISTRATORS` (*telegram.constants.BotCommandScopeType* attribute), 658
- `CHAT_BOOST` (*telegram.constants.UpdateType* attribute), 929
- `CHAT_BOOST` (*telegram.ext.ChatBoostHandler* attribute), 578
- `CHAT_BOOST` (*telegram.Update* attribute), 381
- `chat_boost` (*telegram.Update* attribute), 380
- `chat_boost_types` (*telegram.ext.ChatBoostHandler* attribute), 578
- `CHAT_CREATED` (*telegram.ext.filters.StatusUpdate* attribute), 607
- `chat_data` (*telegram.ext.Application* attribute), 518
- `chat_data` (*telegram.ext.CallbackContext* property), 549
- `chat_data` (*telegram.ext.ContextTypes* property), 553
- `chat_data` (*telegram.ext.DictPersistence* property), 636
- `chat_data` (*telegram.ext.PersistenceInput* attribute), 639
- `chat_data_json` (*telegram.ext.DictPersistence* property), 636
- `CHAT_DESCRIPTION_LENGTH` (*telegram.constants.ChatLimit* attribute), 701
- `chat_has_username` (*telegram.KeyboardButtonRequestChat* attribute), 282
- `chat_id` (*telegram.BotCommandScopeChat* attribute), 162
- `chat_id` (*telegram.BotCommandScopeChatAdministrators* attribute), 163
- `chat_id` (*telegram.BotCommandScopeChatMember* attribute), 164
- `chat_id` (*telegram.ChatShared* attribute), 235
- `chat_id` (*telegram.ext.Job* attribute), 560
- `chat_id` (*telegram.Message* property), 313
- `chat_id` (*telegram.ReplyParameters* attribute), 369

[chat_ids \(telegram.ext.filters.Chat attribute\), 595](#)
[chat_ids \(telegram.ext.filters.ForwardedFrom attribute\), 602](#)
[chat_ids \(telegram.ext.filters.SenderChat attribute\), 605](#)
[chat_instance \(telegram.CallbackQuery attribute\), 167](#)
[chat_is_channel \(telegram.KeyboardButtonRequestChat attribute\), 282](#)
[chat_is_created \(telegram.KeyboardButtonRequestChat attribute\), 282](#)
[chat_is_forum \(telegram.KeyboardButtonRequestChat attribute\), 282](#)
[CHAT_JOIN_REQUEST \(telegram.constants.UpdateType attribute\), 930](#)
[CHAT_JOIN_REQUEST \(telegram.Update attribute\), 381](#)
[chat_join_request \(telegram.Update attribute\), 380](#)
[CHAT_MEMBER \(telegram.BotCommandScope attribute\), 160](#)
[CHAT_MEMBER \(telegram.constants.BotCommandScopeType attribute\), 658](#)
[CHAT_MEMBER \(telegram.constants.UpdateType attribute\), 930](#)
[CHAT_MEMBER \(telegram.ext.ChatMemberHandler attribute\), 581](#)
[CHAT_MEMBER \(telegram.Update attribute\), 381](#)
[chat_member \(telegram.Update attribute\), 380](#)
[chat_member_types \(telegram.ext.ChatMemberHandler attribute\), 581](#)
[CHAT_SHARED \(telegram.constants.MessageType attribute\), 853](#)
[CHAT_SHARED \(telegram.ext.filters.StatusUpdate attribute\), 607](#)
[chat_shared \(telegram.Message attribute\), 308](#)
[chat_type \(telegram.InlineQuery attribute\), 425](#)
[chat_types \(telegram.ext.InlineQueryHandler attribute\), 616](#)
[ChatAction \(class in telegram.constants\), 680](#)
[ChatAdministratorRights \(class in telegram\), 203](#)
[ChatBoost \(class in telegram\), 206](#)
[ChatBoostHandler \(class in telegram.ext\), 577](#)
[ChatBoostRemoved \(class in telegram\), 207](#)
[ChatBoostSource \(class in telegram\), 208](#)
[ChatBoostSourceGiftCode \(class in telegram\), 209](#)
[ChatBoostSourceGiveaway \(class in telegram\), 209](#)
[ChatBoostSourcePremium \(class in telegram\), 210](#)
[ChatBoostSources \(class in telegram.constants\), 686](#)
[ChatBoostUpdated \(class in telegram\), 211](#)
[ChatID \(class in telegram.constants\), 692](#)
[ChatInviteLink \(class in telegram\), 211](#)
[ChatInviteLinkLimit \(class in telegram.constants\), 697](#)
[ChatJoinRequest \(class in telegram\), 213](#)
[ChatJoinRequestHandler \(class in telegram.ext\), 579](#)
[ChatLimit \(class in telegram.constants\), 701](#)
[ChatLocation \(class in telegram\), 216](#)
[ChatMember \(class in telegram\), 217](#)
[ChatMemberAdministrator \(class in telegram\), 218](#)
[ChatMemberBanned \(class in telegram\), 222](#)
[ChatMemberHandler \(class in telegram.ext\), 580](#)
[ChatMemberLeft \(class in telegram\), 222](#)
[ChatMemberMember \(class in telegram\), 223](#)
[ChatMemberOwner \(class in telegram\), 224](#)
[ChatMemberRestricted \(class in telegram\), 224](#)
[ChatMemberStatus \(class in telegram.constants\), 705](#)
[ChatMemberUpdated \(class in telegram\), 228](#)
[ChatMigrated, 945](#)
[ChatPermissions \(class in telegram\), 230](#)
[ChatPhoto \(class in telegram\), 233](#)
[ChatPhotoSize \(class in telegram.constants\), 711](#)
[chats \(telegram.Giveaway attribute\), 251](#)
[ChatShared \(class in telegram\), 234](#)
[ChatType \(class in telegram.constants\), 716](#)
[ChatType \(class in telegram.ext.filters\), 596](#)
[check_update\(\) \(telegram.ext.BaseHandler method\), 575](#)
[check_update\(\) \(telegram.ext.CallbackQueryHandler method\), 577](#)
[check_update\(\) \(telegram.ext.ChatBoostHandler method\), 579](#)
[check_update\(\) \(telegram.ext.ChatJoinRequestHandler method\), 580](#)
[check_update\(\) \(telegram.ext.ChatMemberHandler method\), 581](#)
[check_update\(\) \(telegram.ext.ChosenInlineResultHandler method\), 582](#)
[check_update\(\) \(telegram.ext.CommandHandler method\), 585](#)
[check_update\(\) \(telegram.ext.ConversationHandler method\), 588](#)
[check_update\(\) \(telegram.ext.filters.BaseFilter method\), 594](#)
[check_update\(\) \(telegram.ext.filters.MessageFilter method\), 604](#)
[check_update\(\) \(telegram.ext.filters.UpdateFilter method\), 611](#)
[check_update\(\) \(telegram.ext.InlineQueryHandler method\), 616](#)
[check_update\(\) \(telegram.ext.MessageHandler method\), 617](#)
[check_update\(\) \(telegram.ext.MessageReactionHandler method\), 619](#)
[check_update\(\) \(telegram.ext.PollAnswerHandler method\), 620](#)
[check_update\(\) \(telegram.ext.PollHandler method\), 621](#)
[check_update\(\) \(telegram.ext.PollVoteHandler method\), 622](#)

- gram.ext.PreCheckoutQueryHandler* method), 623
- `check_update()` (*telegram.ext.PrefixHandler* method), 624
- `check_update()` (*telegram.ext.ShippingQueryHandler* method), 625
- `check_update()` (*telegram.ext.StringCommandHandler* method), 627
- `check_update()` (*telegram.ext.StringRegexHandler* method), 628
- `check_update()` (*telegram.ext.TypeHandler* method), 629
- CHIN (*telegram.constants.MaskPosition* attribute), 811
- CHIN (*telegram.MaskPosition* attribute), 417
- CHOOSE_STICKER (*telegram.constants.ChatAction* attribute), 680
- CHOSEN_INLINE_RESULT (*telegram.constants.UpdateType* attribute), 930
- CHOSEN_INLINE_RESULT (*telegram.Update* attribute), 381
- `chosen_inline_result` (*telegram.Update* attribute), 379
- ChosenInlineResult* (class in *telegram*), 422
- ChosenInlineResultHandler* (class in *telegram.ext*), 582
- CHRISTMAS_TREE (*telegram.constants.ReactionEmoji* attribute), 886
- `city` (*telegram.ResidentialAddress* attribute), 512
- `city` (*telegram.ShippingAddress* attribute), 487
- CLAPPING_HANDS (*telegram.constants.ReactionEmoji* attribute), 886
- `clear_callback_data()` (*telegram.ext.CallbackDataCache* method), 644
- `clear_callback_queries()` (*telegram.ext.CallbackDataCache* method), 645
- `close()` (*telegram.Bot* method), 36
- `close_date` (*telegram.Poll* attribute), 354
- `close_forum_topic()` (*telegram.Bot* method), 37
- `close_forum_topic()` (*telegram.Chat* method), 182
- `close_forum_topic()` (*telegram.Message* method), 313
- `close_general_forum_topic()` (*telegram.Bot* method), 38
- `close_general_forum_topic()` (*telegram.Chat* method), 182
- `closeForumTopic()` (*telegram.Bot* method), 37
- `closeGeneralForumTopic()` (*telegram.Bot* method), 37
- CLOWN_FACE (*telegram.constants.ReactionEmoji* attribute), 886
- CODE (*telegram.constants.MessageEntityType* attribute), 835
- CODE (*telegram.MessageEntity* attribute), 343
- `collect_additional_context()` (*telegram.ext.BaseHandler* method), 575
- `collect_additional_context()` (*telegram.ext.CallbackQueryHandler* method), 577
- `collect_additional_context()` (*telegram.ext.ChosenInlineResultHandler* method), 583
- `collect_additional_context()` (*telegram.ext.CommandHandler* method), 585
- `collect_additional_context()` (*telegram.ext.InlineQueryHandler* method), 616
- `collect_additional_context()` (*telegram.ext.MessageHandler* method), 617
- `collect_additional_context()` (*telegram.ext.PrefixHandler* method), 625
- `collect_additional_context()` (*telegram.ext.StringCommandHandler* method), 627
- `collect_additional_context()` (*telegram.ext.StringRegexHandler* method), 628
- COLOR_000 (*telegram.constants.AccentColor* attribute), 651
- COLOR_000 (*telegram.constants.ProfileAccentColor* attribute), 884
- COLOR_001 (*telegram.constants.AccentColor* attribute), 651
- COLOR_001 (*telegram.constants.ProfileAccentColor* attribute), 884
- COLOR_002 (*telegram.constants.AccentColor* attribute), 651
- COLOR_002 (*telegram.constants.ProfileAccentColor* attribute), 884
- COLOR_003 (*telegram.constants.AccentColor* attribute), 651
- COLOR_003 (*telegram.constants.ProfileAccentColor* attribute), 884
- COLOR_004 (*telegram.constants.AccentColor* attribute), 651
- COLOR_004 (*telegram.constants.ProfileAccentColor* attribute), 885
- COLOR_005 (*telegram.constants.AccentColor* attribute), 651
- COLOR_005 (*telegram.constants.ProfileAccentColor* attribute), 885
- COLOR_006 (*telegram.constants.AccentColor* attribute), 651
- COLOR_006 (*telegram.constants.ProfileAccentColor* attribute), 885
- COLOR_007 (*telegram.constants.AccentColor* attribute), 651
- COLOR_007 (*telegram.constants.ProfileAccentColor* attribute), 885
- COLOR_008 (*telegram.constants.AccentColor* attribute), 651
- COLOR_008 (*telegram.constants.ProfileAccentColor* attribute), 885

- [tribute](#)), 885
- [COLOR_009](#) ([telegram.constants.AccentColor](#) attribute), 651
- [COLOR_009](#) ([telegram.constants.ProfileAccentColor](#) attribute), 885
- [COLOR_010](#) ([telegram.constants.AccentColor](#) attribute), 652
- [COLOR_010](#) ([telegram.constants.ProfileAccentColor](#) attribute), 885
- [COLOR_011](#) ([telegram.constants.AccentColor](#) attribute), 652
- [COLOR_011](#) ([telegram.constants.ProfileAccentColor](#) attribute), 885
- [COLOR_012](#) ([telegram.constants.AccentColor](#) attribute), 652
- [COLOR_012](#) ([telegram.constants.ProfileAccentColor](#) attribute), 885
- [COLOR_013](#) ([telegram.constants.AccentColor](#) attribute), 652
- [COLOR_013](#) ([telegram.constants.ProfileAccentColor](#) attribute), 885
- [COLOR_014](#) ([telegram.constants.AccentColor](#) attribute), 652
- [COLOR_014](#) ([telegram.constants.ProfileAccentColor](#) attribute), 885
- [COLOR_015](#) ([telegram.constants.AccentColor](#) attribute), 652
- [COLOR_015](#) ([telegram.constants.ProfileAccentColor](#) attribute), 885
- [COLOR_016](#) ([telegram.constants.AccentColor](#) attribute), 652
- [COLOR_017](#) ([telegram.constants.AccentColor](#) attribute), 652
- [COLOR_018](#) ([telegram.constants.AccentColor](#) attribute), 652
- [COLOR_019](#) ([telegram.constants.AccentColor](#) attribute), 652
- [COLOR_020](#) ([telegram.constants.AccentColor](#) attribute), 652
- [Command](#) (class in [telegram.ext.filters](#)), 597
- [COMMAND](#) (in module [telegram.ext.filters](#)), 590
- [command](#) ([telegram.BotCommand](#) attribute), 159
- [command](#) ([telegram.ext.StringCommandHandler](#) attribute), 626
- [CommandHandler](#) (class in [telegram.ext](#)), 583
- [COMMANDS](#) ([telegram.constants.MenuButtonType](#) attribute), 821
- [commands](#) ([telegram.ext.CommandHandler](#) attribute), 584
- [commands](#) ([telegram.ext.PrefixHandler](#) attribute), 624
- [COMMANDS](#) ([telegram.MenuButton](#) attribute), 290
- [compute_quote_position_and_entities\(\)](#) ([telegram.Message](#) method), 313
- [concurrent_updates](#) ([telegram.ext.Application](#) property), 521
- [concurrent_updates\(\)](#) ([telegram.ext.ApplicationBuilder](#) method), 533
- [Conflict](#), 946
- [conjugate\(\)](#) ([telegram.constants.BotCommandLimit](#) method), 656
- [conjugate\(\)](#) ([telegram.constants.BotDescriptionLimit](#) method), 666
- [conjugate\(\)](#) ([telegram.constants.BotNameLimit](#) method), 670
- [conjugate\(\)](#) ([telegram.constants.BulkRequestLimit](#) method), 675
- [conjugate\(\)](#) ([telegram.constants.CallbackQueryLimit](#) method), 679
- [conjugate\(\)](#) ([telegram.constants.ChatID](#) method), 696
- [conjugate\(\)](#) ([telegram.constants.ChatInviteLinkLimit](#) method), 700
- [conjugate\(\)](#) ([telegram.constants.ChatLimit](#) method), 704
- [conjugate\(\)](#) ([telegram.constants.ChatPhotoSize](#) method), 715
- [conjugate\(\)](#) ([telegram.constants.ContactLimit](#) method), 725
- [conjugate\(\)](#) ([telegram.constants.CustomEmojiStickerLimit](#) method), 729
- [conjugate\(\)](#) ([telegram.constants.DiceLimit](#) method), 740
- [conjugate\(\)](#) ([telegram.constants.FileSizeLimit](#) method), 744
- [conjugate\(\)](#) ([telegram.constants.FloodLimit](#) method), 748
- [conjugate\(\)](#) ([telegram.constants.ForumIconColor](#) method), 753
- [conjugate\(\)](#) ([telegram.constants.ForumTopicLimit](#) method), 757
- [conjugate\(\)](#) ([telegram.constants.GiveawayLimit](#) method), 761
- [conjugate\(\)](#) ([telegram.constants.InlineKeyboardButtonLimit](#) method), 765
- [conjugate\(\)](#) ([telegram.constants.InlineKeyboardMarkupLimit](#) method), 770
- [conjugate\(\)](#) ([telegram.constants.InlineQueryLimit](#) method), 774
- [conjugate\(\)](#) ([telegram.constants.InlineQueryResultLimit](#) method), 778
- [conjugate\(\)](#) ([telegram.constants.InlineQueryResultsButtonLimit](#) method), 789
- [conjugate\(\)](#) ([telegram.constants.InvoiceLimit](#) method), 800
- [conjugate\(\)](#) ([telegram.constants.KeyboardButtonRequestUsersLimit](#) method), 805
- [conjugate\(\)](#) ([telegram.constants.LocationLimit](#) method), 810
- [conjugate\(\)](#) ([telegram.constants.MediaGroupLimit](#) method), 820
- [conjugate\(\)](#) ([telegram.constants.MessageLimit](#) method), 846
- [conjugate\(\)](#) ([telegram.constants.PollingLimit](#) method), 883
- [conjugate\(\)](#) ([telegram.constants.PollLimit](#) method),

- 873
- `conjugate()` (*telegram.constants.ReplyLimit* method), 907
- `conjugate()` (*telegram.constants.StickerLimit* method), 918
- `conjugate()` (*telegram.constants.StickerSetLimit* method), 922
- `conjugate()` (*telegram.constants.UserProfilePhotosLimit* method), 940
- `conjugate()` (*telegram.constants.WebhookLimit* method), 944
- `connect_timeout()` (*telegram.ext.ApplicationBuilder* method), 534
- `CONNECTED_WEBSITE` (*telegram.constants.MessageType* attribute), 853
- `CONNECTED_WEBSITE` (*telegram.ext.filters.StatusUpdate* attribute), 607
- `connected_website` (*telegram.Message* attribute), 305
- `connection_pool_size()` (*telegram.ext.ApplicationBuilder* method), 534
- `Contact` (class in *telegram*), 235
- `CONTACT` (in module *telegram.ext.filters*), 590
- `CONTACT` (*telegram.constants.InlineQueryResultType* attribute), 779
- `CONTACT` (*telegram.constants.MessageAttachmentType* attribute), 827
- `CONTACT` (*telegram.constants.MessageType* attribute), 853
- `contact` (*telegram.ExternalReplyInfo* attribute), 242
- `contact` (*telegram.Message* attribute), 303
- `ContactLimit` (class in *telegram.constants*), 722
- `contains_files` (*telegram.request.RequestData* attribute), 954
- `context` (*telegram.ext.ContextTypes* property), 553
- `context_types` (*telegram.ext.Application* attribute), 519
- `context_types` (*telegram.ext.PicklePersistence* attribute), 641
- `context_types()` (*telegram.ext.ApplicationBuilder* method), 534
- `ContextTypes` (class in *telegram.ext*), 552
- `conversation_timeout` (*telegram.ext.ConversationHandler* property), 588
- `ConversationHandler` (class in *telegram.ext*), 585
- `conversations` (*telegram.ext.DictPersistence* property), 637
- `conversations_json` (*telegram.ext.DictPersistence* property), 637
- `copy()` (*telegram.Message* method), 313
- `copy_message()` (*telegram.Bot* method), 39
- `copy_message()` (*telegram.CallbackQuery* method), 168
- `copy_message()` (*telegram.Chat* method), 182
- `copy_message()` (*telegram.User* method), 387
- `copy_messages()` (*telegram.Bot* method), 40
- `copy_messages()` (*telegram.Chat* method), 183
- `copy_messages()` (*telegram.User* method), 387
- `copyMessage()` (*telegram.Bot* method), 38
- `copyMessages()` (*telegram.Bot* method), 39
- `coroutine` (*telegram.ext.CallbackContext* attribute), 548
- `correct_option_id` (*telegram.Poll* attribute), 353
- `count()` (*telegram.constants.BotCommandScopeType* method), 659
- `count()` (*telegram.constants.ChatAction* method), 682
- `count()` (*telegram.constants.ChatBoostSources* method), 688
- `count()` (*telegram.constants.ChatMemberStatus* method), 707
- `count()` (*telegram.constants.ChatType* method), 717
- `count()` (*telegram.constants.DiceEmoji* method), 732
- `count()` (*telegram.constants.InlineQueryResultType* method), 782
- `count()` (*telegram.constants.InputMediaType* method), 792
- `count()` (*telegram.constants.MaskPosition* method), 813
- `count()` (*telegram.constants.MenuButtonType* method), 823
- `count()` (*telegram.constants.MessageAttachmentType* method), 830
- `count()` (*telegram.constants.MessageEntityType* method), 838
- `count()` (*telegram.constants.MessageOriginType* method), 848
- `count()` (*telegram.constants.MessageType* method), 859
- `count()` (*telegram.constants.ParseMode* method), 865
- `count()` (*telegram.constants.PollType* method), 876
- `count()` (*telegram.constants.ReactionEmoji* method), 894
- `count()` (*telegram.constants.ReactionType* method), 899
- `count()` (*telegram.constants.StickerFormat* method), 909
- `count()` (*telegram.constants.StickerType* method), 925
- `count()` (*telegram.constants.UpdateType* method), 932
- `country_code` (*telegram.PersonalDetails* attribute), 511
- `country_code` (*telegram.ResidentialAddress* attribute), 512
- `country_code` (*telegram.ShippingAddress* attribute), 486
- `country_codes` (*telegram.Giveaway* attribute), 252
- `create_chat_invite_link()` (*telegram.Bot* method), 42
- `create_deep_linked_url()` (in module *telegram.helpers*), 948
- `create_forum_topic()` (*telegram.Bot* method), 43
- `create_forum_topic()` (*telegram.Chat* method), 183

- `create_invite_link()` (*telegram.Chat* method), 183
- `create_invoice_link()` (*telegram.Bot* method), 44
- `create_new_sticker_set()` (*telegram.Bot* method), 45
- `create_task()` (*telegram.ext.Application* method), 521
- `createChatInviteLink()` (*telegram.Bot* method), 41
- `createForumTopic()` (*telegram.Bot* method), 42
- `createInvoiceLink()` (*telegram.Bot* method), 42
- `createNewStickerSet()` (*telegram.Bot* method), 42
- `creates_join_request` (*telegram.ChatInviteLink* attribute), 212
- `creator` (*telegram.ChatInviteLink* attribute), 212
- `Credentials` (class in *telegram*), 493
- `credentials` (*telegram.PassportData* attribute), 500
- `CRYING_FACE` (*telegram.constants.ReactionEmoji* attribute), 886
- `currency` (*telegram.InputInvoiceMessageContent* attribute), 479
- `currency` (*telegram.Invoice* attribute), 482
- `currency` (*telegram.PreCheckoutQuery* attribute), 485
- `currency` (*telegram.SuccessfulPayment* attribute), 489
- `CUSTOM_EMOJI` (*telegram.constants.MessageEntityType* attribute), 835
- `CUSTOM_EMOJI` (*telegram.constants.ReactionType* attribute), 898
- `CUSTOM_EMOJI` (*telegram.constants.StickerType* attribute), 923
- `CUSTOM_EMOJI` (*telegram.MessageEntity* attribute), 343
- `CUSTOM_EMOJI` (*telegram.ReactionType* attribute), 360
- `CUSTOM_EMOJI` (*telegram.Sticker* attribute), 420
- `custom_emoji_id` (*telegram.MessageEntity* attribute), 341
- `custom_emoji_id` (*telegram.ReactionTypeCustomEmoji* attribute), 360
- `custom_emoji_id` (*telegram.Sticker* attribute), 420
- `CUSTOM_EMOJI_IDENTIFIER_LIMIT` (*telegram.constants.CustomEmojiStickerLimit* attribute), 726
- `custom_title` (*telegram.ChatMemberAdministrator* attribute), 221
- `custom_title` (*telegram.ChatMemberOwner* attribute), 224
- `CustomEmojiStickerLimit` (class in *telegram.constants*), 726
- D**
- `DARTS` (*telegram.constants.DiceEmoji* attribute), 730
- `DARTS` (*telegram.Dice* attribute), 237
- `DARTS` (*telegram.ext.filters.Dice* attribute), 598
- `data` (*telegram.CallbackQuery* attribute), 167
- `data` (*telegram.EncryptedCredentials* attribute), 495
- `data` (*telegram.EncryptedPassportElement* attribute), 497
- `data` (*telegram.ext.Job* attribute), 560
- `data` (*telegram.PassportData* attribute), 500
- `data` (*telegram.SecureValue* attribute), 515
- `data` (*telegram.WebAppData* attribute), 412
- `data_filter` (*telegram.ext.filters.BaseFilter* property), 593
- `data_hash` (*telegram.PassportElementErrorDataField* attribute), 502
- `DataCredentials` (class in *telegram*), 494
- `date` (*telegram.ChatJoinRequest* attribute), 214
- `date` (*telegram.ChatMemberUpdated* attribute), 229
- `date` (*telegram.InaccessibleMessage* attribute), 256
- `date` (*telegram.MaybeInaccessibleMessage* attribute), 289
- `date` (*telegram.Message* attribute), 300
- `date` (*telegram.MessageOrigin* attribute), 344
- `date` (*telegram.MessageOriginChannel* attribute), 345
- `date` (*telegram.MessageOriginChat* attribute), 346
- `date` (*telegram.MessageOriginHiddenUser* attribute), 347
- `date` (*telegram.MessageOriginUser* attribute), 348
- `date` (*telegram.MessageReactionCountUpdated* attribute), 349
- `date` (*telegram.MessageReactionUpdated* attribute), 350
- `de_json()` (*telegram.Animation* class method), 156
- `de_json()` (*telegram.Audio* class method), 158
- `de_json()` (*telegram.BotCommandScope* class method), 161
- `de_json()` (*telegram.CallbackQuery* class method), 168
- `de_json()` (*telegram.Chat* class method), 183
- `de_json()` (*telegram.ChatBoost* class method), 207
- `de_json()` (*telegram.ChatBoostRemoved* class method), 208
- `de_json()` (*telegram.ChatBoostSource* class method), 208
- `de_json()` (*telegram.ChatBoostUpdated* class method), 211
- `de_json()` (*telegram.ChatInviteLink* class method), 213
- `de_json()` (*telegram.ChatJoinRequest* class method), 215
- `de_json()` (*telegram.ChatLocation* class method), 216
- `de_json()` (*telegram.ChatMember* class method), 218
- `de_json()` (*telegram.ChatMemberUpdated* class method), 229
- `de_json()` (*telegram.ChatPermissions* class method), 233
- `de_json()` (*telegram.ChosenInlineResult* class method), 423
- `de_json()` (*telegram.Credentials* class method), 494
- `de_json()` (*telegram.Document* class method), 239
- `de_json()` (*telegram.EncryptedPassportElement* class method), 498
- `de_json()` (*telegram.ExternalReplyInfo* class method), 243
- `de_json()` (*telegram.Game* class method), 492
- `de_json()` (*telegram.GameHighScore* class method),

- 493
- `de_json()` (*telegram.Giveaway* class method), 252
- `de_json()` (*telegram.GiveawayCompleted* class method), 253
- `de_json()` (*telegram.GiveawayWinners* class method), 255
- `de_json()` (*telegram.InlineKeyboardButton* class method), 260
- `de_json()` (*telegram.InlineKeyboardMarkup* class method), 263
- `de_json()` (*telegram.InlineQuery* class method), 426
- `de_json()` (*telegram.InlineQueryResultsButton* class method), 462
- `de_json()` (*telegram.InputInvoiceMessageContent* class method), 481
- `de_json()` (*telegram.KeyboardButton* class method), 280
- `de_json()` (*telegram.KeyboardButtonRequestChat* class method), 283
- `de_json()` (*telegram.MaybeInaccessibleMessage* class method), 289
- `de_json()` (*telegram.MenuButton* class method), 290
- `de_json()` (*telegram.MenuButtonWebApp* class method), 292
- `de_json()` (*telegram.Message* class method), 314
- `de_json()` (*telegram.MessageEntity* class method), 343
- `de_json()` (*telegram.MessageOrigin* class method), 345
- `de_json()` (*telegram.MessageReactionCountUpdated* class method), 349
- `de_json()` (*telegram.MessageReactionUpdated* class method), 350
- `de_json()` (*telegram.OrderInfo* class method), 484
- `de_json()` (*telegram.PassportData* class method), 500
- `de_json()` (*telegram.Poll* class method), 355
- `de_json()` (*telegram.PollAnswer* class method), 357
- `de_json()` (*telegram.PreCheckoutQuery* class method), 486
- `de_json()` (*telegram.ProximityAlertTriggered* class method), 358
- `de_json()` (*telegram.ReactionCount* class method), 359
- `de_json()` (*telegram.ReactionType* class method), 360
- `de_json()` (*telegram.ReplyParameters* class method), 370
- `de_json()` (*telegram.SecureData* class method), 514
- `de_json()` (*telegram.SecureValue* class method), 516
- `de_json()` (*telegram.ShippingQuery* class method), 489
- `de_json()` (*telegram.Sticker* class method), 420
- `de_json()` (*telegram.StickerSet* class method), 422
- `de_json()` (*telegram.SuccessfulPayment* class method), 490
- `de_json()` (*telegram.TelegramObject* class method), 374
- `de_json()` (*telegram.TextQuote* class method), 376
- `de_json()` (*telegram.Update* class method), 382
- `de_json()` (*telegram.UserChatBoosts* class method), 402
- `de_json()` (*telegram.UserProfilePhotos* class method), 403
- `de_json()` (*telegram.Venue* class method), 405
- `de_json()` (*telegram.Video* class method), 407
- `de_json()` (*telegram.VideoChatParticipantsInvited* class method), 408
- `de_json()` (*telegram.VideoChatScheduled* class method), 409
- `de_json()` (*telegram.VideoNote* class method), 410
- `de_json()` (*telegram.WebhookInfo* class method), 415
- `de_json_decrypted()` (*telegram.EncryptedPassportElement* class method), 498
- `de_json_decrypted()` (*telegram.PassportFile* class method), 509
- `de_list()` (*telegram.TelegramObject* class method), 375
- `de_list_decrypted()` (*telegram.PassportFile* class method), 509
- `decline()` (*telegram.ChatJoinRequest* method), 215
- `decline_chat_join_request()` (*telegram.Bot* method), 47
- `decline_join_request()` (*telegram.Chat* method), 183
- `decline_join_request()` (*telegram.User* method), 387
- `declineChatJoinRequest()` (*telegram.Bot* method), 46
- `decrypted_credentials` (*telegram.PassportData* property), 500
- `decrypted_data` (*telegram.EncryptedCredentials* property), 495
- `decrypted_data` (*telegram.PassportData* property), 500
- `decrypted_secret` (*telegram.EncryptedCredentials* property), 495
- `DEEP_LINK_LENGTH` (*telegram.constants.MessageLimit* attribute), 842
- `DEFAULT` (*telegram.BotCommandScope* attribute), 160
- `DEFAULT` (*telegram.constants.BotCommandScopeType* attribute), 658
- `DEFAULT` (*telegram.constants.MenuButtonType* attribute), 821
- `DEFAULT` (*telegram.MenuButton* attribute), 290
- `DEFAULT_NONE` (*telegram.request.BaseRequest* attribute), 950
- `DEFAULT_TYPE` (*telegram.ext.ContextTypes* attribute), 553
- `Defaults` (class in *telegram.ext*), 553
- `defaults` (*telegram.ext.ExtBot* property), 558
- `defaults()` (*telegram.ext.ApplicationBuilder* method), 534
- `delete()` (*telegram.Message* method), 314
- `DELETE_CHAT_PHOTO` (*telegram.constants.MessageType* attribute),

- 853
- DELETE_CHAT_PHOTO (telegram.ext.filters.StatusUpdate attribute), 607
- delete_chat_photo (telegram.Message attribute), 304
- delete_chat_photo() (telegram.Bot method), 48
- delete_chat_sticker_set() (telegram.Bot method), 49
- delete_forum_topic() (telegram.Bot method), 49
- delete_forum_topic() (telegram.Chat method), 184
- delete_forum_topic() (telegram.Message method), 314
- delete_message() (telegram.Bot method), 50
- delete_message() (telegram.CallbackQuery method), 168
- delete_message() (telegram.Chat method), 184
- delete_message() (telegram.User method), 388
- delete_messages() (telegram.Bot method), 51
- delete_messages() (telegram.Chat method), 184
- delete_messages() (telegram.User method), 388
- delete_my_commands() (telegram.Bot method), 51
- delete_photo() (telegram.Chat method), 185
- delete_sticker_from_set() (telegram.Bot method), 52
- delete_sticker_set() (telegram.Bot method), 53
- delete_webhook() (telegram.Bot method), 53
- deleteChatPhoto() (telegram.Bot method), 47
- deleteChatStickerSet() (telegram.Bot method), 47
- deleteForumTopic() (telegram.Bot method), 47
- deleteMessage() (telegram.Bot method), 48
- deleteMessages() (telegram.Bot method), 48
- deleteMyCommands() (telegram.Bot method), 48
- deleteStickerFromSet() (telegram.Bot method), 48
- deleteStickerSet() (telegram.Bot method), 48
- deleteWebhook() (telegram.Bot method), 48
- denominator (telegram.constants.BotCommandLimit attribute), 656
- denominator (telegram.constants.BotDescriptionLimit attribute), 667
- denominator (telegram.constants.BotNameLimit attribute), 671
- denominator (telegram.constants.BulkRequestLimit attribute), 675
- denominator (telegram.constants.CallbackQueryLimit attribute), 679
- denominator (telegram.constants.ChatID attribute), 696
- denominator (telegram.constants.ChatInviteLinkLimit attribute), 700
- denominator (telegram.constants.ChatLimit attribute), 705
- denominator (telegram.constants.ChatPhotoSize attribute), 715
- denominator (telegram.constants.ContactLimit attribute), 725
- denominator (telegram.constants.CustomEmojiStickerLimit attribute), 729
- denominator (telegram.constants.DiceLimit attribute), 740
- denominator (telegram.constants.FileSizeLimit attribute), 744
- denominator (telegram.constants.FloodLimit attribute), 748
- denominator (telegram.constants.ForumIconColor attribute), 753
- denominator (telegram.constants.ForumTopicLimit attribute), 757
- denominator (telegram.constants.GiveawayLimit attribute), 761
- denominator (telegram.constants.InlineKeyboardButtonLimit attribute), 765
- denominator (telegram.constants.InlineKeyboardMarkupLimit attribute), 770
- denominator (telegram.constants.InlineQueryLimit attribute), 774
- denominator (telegram.constants.InlineQueryResultLimit attribute), 778
- denominator (telegram.constants.InlineQueryResultsButtonLimit attribute), 789
- denominator (telegram.constants.InvoiceLimit attribute), 801
- denominator (telegram.constants.KeyboardButtonRequestUsersLimit attribute), 805
- denominator (telegram.constants.LocationLimit attribute), 810
- denominator (telegram.constants.MediaGroupLimit attribute), 820
- denominator (telegram.constants.MessageLimit attribute), 846
- denominator (telegram.constants.PollingLimit attribute), 883
- denominator (telegram.constants.PollLimit attribute), 873
- denominator (telegram.constants.ReplyLimit attribute), 907
- denominator (telegram.constants.StickerLimit attribute), 918
- denominator (telegram.constants.StickerSetLimit attribute), 922
- denominator (telegram.constants.UserProfilePhotosLimit attribute), 940
- denominator (telegram.constants.WebhookLimit attribute), 944
- description (telegram.BotCommand attribute), 159
- description (telegram.BotDescription attribute), 165
- description (telegram.Chat attribute), 177
- description (telegram.Game attribute), 491
- description (telegram.InlineQueryResultArticle attribute), 429
- description (telegram.InlineQueryResultCachedDocument attribute), 434
- description (telegram.InlineQueryResultCachedPhoto attribute), 439
- description (telegram.InlineQueryResultCachedVideo attribute), 442

- `description` (*telegram.InlineQueryResultDocument* attribute), 448
- `description` (*telegram.InlineQueryResultPhoto* attribute), 460
- `description` (*telegram.InlineQueryResultVideo* attribute), 467
- `description` (*telegram.InputInvoiceMessageContent* attribute), 479
- `description` (*telegram.Invoice* attribute), 481
- `Dice` (class in *telegram*), 236
- `Dice` (class in *telegram.ext.filters*), 597
- `DICE` (*telegram.constants.DiceEmoji* attribute), 730
- `DICE` (*telegram.constants.MessageAttachmentType* attribute), 828
- `DICE` (*telegram.constants.MessageType* attribute), 853
- `DICE` (*telegram.Dice* attribute), 237
- `DICE` (*telegram.ext.filters.Dice* attribute), 598
- `dice` (*telegram.ExternalReplyInfo* attribute), 242
- `dice` (*telegram.Message* attribute), 305
- `Dice.Basketball` (class in *telegram.ext.filters*), 597
- `Dice.Bowling` (class in *telegram.ext.filters*), 598
- `Dice.Darts` (class in *telegram.ext.filters*), 598
- `Dice.Dice` (class in *telegram.ext.filters*), 598
- `Dice.Football` (class in *telegram.ext.filters*), 598
- `Dice.SlotMachine` (class in *telegram.ext.filters*), 598
- `DiceEmoji` (class in *telegram.constants*), 730
- `DiceLimit` (class in *telegram.constants*), 736
- `DictPersistence` (class in *telegram.ext*), 635
- `difference()` (*telegram.ChatMemberUpdated* method), 229
- `disable_content_type_detection` (*telegram.InputMediaDocument* attribute), 273
- `disable_notification` (*telegram.ext.Defaults* property), 555
- `disable_web_page_preview` (*telegram.ext.Defaults* property), 555
- `disable_web_page_preview` (*telegram.InputTextMessageContent* property), 471
- `distance` (*telegram.ProximityAlertTriggered* attribute), 358
- `do_api_request()` (*telegram.Bot* method), 54
- `do_process_update()` (*telegram.ext.BaseUpdateProcessor* method), 547
- `do_process_update()` (*telegram.ext.SimpleUpdateProcessor* method), 569
- `do_quote` (*telegram.ext.Defaults* property), 555
- `do_request()` (*telegram.request.BaseRequest* method), 951
- `do_request()` (*telegram.request.HTTPXRequest* method), 956
- `DOC` (*telegram.ext.filters.Document* attribute), 600
- `Document` (class in *telegram*), 238
- `Document` (class in *telegram.ext.filters*), 598
- `DOCUMENT` (*telegram.constants.InlineQueryResultType* attribute), 779
- `DOCUMENT` (*telegram.constants.InputMediaType* attribute), 790
- `DOCUMENT` (*telegram.constants.MessageAttachmentType* attribute), 828
- `DOCUMENT` (*telegram.constants.MessageType* attribute), 853
- `document` (*telegram.ExternalReplyInfo* attribute), 241
- `document` (*telegram.Message* attribute), 302
- `Document.Category` (class in *telegram.ext.filters*), 599
- `Document.FileExtension` (class in *telegram.ext.filters*), 599
- `Document.MimeType` (class in *telegram.ext.filters*), 600
- `document_file_id` (*telegram.InlineQueryResultCachedDocument* attribute), 434
- `document_no` (*telegram.IdDocumentData* attribute), 499
- `document_url` (*telegram.InlineQueryResultDocument* attribute), 448
- `DOCX` (*telegram.ext.filters.Document* attribute), 600
- `DOVE_OF_PEACE` (*telegram.constants.ReactionEmoji* attribute), 886
- `download_as_bytearray()` (*telegram.File* method), 244
- `download_to_drive()` (*telegram.File* method), 245
- `download_to_memory()` (*telegram.File* method), 245
- `driver_license` (*telegram.SecureData* attribute), 513
- `drop_callback_data()` (*telegram.ext.CallbackContext* method), 550
- `drop_chat_data()` (*telegram.ext.Application* method), 522
- `drop_chat_data()` (*telegram.ext.BasePersistence* method), 631
- `drop_chat_data()` (*telegram.ext.DictPersistence* method), 637
- `drop_chat_data()` (*telegram.ext.PicklePersistence* method), 641
- `drop_data()` (*telegram.ext.CallbackDataCache* method), 645
- `drop_user_data()` (*telegram.ext.Application* method), 522
- `drop_user_data()` (*telegram.ext.BasePersistence* method), 631
- `drop_user_data()` (*telegram.ext.DictPersistence* method), 637
- `drop_user_data()` (*telegram.ext.PicklePersistence* method), 642
- `duration` (*telegram.Animation* attribute), 156
- `duration` (*telegram.Audio* attribute), 157
- `duration` (*telegram.InputMediaAnimation* attribute), 269
- `duration` (*telegram.InputMediaAudio* attribute), 271
- `duration` (*telegram.InputMediaVideo* attribute), 277
- `duration` (*telegram.Video* attribute), 406

`duration` (*telegram.VideoChatEnded* attribute), 407
`duration` (*telegram.VideoNote* attribute), 410
`duration` (*telegram.Voice* attribute), 411

E

`edit_caption()` (*telegram.Message* method), 314
`edit_chat_invite_link()` (*telegram.Bot* method), 55
`edit_date` (*telegram.Message* attribute), 300
`edit_forum_topic()` (*telegram.Bot* method), 56
`edit_forum_topic()` (*telegram.Chat* method), 185
`edit_forum_topic()` (*telegram.Message* method), 315
`edit_general_forum_topic()` (*telegram.Bot* method), 57
`edit_general_forum_topic()` (*telegram.Chat* method), 185
`edit_invite_link()` (*telegram.Chat* method), 185
`edit_live_location()` (*telegram.Message* method), 315
`edit_media()` (*telegram.Message* method), 316
`edit_message_caption()` (*telegram.Bot* method), 58
`edit_message_caption()` (*telegram.CallbackQuery* method), 168
`edit_message_live_location()` (*telegram.Bot* method), 59
`edit_message_live_location()` (*telegram.CallbackQuery* method), 169
`edit_message_media()` (*telegram.Bot* method), 60
`edit_message_media()` (*telegram.CallbackQuery* method), 169
`edit_message_reply_markup()` (*telegram.Bot* method), 61
`edit_message_reply_markup()` (*telegram.CallbackQuery* method), 170
`edit_message_text()` (*telegram.Bot* method), 62
`edit_message_text()` (*telegram.CallbackQuery* method), 170
`edit_reply_markup()` (*telegram.Message* method), 316
`edit_text()` (*telegram.Message* method), 316
`editChatInviteLink()` (*telegram.Bot* method), 54
`EDITED` (*telegram.ext.filters.UpdateType* attribute), 611
`EDITED_CHANNEL_POST` (*telegram.constants.UpdateType* attribute), 930
`EDITED_CHANNEL_POST` (*telegram.ext.filters.UpdateType* attribute), 612
`EDITED_CHANNEL_POST` (*telegram.Update* attribute), 381
`edited_channel_post` (*telegram.Update* attribute), 379
`EDITED_MESSAGE` (*telegram.constants.UpdateType* attribute), 930
`EDITED_MESSAGE` (*telegram.ext.filters.UpdateType* attribute), 612
`EDITED_MESSAGE` (*telegram.Update* attribute), 382

`edited_message` (*telegram.Update* attribute), 379
`editForumTopic()` (*telegram.Bot* method), 54
`editGeneralForumTopic()` (*telegram.Bot* method), 55
`editMessageCaption()` (*telegram.Bot* method), 55
`editMessageLiveLocation()` (*telegram.Bot* method), 55
`editMessageMedia()` (*telegram.Bot* method), 55
`editMessageReplyMarkup()` (*telegram.Bot* method), 55
`editMessageText()` (*telegram.Bot* method), 55
`effective_attachment` (*telegram.Message* property), 317
`effective_chat` (*telegram.Update* property), 382
`effective_message` (*telegram.Update* property), 383
`effective_message_type()` (in module *telegram.helpers*), 949
`effective_name` (*telegram.Chat* property), 186
`effective_user` (*telegram.Update* property), 383
`element_hash` (*telegram.PassportElementErrorUnspecified* attribute), 508
`EMAIL` (*telegram.constants.MessageEntityType* attribute), 835
`email` (*telegram.EncryptedPassportElement* attribute), 497
`EMAIL` (*telegram.MessageEntity* attribute), 343
`email` (*telegram.OrderInfo* attribute), 484
`EMOJI` (*telegram.constants.ReactionType* attribute), 898
`emoji` (*telegram.Dice* attribute), 237
`EMOJI` (*telegram.ReactionType* attribute), 360
`emoji` (*telegram.ReactionTypeEmoji* attribute), 361
`emoji` (*telegram.Sticker* attribute), 419
`emoji_list` (*telegram.InputSticker* attribute), 278
`emoji_status_custom_emoji_id` (*telegram.Chat* attribute), 180
`emoji_status_expiration_date` (*telegram.Chat* attribute), 180
`enabled` (*telegram.ext.Job* property), 561
`encode()` (*telegram.constants.BotCommandScopeType* method), 659
`encode()` (*telegram.constants.ChatAction* method), 682
`encode()` (*telegram.constants.ChatBoostSources* method), 688
`encode()` (*telegram.constants.ChatMemberStatus* method), 707
`encode()` (*telegram.constants.ChatType* method), 717
`encode()` (*telegram.constants.DiceEmoji* method), 732
`encode()` (*telegram.constants.InlineQueryResultType* method), 782
`encode()` (*telegram.constants.InputMediaType* method), 792
`encode()` (*telegram.constants.MaskPosition* method), 813
`encode()` (*telegram.constants.MenuButtonType* method), 823
`encode()` (*telegram.constants.MessageAttachmentType*

- method*), 830
- `encode()` (*telegram.constants.MessageEntityType method*), 838
- `encode()` (*telegram.constants.MessageOriginType method*), 848
- `encode()` (*telegram.constants.MessageType method*), 859
- `encode()` (*telegram.constants.ParseMode method*), 865
- `encode()` (*telegram.constants.PollType method*), 876
- `encode()` (*telegram.constants.ReactionEmoji method*), 894
- `encode()` (*telegram.constants.ReactionType method*), 899
- `encode()` (*telegram.constants.StickerFormat method*), 910
- `encode()` (*telegram.constants.StickerType method*), 925
- `encode()` (*telegram.constants.UpdateType method*), 932
- `EncryptedCredentials` (*class in telegram*), 494
- `EncryptedPassportElement` (*class in telegram*), 496
- `END` (*telegram.ext.ConversationHandler attribute*), 587
- `EndPointNotFound`, 946
- `endswith()` (*telegram.constants.BotCommandScopeType expandtabs() method*), 659
- `endswith()` (*telegram.constants.ChatAction method*), 682
- `endswith()` (*telegram.constants.ChatBoostSources method*), 688
- `endswith()` (*telegram.constants.ChatMemberStatus method*), 707
- `endswith()` (*telegram.constants.ChatType method*), 718
- `endswith()` (*telegram.constants.DiceEmoji method*), 732
- `endswith()` (*telegram.constants.InlineQueryResultType method*), 782
- `endswith()` (*telegram.constants.InputMediaType method*), 792
- `endswith()` (*telegram.constants.MaskPosition method*), 813
- `endswith()` (*telegram.constants.MenuButtonType method*), 823
- `endswith()` (*telegram.constants.MessageAttachmentType expandtabs() method*), 830
- `endswith()` (*telegram.constants.MessageEntityType method*), 838
- `endswith()` (*telegram.constants.MessageOriginType method*), 849
- `endswith()` (*telegram.constants.MessageType method*), 859
- `endswith()` (*telegram.constants.ParseMode method*), 865
- `endswith()` (*telegram.constants.PollType method*), 876
- `endswith()` (*telegram.constants.ReactionEmoji method*), 894
- `endswith()` (*telegram.constants.ReactionType method*), 900
- `endswith()` (*telegram.constants.StickerFormat method*), 910
- `endswith()` (*telegram.constants.StickerType method*), 925
- `endswith()` (*telegram.constants.UpdateType method*), 933
- `entities` (*telegram.InputTextMessageContent attribute*), 471
- `entities` (*telegram.Message attribute*), 301
- `entities` (*telegram.TextQuote attribute*), 376
- `Entity` (*class in telegram.ext.filters*), 601
- `entry_points` (*telegram.ext.ConversationHandler property*), 588
- `error` (*telegram.ext.CallbackContext attribute*), 549
- `error_handlers` (*telegram.ext.Application attribute*), 519
- `escape_markdown()` (*in module telegram.helpers*), 949
- `EXE` (*telegram.ext.filters.Document attribute*), 600
- `expandtabs()` (*telegram.constants.BotCommandScopeType method*), 659
- `expandtabs()` (*telegram.constants.ChatAction method*), 682
- `expandtabs()` (*telegram.constants.ChatBoostSources method*), 688
- `expandtabs()` (*telegram.constants.ChatMemberStatus method*), 708
- `expandtabs()` (*telegram.constants.ChatType method*), 718
- `expandtabs()` (*telegram.constants.DiceEmoji method*), 732
- `expandtabs()` (*telegram.constants.InlineQueryResultType method*), 782
- `expandtabs()` (*telegram.constants.InputMediaType method*), 792
- `expandtabs()` (*telegram.constants.MaskPosition method*), 813
- `expandtabs()` (*telegram.constants.MenuButtonType method*), 823
- `expandtabs()` (*telegram.constants.MessageAttachmentType expandtabs() method*), 831
- `expandtabs()` (*telegram.constants.MessageEntityType method*), 838
- `expandtabs()` (*telegram.constants.MessageOriginType method*), 849
- `expandtabs()` (*telegram.constants.MessageType method*), 859
- `expandtabs()` (*telegram.constants.ParseMode method*), 865
- `expandtabs()` (*telegram.constants.PollType method*),

- 876
- `expandtabs()` (*telegram.constants.ReactionEmoji* attribute), 894
- `expandtabs()` (*telegram.constants.ReactionType* attribute), 900
- `expandtabs()` (*telegram.constants.StickerFormat* attribute), 910
- `expandtabs()` (*telegram.constants.StickerType* attribute), 925
- `expandtabs()` (*telegram.constants.UpdateType* attribute), 933
- `expiration_date` (*telegram.ChatBoost* attribute), 206
- `expire_date` (*telegram.ChatInviteLink* attribute), 213
- `expiry_date` (*telegram.IdDocumentData* attribute), 499
- `explanation` (*telegram.Poll* attribute), 353
- `explanation_entities` (*telegram.Poll* attribute), 354
- `explanation_parse_mode` (*telegram.ext.Defaults* property), 556
- `export_chat_invite_link()` (*telegram.Bot* method), 63
- `export_invite_link()` (*telegram.Chat* method), 186
- `exportChatInviteLink()` (*telegram.Bot* method), 63
- `ExtBot` (class in *telegram.ext*), 557
- `external_reply` (*telegram.Message* attribute), 308
- `ExternalReplyInfo` (class in *telegram*), 240
- `extract_uuids()` (*telegram.ext.CallbackDataCache* static method), 645
- `EYES` (*telegram.constants.MaskPosition* attribute), 811
- `EYES` (*telegram.constants.ReactionEmoji* attribute), 886
- `EYES` (*telegram.MaskPosition* attribute), 417
- ## F
- `FACE_SCREAMING_IN_FEAR` (*telegram.constants.ReactionEmoji* attribute), 887
- `FACE_THROWING_A_KISS` (*telegram.constants.ReactionEmoji* attribute), 887
- `FACE_WITH_ONE_EYEBROW_RAISED` (*telegram.constants.ReactionEmoji* attribute), 887
- `FACE_WITH_OPEN_MOUTH_VOMITING` (*telegram.constants.ReactionEmoji* attribute), 887
- `FACE_WITH_UNEVEN_EYES_AND_WAVY_MOUTH` (*telegram.constants.ReactionEmoji* attribute), 887
- `FAKE_CHANNEL` (*telegram.constants.ChatID* attribute), 692
- `fallbacks` (*telegram.ext.ConversationHandler* property), 588
- `FATHER_CHRISTMAS` (*telegram.constants.ReactionEmoji* attribute), 887
- `FEARFUL_FACE` (*telegram.constants.ReactionEmoji* attribute), 887
- `field_name` (*telegram.PassportElementErrorDataField* attribute), 502
- `field_tuple` (*telegram.InputFile* property), 265
- `File` (class in *telegram*), 243
- `file_date` (*telegram.PassportFile* property), 509
- `file_hash` (*telegram.PassportElementErrorFile* attribute), 502
- `file_hash` (*telegram.PassportElementErrorFrontSide* attribute), 504
- `file_hash` (*telegram.PassportElementErrorReverseSide* attribute), 505
- `file_hash` (*telegram.PassportElementErrorSelfie* attribute), 505
- `file_hash` (*telegram.PassportElementErrorTranslationFile* attribute), 506
- `file_hashes` (*telegram.PassportElementErrorFiles* property), 503
- `file_hashes` (*telegram.PassportElementErrorTranslationFiles* property), 507
- `file_id` (*telegram.Animation* attribute), 155
- `file_id` (*telegram.Audio* attribute), 157
- `file_id` (*telegram.Document* attribute), 238
- `file_id` (*telegram.File* attribute), 244
- `file_id` (*telegram.PassportFile* attribute), 508
- `file_id` (*telegram.PhotoSize* attribute), 351
- `file_id` (*telegram.Sticker* attribute), 419
- `file_id` (*telegram.Video* attribute), 406
- `file_id` (*telegram.VideoNote* attribute), 410
- `file_id` (*telegram.Voice* attribute), 411
- `file_name` (*telegram.Animation* attribute), 156
- `file_name` (*telegram.Audio* attribute), 158
- `file_name` (*telegram.Document* attribute), 239
- `file_name` (*telegram.Video* attribute), 406
- `file_path` (*telegram.File* attribute), 244
- `file_size` (*telegram.Animation* attribute), 156
- `file_size` (*telegram.Audio* attribute), 158
- `file_size` (*telegram.Document* attribute), 239
- `file_size` (*telegram.File* attribute), 244
- `file_size` (*telegram.PassportFile* attribute), 509
- `file_size` (*telegram.PhotoSize* attribute), 351
- `file_size` (*telegram.Sticker* attribute), 420
- `file_size` (*telegram.Video* attribute), 407
- `file_size` (*telegram.VideoNote* attribute), 410
- `file_size` (*telegram.Voice* attribute), 411
- `file_unique_id` (*telegram.Animation* attribute), 155
- `file_unique_id` (*telegram.Audio* attribute), 157
- `file_unique_id` (*telegram.Document* attribute), 239
- `file_unique_id` (*telegram.File* attribute), 244
- `file_unique_id` (*telegram.PassportFile* attribute), 508
- `file_unique_id` (*telegram.PhotoSize* attribute), 351
- `file_unique_id` (*telegram.Sticker* attribute), 419
- `file_unique_id` (*telegram.Video* attribute), 406
- `file_unique_id` (*telegram.VideoNote* attribute), 410
- `file_unique_id` (*telegram.Voice* attribute), 411
- `FileCredentials` (class in *telegram*), 498

- `filename` (*telegram.InputFile* attribute), 265
- `filepath` (*telegram.ext.PicklePersistence* attribute), 641
- `files` (*telegram.EncryptedPassportElement* attribute), 497
- `files` (*telegram.SecureValue* attribute), 516
- `FILESIZE_DOWNLOAD` (*telegram.constants.FileSizeLimit* attribute), 741
- `FILESIZE_DOWNLOAD_LOCAL_MODE` (*telegram.constants.FileSizeLimit* attribute), 741
- `FILESIZE_UPLOAD` (*telegram.constants.FileSizeLimit* attribute), 741
- `FILESIZE_UPLOAD_LOCAL_MODE` (*telegram.constants.FileSizeLimit* attribute), 741
- `FileSizeLimit` (class in *telegram.constants*), 741
- `filter()` (*telegram.ext.filters.MessageFilter* method), 604
- `filter()` (*telegram.ext.filters.UpdateFilter* method), 611
- `filters` (*telegram.ext.CommandHandler* attribute), 584
- `filters` (*telegram.ext.MessageHandler* attribute), 617
- `filters` (*telegram.ext.PrefixHandler* attribute), 624
- `find()` (*telegram.constants.BotCommandScopeType* method), 659
- `find()` (*telegram.constants.ChatAction* method), 682
- `find()` (*telegram.constants.ChatBoostSources* method), 688
- `find()` (*telegram.constants.ChatMemberStatus* method), 708
- `find()` (*telegram.constants.ChatType* method), 718
- `find()` (*telegram.constants.DiceEmoji* method), 732
- `find()` (*telegram.constants.InlineQueryResultType* method), 782
- `find()` (*telegram.constants.InputMediaType* method), 792
- `find()` (*telegram.constants.MaskPosition* method), 813
- `find()` (*telegram.constants.MenuButtonType* method), 823
- `find()` (*telegram.constants.MessageAttachmentType* method), 831
- `find()` (*telegram.constants.MessageEntityType* method), 838
- `find()` (*telegram.constants.MessageOriginType* method), 849
- `find()` (*telegram.constants.MessageType* method), 859
- `find()` (*telegram.constants.ParseMode* method), 865
- `find()` (*telegram.constants.PollType* method), 876
- `find()` (*telegram.constants.ReactionEmoji* method), 894
- `find()` (*telegram.constants.ReactionType* method), 900
- `find()` (*telegram.constants.StickerFormat* method), 910
- `find()` (*telegram.constants.StickerType* method), 925
- `find()` (*telegram.constants.UpdateType* method), 933
- `FIND_LOCATION` (*telegram.constants.ChatAction* attribute), 680
- `FIRE` (*telegram.constants.ReactionEmoji* attribute), 887
- `first_name` (*telegram.Bot* property), 64
- `first_name` (*telegram.Chat* attribute), 176
- `first_name` (*telegram.Contact* attribute), 235
- `first_name` (*telegram.InlineQueryResultContact* attribute), 446
- `first_name` (*telegram.InputContactMessageContent* attribute), 476
- `first_name` (*telegram.PersonalDetails* attribute), 510
- `first_name` (*telegram.User* attribute), 385
- `first_name_native` (*telegram.PersonalDetails* attribute), 511
- `FloodLimit` (class in *telegram.constants*), 745
- `flush()` (*telegram.ext.BasePersistence* method), 632
- `flush()` (*telegram.ext.DictPersistence* method), 637
- `flush()` (*telegram.ext.PicklePersistence* method), 642
- `FOOTBALL` (*telegram.constants.DiceEmoji* attribute), 730
- `FOOTBALL` (*telegram.Dice* attribute), 237
- `FOOTBALL` (*telegram.ext.filters.Dice* attribute), 598
- `Forbidden`, 946
- `force_reply` (*telegram.ForceReply* attribute), 247
- `ForceReply` (class in *telegram*), 246
- `FOREHEAD` (*telegram.constants.MaskPosition* attribute), 812
- `FOREHEAD` (*telegram.MaskPosition* attribute), 417
- `format()` (*telegram.constants.BotCommandScopeType* method), 660
- `format()` (*telegram.constants.ChatAction* method), 682
- `format()` (*telegram.constants.ChatBoostSources* method), 688
- `format()` (*telegram.constants.ChatMemberStatus* method), 708
- `format()` (*telegram.constants.ChatType* method), 718
- `format()` (*telegram.constants.DiceEmoji* method), 732
- `format()` (*telegram.constants.InlineQueryResultType* method), 782
- `format()` (*telegram.constants.InputMediaType* method), 792
- `format()` (*telegram.constants.MaskPosition* method), 813
- `format()` (*telegram.constants.MenuButtonType* method), 823
- `format()` (*telegram.constants.MessageAttachmentType* method), 831
- `format()` (*telegram.constants.MessageEntityType* method), 838
- `format()` (*telegram.constants.MessageOriginType* method), 849
- `format()` (*telegram.constants.MessageType* method), 859

- `format()` (*telegram.constants.ParseMode* method), 865
- `format()` (*telegram.constants.PollType* method), 876
- `format()` (*telegram.constants.ReactionEmoji* method), 894
- `format()` (*telegram.constants.ReactionType* method), 900
- `format()` (*telegram.constants.StickerFormat* method), 910
- `format()` (*telegram.constants.StickerType* method), 925
- `format()` (*telegram.constants.UpdateType* method), 933
- `format_map()` (*telegram.constants.BotCommandScopeType* method), 660
- `format_map()` (*telegram.constants.ChatAction* method), 682
- `format_map()` (*telegram.constants.ChatBoostSources* method), 688
- `format_map()` (*telegram.constants.ChatMemberStatus* method), 708
- `format_map()` (*telegram.constants.ChatType* method), 718
- `format_map()` (*telegram.constants.DiceEmoji* method), 732
- `format_map()` (*telegram.constants.InlineQueryResultType* method), 782
- `format_map()` (*telegram.constants.InputMediaType* method), 792
- `format_map()` (*telegram.constants.MaskPosition* method), 813
- `format_map()` (*telegram.constants.MenuButtonType* method), 823
- `format_map()` (*telegram.constants.MessageAttachmentType* method), 831
- `format_map()` (*telegram.constants.MessageEntityType* method), 838
- `format_map()` (*telegram.constants.MessageOriginType* method), 849
- `format_map()` (*telegram.constants.MessageType* method), 859
- `format_map()` (*telegram.constants.ParseMode* method), 865
- `format_map()` (*telegram.constants.PollType* method), 876
- `format_map()` (*telegram.constants.ReactionEmoji* method), 894
- `format_map()` (*telegram.constants.ReactionType* method), 900
- `format_map()` (*telegram.constants.StickerFormat* method), 910
- `format_map()` (*telegram.constants.StickerType* method), 925
- `format_map()` (*telegram.constants.UpdateType* method), 933
- `FORUM_TOPIC_CLOSED` (*telegram.constants.MessageType* attribute), 853
- `FORUM_TOPIC_CLOSED` (*telegram.ext.filters.StatusUpdate* attribute), 607
- `forum_topic_closed` (*telegram.Message* attribute), 307
- `FORUM_TOPIC_CREATED` (*telegram.constants.MessageType* attribute), 854
- `FORUM_TOPIC_CREATED` (*telegram.ext.filters.StatusUpdate* attribute), 607
- `forum_topic_created` (*telegram.Message* attribute), 307
- `FORUM_TOPIC_EDITED` (*telegram.constants.MessageType* attribute), 854
- `FORUM_TOPIC_EDITED` (*telegram.ext.filters.StatusUpdate* attribute), 607
- `forum_topic_edited` (*telegram.Message* attribute), 307
- `FORUM_TOPIC_REOPENED` (*telegram.constants.MessageType* attribute), 854
- `FORUM_TOPIC_REOPENED` (*telegram.ext.filters.StatusUpdate* attribute), 607
- `forum_topic_reopened` (*telegram.Message* attribute), 307
- `ForumIconColor` (class in *telegram.constants*), 749
- `ForumTopic` (class in *telegram*), 248
- `ForumTopicClosed` (class in *telegram*), 248
- `ForumTopicCreated` (class in *telegram*), 249
- `ForumTopicEdited` (class in *telegram*), 249
- `ForumTopicLimit` (class in *telegram.constants*), 754
- `ForumTopicReopened` (class in *telegram*), 250
- `forward()` (*telegram.Message* method), 318
- `forward_date` (*telegram.Message* property), 318
- `forward_from` (*telegram.Message* property), 318
- `forward_from()` (*telegram.Chat* method), 186
- `forward_from()` (*telegram.User* method), 388
- `forward_from_chat` (*telegram.Message* property), 318
- `forward_from_message_id` (*telegram.Message* property), 318
- `forward_message()` (*telegram.Bot* method), 64
- `forward_messages()` (*telegram.Bot* method), 66
- `forward_messages_from()` (*telegram.Chat* method), 186
- `forward_messages_from()` (*telegram.User* method), 389
- `forward_messages_to()` (*telegram.Chat* method), 186

- 187
- `forward_messages_to()` (*telegram.User* method), 389
- `forward_origin` (*telegram.Message* attribute), 308
- `forward_sender_name` (*telegram.Message* property), 319
- `forward_signature` (*telegram.Message* property), 319
- `forward_text` (*telegram.LoginUrl* attribute), 288
- `forward_to()` (*telegram.Chat* method), 187
- `forward_to()` (*telegram.User* method), 389
- FORWARDED (in module *telegram.ext.filters*), 590
- ForwardedFrom (class in *telegram.ext.filters*), 601
- `forwardMessage()` (*telegram.Bot* method), 64
- `forwardMessages()` (*telegram.Bot* method), 64
- `foursquare_id` (*telegram.InlineQueryResultVenue* attribute), 463
- `foursquare_id` (*telegram.InputVenueMessageContent* attribute), 475
- `foursquare_id` (*telegram.Venue* attribute), 405
- `foursquare_type` (*telegram.InlineQueryResultVenue* attribute), 463
- `foursquare_type` (*telegram.InputVenueMessageContent* attribute), 475
- `foursquare_type` (*telegram.Venue* attribute), 405
- `from_aps_job()` (*telegram.ext.Job* class method), 561
- `from_attachment_menu` (*telegram.WriteAccessAllowed* attribute), 416
- `from_button()` (*telegram.InlineKeyboardMarkup* class method), 263
- `from_button()` (*telegram.ReplyKeyboardMarkup* class method), 364
- `from_bytes()` (*telegram.constants.BotCommandLimit* method), 656
- `from_bytes()` (*telegram.constants.BotDescriptionLimit* method), 667
- `from_bytes()` (*telegram.constants.BotNameLimit* method), 671
- `from_bytes()` (*telegram.constants.BulkRequestLimit* method), 675
- `from_bytes()` (*telegram.constants.CallbackQueryLimit* method), 679
- `from_bytes()` (*telegram.constants.ChatID* method), 696
- `from_bytes()` (*telegram.constants.ChatInviteLinkLimit* method), 700
- `from_bytes()` (*telegram.constants.ChatLimit* method), 705
- `from_bytes()` (*telegram.constants.ChatPhotoSize* method), 715
- `from_bytes()` (*telegram.constants.ContactLimit* method), 725
- `from_bytes()` (*telegram.constants.CustomEmojiStickerLimit* method), 729
- `from_bytes()` (*telegram.constants.DiceLimit* method), 740
- `from_bytes()` (*telegram.constants.FileSizeLimit* method), 744
- `from_bytes()` (*telegram.constants.FloodLimit* method), 748
- `from_bytes()` (*telegram.constants.ForumIconColor* method), 753
- `from_bytes()` (*telegram.constants.ForumTopicLimit* method), 757
- `from_bytes()` (*telegram.constants.GiveawayLimit* method), 761
- `from_bytes()` (*telegram.constants.InlineKeyboardButtonLimit* method), 765
- `from_bytes()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 770
- `from_bytes()` (*telegram.constants.InlineQueryLimit* method), 774
- `from_bytes()` (*telegram.constants.InlineQueryResultLimit* method), 778
- `from_bytes()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 789
- `from_bytes()` (*telegram.constants.InvoiceLimit* method), 801
- `from_bytes()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 805
- `from_bytes()` (*telegram.constants.LocationLimit* method), 810
- `from_bytes()` (*telegram.constants.MediaGroupLimit* method), 821
- `from_bytes()` (*telegram.constants.MessageLimit* method), 846
- `from_bytes()` (*telegram.constants.PollingLimit* method), 883
- `from_bytes()` (*telegram.constants.PollLimit* method), 873
- `from_bytes()` (*telegram.constants.ReplyLimit* method), 907
- `from_bytes()` (*telegram.constants.StickerLimit* method), 918
- `from_bytes()` (*telegram.constants.StickerSetLimit* method), 923
- `from_bytes()` (*telegram.constants.UserProfilePhotosLimit* method), 940
- `from_bytes()` (*telegram.constants.WebhookLimit* method), 944
- `from_column()` (*telegram.InlineKeyboardMarkup* class method), 263
- `from_column()` (*telegram.ReplyKeyboardMarkup*

- class method*), 365
 - `from_error()` (*telegram.ext.CallbackContext class method*), 550
 - `from_job()` (*telegram.ext.CallbackContext class method*), 550
 - `from_request` (*telegram.WriteAccessAllowed attribute*), 416
 - `from_row()` (*telegram.InlineKeyboardMarkup class method*), 263
 - `from_row()` (*telegram.ReplyKeyboardMarkup class method*), 366
 - `from_update()` (*telegram.ext.CallbackContext class method*), 551
 - `from_user` (*telegram.CallbackQuery attribute*), 166
 - `from_user` (*telegram.ChatJoinRequest attribute*), 214
 - `from_user` (*telegram.ChatMemberUpdated attribute*), 228
 - `from_user` (*telegram.ChosenInlineResult attribute*), 423
 - `from_user` (*telegram.InlineQuery attribute*), 425
 - `from_user` (*telegram.Message attribute*), 300
 - `from_user` (*telegram.PreCheckoutQuery attribute*), 485
 - `from_user` (*telegram.ShippingQuery attribute*), 488
 - `front_side` (*telegram.EncryptedPassportElement attribute*), 497
 - `front_side` (*telegram.SecureValue attribute*), 515
 - `full_name` (*telegram.Chat property*), 187
 - `full_name` (*telegram.User property*), 390
- ## G
- `Game` (*class in telegram*), 491
 - `GAME` (*in module telegram.ext.filters*), 590
 - `GAME` (*telegram.constants.InlineQueryResultType attribute*), 780
 - `GAME` (*telegram.constants.MessageAttachmentType attribute*), 828
 - `GAME` (*telegram.constants.MessageType attribute*), 854
 - `game` (*telegram.ExternalReplyInfo attribute*), 242
 - `game` (*telegram.Message attribute*), 302
 - `game_short_name` (*telegram.CallbackQuery attribute*), 167
 - `game_short_name` (*telegram.InlineQueryResultGame attribute*), 449
 - `GameHighScore` (*class in telegram*), 493
 - `gender` (*telegram.PersonalDetails attribute*), 511
 - `GENERAL_FORUM_TOPIC_HIDDEN` (*telegram.constants.MessageType attribute*), 854
 - `GENERAL_FORUM_TOPIC_HIDDEN` (*telegram.ext.filters.StatusUpdate attribute*), 607
 - `general_forum_topic_hidden` (*telegram.Message attribute*), 307
 - `GENERAL_FORUM_TOPIC_UNHIDDEN` (*telegram.constants.MessageType attribute*), 854
 - `GENERAL_FORUM_TOPIC_UNHIDDEN` (*telegram.ext.filters.StatusUpdate attribute*), 607
 - `general_forum_topic_unhidden` (*telegram.Message attribute*), 307
 - `GeneralForumTopicHidden` (*class in telegram*), 250
 - `GeneralForumTopicUnhidden` (*class in telegram*), 251
 - `get_administrators()` (*telegram.Chat method*), 188
 - `get_big_file()` (*telegram.ChatPhoto method*), 234
 - `get_bot()` (*telegram.TelegramObject method*), 375
 - `get_bot_data()` (*telegram.ext.BasePersistence method*), 632
 - `get_bot_data()` (*telegram.ext.DictPersistence method*), 637
 - `get_bot_data()` (*telegram.ext.PicklePersistence method*), 642
 - `get_callback_data()` (*telegram.ext.BasePersistence method*), 632
 - `get_callback_data()` (*telegram.ext.DictPersistence method*), 637
 - `get_callback_data()` (*telegram.ext.PicklePersistence method*), 642
 - `get_chat()` (*telegram.Bot method*), 68
 - `get_chat_administrators()` (*telegram.Bot method*), 68
 - `get_chat_boosts()` (*telegram.User method*), 390
 - `get_chat_data()` (*telegram.ext.BasePersistence method*), 632
 - `get_chat_data()` (*telegram.ext.DictPersistence method*), 637
 - `get_chat_data()` (*telegram.ext.PicklePersistence method*), 642
 - `get_chat_member()` (*telegram.Bot method*), 69
 - `get_chat_member_count()` (*telegram.Bot method*), 70
 - `get_chat_menu_button()` (*telegram.Bot method*), 70
 - `get_conversations()` (*telegram.ext.BasePersistence method*), 632
 - `get_conversations()` (*telegram.ext.DictPersistence method*), 638
 - `get_conversations()` (*telegram.ext.PicklePersistence method*), 642
 - `get_custom_emoji_stickers()` (*telegram.Bot method*), 71
 - `get_file()` (*telegram.Animation method*), 156
 - `get_file()` (*telegram.Audio method*), 158
 - `get_file()` (*telegram.Bot method*), 72
 - `get_file()` (*telegram.Document method*), 239
 - `get_file()` (*telegram.PassportFile method*), 509
 - `get_file()` (*telegram.PhotoSize method*), 351
 - `get_file()` (*telegram.Sticker method*), 420
 - `get_file()` (*telegram.Video method*), 407
 - `get_file()` (*telegram.VideoNote method*), 410
 - `get_file()` (*telegram.Voice method*), 412
 - `get_forum_topic_icon_stickers()` (*telegram.Bot method*), 72
 - `get_game_high_scores()` (*telegram.Bot method*), 73

`get_game_high_scores()` (*telegram.CallbackQuery method*), 171

`get_game_high_scores()` (*telegram.Message method*), 319

`get_jobs_by_name()` (*telegram.ext.JobQueue method*), 563

`get_me()` (*telegram.Bot method*), 74

`get_member()` (*telegram.Chat method*), 188

`get_member_count()` (*telegram.Chat method*), 188

`get_menu_button()` (*telegram.Chat method*), 188

`get_menu_button()` (*telegram.User method*), 390

`get_my_commands()` (*telegram.Bot method*), 74

`get_my_default_administrator_rights()` (*telegram.Bot method*), 75

`get_my_description()` (*telegram.Bot method*), 75

`get_my_name()` (*telegram.Bot method*), 76

`get_my_short_description()` (*telegram.Bot method*), 76

`get_profile_photos()` (*telegram.User method*), 390

`get_small_file()` (*telegram.ChatPhoto method*), 234

`get_sticker_set()` (*telegram.Bot method*), 77

`get_updates()` (*telegram.Bot method*), 77

`get_updates_connect_timeout()` (*telegram.ext.ApplicationBuilder method*), 535

`get_updates_connection_pool_size()` (*telegram.ext.ApplicationBuilder method*), 535

`get_updates_http_version()` (*telegram.ext.ApplicationBuilder method*), 535

`get_updates_pool_timeout()` (*telegram.ext.ApplicationBuilder method*), 536

`get_updates_proxy()` (*telegram.ext.ApplicationBuilder method*), 536

`get_updates_proxy_url()` (*telegram.ext.ApplicationBuilder method*), 536

`get_updates_read_timeout()` (*telegram.ext.ApplicationBuilder method*), 537

`get_updates_request()` (*telegram.ext.ApplicationBuilder method*), 537

`get_updates_socket_options()` (*telegram.ext.ApplicationBuilder method*), 537

`get_updates_write_timeout()` (*telegram.ext.ApplicationBuilder method*), 537

`get_user_chat_boosts()` (*telegram.Bot method*), 79

`get_user_chat_boosts()` (*telegram.Chat method*), 189

`get_user_data()` (*telegram.ext.BasePersistence method*), 633

`get_user_data()` (*telegram.ext.DictPersistence method*), 638

`get_user_data()` (*telegram.ext.PicklePersistence method*), 642

`get_user_profile_photos()` (*telegram.Bot method*), 79

`get_webhook_info()` (*telegram.Bot method*), 80

`getChat()` (*telegram.Bot method*), 67

`getChatAdministrators()` (*telegram.Bot method*), 67

`getChatMember()` (*telegram.Bot method*), 67

`getChatMemberCount()` (*telegram.Bot method*), 67

`getChatMenuButton()` (*telegram.Bot method*), 67

`getCustomEmojiStickers()` (*telegram.Bot method*), 67

`getFile()` (*telegram.Bot method*), 67

`getForumTopicIconStickers()` (*telegram.Bot method*), 67

`getGameHighScores()` (*telegram.Bot method*), 67

`getMe()` (*telegram.Bot method*), 67

`getMyCommands()` (*telegram.Bot method*), 67

`getMyDefaultAdministratorRights()` (*telegram.Bot method*), 67

`getMyDescription()` (*telegram.Bot method*), 67

`getMyName()` (*telegram.Bot method*), 67

`getMyShortDescription()` (*telegram.Bot method*), 68

`getStickerSet()` (*telegram.Bot method*), 68

`getUpdates()` (*telegram.Bot method*), 68

`getUserChatBoosts()` (*telegram.Bot method*), 68

`getUserProfilePhotos()` (*telegram.Bot method*), 68

`getWebhookInfo()` (*telegram.Bot method*), 68

`GHOST` (*telegram.constants.ReactionEmoji attribute*), 887

`GIF` (*telegram.constants.InlineQueryResultType attribute*), 780

`GIF` (*telegram.ext.filters.Document attribute*), 600

`gif_duration` (*telegram.InlineQueryResultGif attribute*), 451

`gif_file_id` (*telegram.InlineQueryResultCachedGif attribute*), 436

`gif_height` (*telegram.InlineQueryResultGif attribute*), 451

`gif_url` (*telegram.InlineQueryResultGif attribute*), 451

`gif_width` (*telegram.InlineQueryResultGif attribute*), 451

`GIFT_CODE` (*telegram.ChatBoostSource attribute*), 208

`GIFT_CODE` (*telegram.constants.ChatBoostSources attribute*), 686

`Giveaway` (*class in telegram*), 251

`GIVEAWAY` (*in module telegram.ext.filters*), 590

`GIVEAWAY` (*telegram.ChatBoostSource attribute*), 208

`GIVEAWAY` (*telegram.constants.ChatBoostSources attribute*), 686

`GIVEAWAY` (*telegram.constants.MessageType attribute*), 854

`giveaway` (*telegram.ExternalReplyInfo attribute*), 242

`giveaway` (*telegram.Message attribute*), 308

- GIVEAWAY_COMPLETED (*telegram.constants.MessageType* attribute), 854
- GIVEAWAY_COMPLETED (*telegram.ext.filters.StatusUpdate* attribute), 608
- giveaway_completed (*telegram.Message* attribute), 308
- GIVEAWAY_CREATED (*telegram.constants.MessageType* attribute), 854
- GIVEAWAY_CREATED (*telegram.ext.filters.StatusUpdate* attribute), 608
- giveaway_created (*telegram.Message* attribute), 308
- giveaway_message (*telegram.GiveawayCompleted* attribute), 253
- giveaway_message_id (*telegram.ChatBoostSourceGiveaway* attribute), 210
- giveaway_message_id (*telegram.GiveawayWinners* attribute), 254
- GIVEAWAY_WINNERS (in module *telegram.ext.filters*), 590
- GIVEAWAY_WINNERS (*telegram.constants.MessageType* attribute), 855
- giveaway_winners (*telegram.ExternalReplyInfo* attribute), 242
- giveaway_winners (*telegram.Message* attribute), 308
- GiveawayCompleted (class in *telegram*), 253
- GiveawayCreated (class in *telegram*), 253
- GiveawayLimit (class in *telegram.constants*), 758
- GiveawayWinners (class in *telegram*), 254
- google_place_id (*telegram.InlineQueryResultVenue* attribute), 463
- google_place_id (*telegram.InputVenueMessageContent* attribute), 475
- google_place_id (*telegram.Venue* attribute), 405
- google_place_type (*telegram.InlineQueryResultVenue* attribute), 463
- google_place_type (*telegram.InputVenueMessageContent* attribute), 475
- google_place_type (*telegram.Venue* attribute), 405
- GREEN (*telegram.constants.ForumIconColor* attribute), 749
- GRINNING_FACE_WITH_ONE_LARGE_AND_ONE_SMALL_EYE (*telegram.constants.ReactionEmoji* attribute), 887
- GRINNING_FACE_WITH_SMILING_EYES (*telegram.constants.ReactionEmoji* attribute), 887
- GRINNING_FACE_WITH_STAR_EYES (*telegram.constants.ReactionEmoji* attribute), 888
- GROUP (*telegram.Chat* attribute), 180
- GROUP (*telegram.constants.ChatType* attribute), 716
- GROUP (*telegram.ext.filters.ChatType* attribute), 596
- GROUP_CHAT_CREATED (*telegram.constants.MessageType* attribute), 855
- group_chat_created (*telegram.Message* attribute), 304
- GROUPS (*telegram.ext.filters.ChatType* attribute), 596
- ## H
- handle_update() (*telegram.ext.BaseHandler* method), 575
- handle_update() (*telegram.ext.ConversationHandler* method), 588
- handlers (*telegram.ext.Application* attribute), 519
- HANDSHAKE (*telegram.constants.ReactionEmoji* attribute), 888
- has_aggressive_anti_spam_enabled (*telegram.Chat* attribute), 180
- has_args (*telegram.ext.CommandHandler* attribute), 584
- has_custom_certificate (*telegram.WebhookInfo* attribute), 414
- has_hidden_members (*telegram.Chat* attribute), 180
- HAS_MEDIA_SPOILER (in module *telegram.ext.filters*), 590
- has_media_spoiler (*telegram.ExternalReplyInfo* attribute), 242
- has_media_spoiler (*telegram.Message* attribute), 307
- has_private_forwards (*telegram.Chat* attribute), 177
- HAS_PROTECTED_CONTENT (in module *telegram.ext.filters*), 590
- has_protected_content (*telegram.Chat* attribute), 178
- has_protected_content (*telegram.Message* attribute), 301
- has_public_winners (*telegram.Giveaway* attribute), 252
- has_restricted_voice_and_video_messages (*telegram.Chat* attribute), 179
- has_spoiler (*telegram.InputMediaAnimation* attribute), 269
- has_spoiler (*telegram.InputMediaPhoto* attribute), 275
- has_spoiler (*telegram.InputMediaVideo* attribute), 277
- has_visible_history (*telegram.Chat* attribute), 178
- hash (*telegram.DataCredentials* attribute), 494
- hash (*telegram.EncryptedCredentials* attribute), 495
- hash (*telegram.EncryptedPassportElement* attribute), 497
- hash (*telegram.FileCredentials* attribute), 498
- HASHTAG (*telegram.constants.MessageEntityType* attribute), 835
- HASHTAG (*telegram.MessageEntity* attribute), 343
- heading (*telegram.InlineQueryResultLocation* attribute), 454

- `heading` (*telegram.InputLocationMessageContent* attribute), 472
- `heading` (*telegram.Location* attribute), 286
- `HEAR_NO_EVIL_MONKEY` (*telegram.constants.ReactionEmoji* attribute), 888
- `HEART_ON_FIRE` (*telegram.constants.ReactionEmoji* attribute), 888
- `HEART_WITH_ARROW` (*telegram.constants.ReactionEmoji* attribute), 888
- `height` (*telegram.Animation* attribute), 156
- `height` (*telegram.InputMediaAnimation* attribute), 269
- `height` (*telegram.InputMediaVideo* attribute), 277
- `height` (*telegram.PhotoSize* attribute), 351
- `height` (*telegram.Sticker* attribute), 419
- `height` (*telegram.Video* attribute), 406
- `HIDDEN_USER` (*telegram.constants.MessageOriginType* attribute), 847
- `HIDDEN_USER` (*telegram.MessageOrigin* attribute), 345
- `hide_general_forum_topic()` (*telegram.Bot* method), 80
- `hide_general_forum_topic()` (*telegram.Chat* method), 189
- `hide_url` (*telegram.InlineQueryResultArticle* attribute), 429
- `hideGeneralForumTopic()` (*telegram.Bot* method), 80
- `HIGH_VOLTAGE_SIGN` (*telegram.constants.ReactionEmoji* attribute), 888
- `HORIZONTAL_ACCURACY` (*telegram.constants.LocationLimit* attribute), 806
- `HORIZONTAL_ACCURACY` (*telegram.InlineQueryResultLocation* attribute), 455
- `horizontal_accuracy` (*telegram.InlineQueryResultLocation* attribute), 454
- `HORIZONTAL_ACCURACY` (*telegram.InputLocationMessageContent* attribute), 473
- `horizontal_accuracy` (*telegram.InputLocationMessageContent* attribute), 472
- `HORIZONTAL_ACCURACY` (*telegram.Location* attribute), 287
- `horizontal_accuracy` (*telegram.Location* attribute), 286
- `HOT_DOG` (*telegram.constants.ReactionEmoji* attribute), 888
- `HTML` (*telegram.constants.ParseMode* attribute), 863
- `http_version` (*telegram.request.HTTPXRequest* property), 956
- `http_version()` (*telegram.ext.ApplicationBuilder* method), 538
- `HTTPXRequest` (class in *telegram.request*), 955
- `HUGGING_FACE` (*telegram.constants.ReactionEmoji* attribute), 888
- `HUNDRED_POINTS_SYMBOL` (*telegram.constants.ReactionEmoji* attribute), 888
- I**
- `icon_color` (*telegram.ForumTopic* attribute), 248
- `icon_color` (*telegram.ForumTopicCreated* attribute), 249
- `icon_custom_emoji_id` (*telegram.ForumTopic* attribute), 248
- `icon_custom_emoji_id` (*telegram.ForumTopicCreated* attribute), 249
- `icon_custom_emoji_id` (*telegram.ForumTopicEdited* attribute), 250
- `id` (*telegram.Bot* property), 81
- `id` (*telegram.CallbackQuery* attribute), 166
- `id` (*telegram.Chat* attribute), 176
- `id` (*telegram.InlineQuery* attribute), 425
- `id` (*telegram.InlineQueryResult* attribute), 427
- `id` (*telegram.InlineQueryResultArticle* attribute), 428
- `id` (*telegram.InlineQueryResultAudio* attribute), 430
- `id` (*telegram.InlineQueryResultCachedAudio* attribute), 432
- `id` (*telegram.InlineQueryResultCachedDocument* attribute), 434
- `id` (*telegram.InlineQueryResultCachedGif* attribute), 435
- `id` (*telegram.InlineQueryResultCachedMpeg4Gif* attribute), 437
- `id` (*telegram.InlineQueryResultCachedPhoto* attribute), 439
- `id` (*telegram.InlineQueryResultCachedSticker* attribute), 441
- `id` (*telegram.InlineQueryResultCachedVideo* attribute), 442
- `id` (*telegram.InlineQueryResultCachedVoice* attribute), 444
- `id` (*telegram.InlineQueryResultContact* attribute), 445
- `id` (*telegram.InlineQueryResultDocument* attribute), 448
- `id` (*telegram.InlineQueryResultGame* attribute), 449
- `id` (*telegram.InlineQueryResultGif* attribute), 451
- `id` (*telegram.InlineQueryResultLocation* attribute), 453
- `id` (*telegram.InlineQueryResultMpeg4Gif* attribute), 457
- `id` (*telegram.InlineQueryResultPhoto* attribute), 459
- `id` (*telegram.InlineQueryResultVenue* attribute), 463
- `id` (*telegram.InlineQueryResultVideo* attribute), 466
- `id` (*telegram.InlineQueryResultVoice* attribute), 468
- `id` (*telegram.Message* property), 319
- `id` (*telegram.Poll* attribute), 353
- `id` (*telegram.PreCheckoutQuery* attribute), 485
- `id` (*telegram.ShippingOption* attribute), 487
- `id` (*telegram.ShippingQuery* attribute), 488
- `id` (*telegram.User* attribute), 385
- `IdDocumentData` (class in *telegram*), 499

- `identity_card` (*telegram.SecureData* attribute), 514
- `imag` (*telegram.constants.BotCommandLimit* attribute), 657
- `imag` (*telegram.constants.BotDescriptionLimit* attribute), 667
- `imag` (*telegram.constants.BotNameLimit* attribute), 671
- `imag` (*telegram.constants.BulkRequestLimit* attribute), 675
- `imag` (*telegram.constants.CallbackQueryLimit* attribute), 679
- `imag` (*telegram.constants.ChatID* attribute), 696
- `imag` (*telegram.constants.ChatInviteLinkLimit* attribute), 701
- `imag` (*telegram.constants.ChatLimit* attribute), 705
- `imag` (*telegram.constants.ChatPhotoSize* attribute), 715
- `imag` (*telegram.constants.ContactLimit* attribute), 725
- `imag` (*telegram.constants.CustomEmojiStickerLimit* attribute), 729
- `imag` (*telegram.constants.DiceLimit* attribute), 740
- `imag` (*telegram.constants.FileSizeLimit* attribute), 745
- `imag` (*telegram.constants.FloodLimit* attribute), 749
- `imag` (*telegram.constants.ForumIconColor* attribute), 753
- `imag` (*telegram.constants.ForumTopicLimit* attribute), 758
- `imag` (*telegram.constants.GiveawayLimit* attribute), 762
- `imag` (*telegram.constants.InlineKeyboardButtonLimit* attribute), 766
- `imag` (*telegram.constants.InlineKeyboardMarkupLimit* attribute), 770
- `imag` (*telegram.constants.InlineQueryLimit* attribute), 774
- `imag` (*telegram.constants.InlineQueryResultLimit* attribute), 779
- `imag` (*telegram.constants.InlineQueryResultsButtonLimit* attribute), 790
- `imag` (*telegram.constants.InvoiceLimit* attribute), 801
- `imag` (*telegram.constants.KeyboardButtonRequestUsersLimit* attribute), 805
- `imag` (*telegram.constants.LocationLimit* attribute), 811
- `imag` (*telegram.constants.MediaGroupLimit* attribute), 821
- `imag` (*telegram.constants.MessageLimit* attribute), 846
- `imag` (*telegram.constants.PollingLimit* attribute), 883
- `imag` (*telegram.constants.PollLimit* attribute), 874
- `imag` (*telegram.constants.ReplyLimit* attribute), 907
- `imag` (*telegram.constants.StickerLimit* attribute), 918
- `imag` (*telegram.constants.StickerSetLimit* attribute), 923
- `imag` (*telegram.constants.UserProfilePhotosLimit* attribute), 940
- `imag` (*telegram.constants.WebhookLimit* attribute), 944
- `IMAGE` (*telegram.ext.filters.Document* attribute), 599
- `InaccessibleMessage` (class in *telegram*), 256
- `index()` (*telegram.constants.BotCommandScopeType* method), 660
- `index()` (*telegram.constants.ChatAction* method), 682
- `index()` (*telegram.constants.ChatBoostSources* method), 688
- `index()` (*telegram.constants.ChatMemberStatus* method), 708
- `index()` (*telegram.constants.ChatType* method), 718
- `index()` (*telegram.constants.DiceEmoji* method), 732
- `index()` (*telegram.constants.InlineQueryResultType* method), 782
- `index()` (*telegram.constants.InputMediaType* method), 792
- `index()` (*telegram.constants.MaskPosition* method), 814
- `index()` (*telegram.constants.MenuButtonType* method), 823
- `index()` (*telegram.constants.MessageAttachmentType* method), 831
- `index()` (*telegram.constants.MessageEntityType* method), 838
- `index()` (*telegram.constants.MessageOriginType* method), 849
- `index()` (*telegram.constants.MessageType* method), 859
- `index()` (*telegram.constants.ParseMode* method), 865
- `index()` (*telegram.constants.PollType* method), 876
- `index()` (*telegram.constants.ReactionEmoji* method), 894
- `index()` (*telegram.constants.ReactionType* method), 900
- `index()` (*telegram.constants.StickerFormat* method), 910
- `index()` (*telegram.constants.StickerType* method), 925
- `index()` (*telegram.constants.UpdateType* method), 933
- `initialize()` (*telegram.Bot* method), 81
- `initialize()` (*telegram.ext.AIORateLimiter* method), 650
- `initialize()` (*telegram.ext.Application* method), 523
- `initialize()` (*telegram.ext.BaseRateLimiter* method), 647
- `initialize()` (*telegram.ext.BaseUpdateProcessor* method), 547
- `initialize()` (*telegram.ext.ExtBot* method), 558
- `initialize()` (*telegram.ext.SimpleUpdateProcessor* method), 569
- `initialize()` (*telegram.ext.Updater* method), 570
- `initialize()` (*telegram.request.BaseRequest* method), 952
- `initialize()` (*telegram.request.HTTPXRequest* method), 956
- `inline_keyboard` (*telegram.InlineKeyboardMarkup* attribute), 263
- `inline_message_id` (*telegram.CallbackQuery* attribute), 167
- `inline_message_id` (*telegram.ChosenInlineResult* attribute), 423
- `inline_message_id` (*telegram.SentWebAppMessage* attribute), 370
- `INLINE_QUERY` (*telegram.constants.UpdateType* attribute), 930

- `INLINE_QUERY` (*telegram.Update* attribute), 382
- `inline_query` (*telegram.Update* attribute), 379
- `InlineKeyboardButton` (class in *telegram*), 256
- `InlineKeyboardButtonLimit` (class in *telegram.constants*), 762
- `InlineKeyboardMarkup` (class in *telegram*), 260
- `InlineKeyboardMarkupLimit` (class in *telegram.constants*), 766
- `InlineQuery` (class in *telegram*), 424
- `InlineQueryHandler` (class in *telegram.ext*), 615
- `InlineQueryLimit` (class in *telegram.constants*), 770
- `InlineQueryResult` (class in *telegram*), 426
- `InlineQueryResultArticle` (class in *telegram*), 427
- `InlineQueryResultAudio` (class in *telegram*), 429
- `InlineQueryResultCachedAudio` (class in *telegram*), 431
- `InlineQueryResultCachedDocument` (class in *telegram*), 433
- `InlineQueryResultCachedGif` (class in *telegram*), 435
- `InlineQueryResultCachedMpeg4Gif` (class in *telegram*), 437
- `InlineQueryResultCachedPhoto` (class in *telegram*), 438
- `InlineQueryResultCachedSticker` (class in *telegram*), 440
- `InlineQueryResultCachedVideo` (class in *telegram*), 441
- `InlineQueryResultCachedVoice` (class in *telegram*), 443
- `InlineQueryResultContact` (class in *telegram*), 445
- `InlineQueryResultDocument` (class in *telegram*), 447
- `InlineQueryResultGame` (class in *telegram*), 449
- `InlineQueryResultGif` (class in *telegram*), 450
- `InlineQueryResultLimit` (class in *telegram.constants*), 775
- `InlineQueryResultLocation` (class in *telegram*), 453
- `InlineQueryResultMpeg4Gif` (class in *telegram*), 455
- `InlineQueryResultPhoto` (class in *telegram*), 458
- `InlineQueryResultsButton` (class in *telegram*), 461
- `InlineQueryResultsButtonLimit` (class in *telegram.constants*), 786
- `InlineQueryResultType` (class in *telegram.constants*), 779
- `InlineQueryResultVenue` (class in *telegram*), 462
- `InlineQueryResultVideo` (class in *telegram*), 464
- `InlineQueryResultVoice` (class in *telegram*), 467
- `input_field_placeholder` (*telegram.ForceReply* attribute), 247
- `input_field_placeholder` (*telegram.ReplyKeyboardMarkup* attribute), 364
- `input_file_content` (*telegram.InputFile* attribute), 265
- `input_message_content` (*telegram.InlineQueryResultArticle* attribute), 428
- `input_message_content` (*telegram.InlineQueryResultAudio* attribute), 431
- `input_message_content` (*telegram.InlineQueryResultCachedAudio* attribute), 433
- `input_message_content` (*telegram.InlineQueryResultCachedDocument* attribute), 434
- `input_message_content` (*telegram.InlineQueryResultCachedGif* attribute), 436
- `input_message_content` (*telegram.InlineQueryResultCachedMpeg4Gif* attribute), 438
- `input_message_content` (*telegram.InlineQueryResultCachedPhoto* attribute), 440
- `input_message_content` (*telegram.InlineQueryResultCachedSticker* attribute), 441
- `input_message_content` (*telegram.InlineQueryResultCachedVideo* attribute), 443
- `input_message_content` (*telegram.InlineQueryResultCachedVoice* attribute), 444
- `input_message_content` (*telegram.InlineQueryResultContact* attribute), 446
- `input_message_content` (*telegram.InlineQueryResultDocument* attribute), 449
- `input_message_content` (*telegram.InlineQueryResultGif* attribute), 452
- `input_message_content` (*telegram.InlineQueryResultLocation* attribute), 454
- `input_message_content` (*telegram.InlineQueryResultMpeg4Gif* attribute), 458
- `input_message_content` (*telegram.InlineQueryResultPhoto* attribute), 460
- `input_message_content` (*telegram.InlineQueryResultVenue* attribute), 464
- `input_message_content` (*telegram.InlineQueryResultVideo* attribute), 467
- `input_message_content` (*telegram.InlineQueryResultVoice* attribute), 469
- `InputContactMessageContent` (class in *telegram*), 476

- `InputFile` (class in telegram), 264
- `InputInvoiceMessageContent` (class in telegram), 477
- `InputLocationMessageContent` (class in telegram), 472
- `InputMedia` (class in telegram), 266
- `InputMediaAnimation` (class in telegram), 267
- `InputMediaAudio` (class in telegram), 269
- `InputMediaDocument` (class in telegram), 271
- `InputMediaPhoto` (class in telegram), 273
- `InputMediaType` (class in telegram.constants), 790
- `InputMediaVideo` (class in telegram), 275
- `InputMessageContent` (class in telegram), 469
- `InputSticker` (class in telegram), 277
- `InputTextMessageContent` (class in telegram), 470
- `InputVenueMessageContent` (class in telegram), 474
- `insert_callback_data()` (telegram.ext.ExtBot method), 558
- `internal_passport` (telegram.SecureData attribute), 513
- `InvalidCallbackData` (class in telegram.ext), 647
- `InvalidToken`, 946
- `invite_link` (telegram.Chat attribute), 177
- `invite_link` (telegram.ChatInviteLink attribute), 212
- `invite_link` (telegram.ChatJoinRequest attribute), 215
- `invite_link` (telegram.ChatMemberUpdated attribute), 229
- `Invoice` (class in telegram), 481
- `INVOICE` (in module telegram.ext.filters), 591
- `INVOICE` (telegram.constants.MessageAttachmentType attribute), 828
- `INVOICE` (telegram.constants.MessageType attribute), 855
- `invoice` (telegram.ExternalReplyInfo attribute), 243
- `invoice` (telegram.Message attribute), 305
- `invoice_payload` (telegram.PreCheckoutQuery attribute), 485
- `invoice_payload` (telegram.ShippingQuery attribute), 488
- `invoice_payload` (telegram.SuccessfulPayment attribute), 490
- `InvoiceLimit` (class in telegram.constants), 796
- `ip_address` (telegram.WebhookInfo attribute), 414
- `is_accessible` (telegram.MaybeInaccessibleMessage property), 289
- `is_animated` (telegram.Sticker attribute), 419
- `is_animated` (telegram.StickerSet attribute), 422
- `is_anonymous` (telegram.ChatAdministratorRights attribute), 204
- `is_anonymous` (telegram.ChatMemberAdministrator attribute), 220
- `is_anonymous` (telegram.ChatMemberOwner attribute), 224
- `is_anonymous` (telegram.Poll attribute), 353
- `IS_AUTOMATIC_FORWARD` (in module telegram.ext.filters), 591
- `is_automatic_forward` (telegram.Message attribute), 300
- `is_bot` (telegram.User attribute), 385
- `is_closed` (telegram.Poll attribute), 353
- `is_disabled` (telegram.LinkPreviewOptions attribute), 285
- `is_flexible` (telegram.InputInvoiceMessageContent attribute), 481
- `is_forum` (telegram.Chat attribute), 179
- `is_integer()` (telegram.constants.BotCommandLimit method), 657
- `is_integer()` (telegram.constants.BotDescriptionLimit method), 667
- `is_integer()` (telegram.constants.BotNameLimit method), 671
- `is_integer()` (telegram.constants.BulkRequestLimit method), 675
- `is_integer()` (telegram.constants.CallbackQueryLimit method), 679
- `is_integer()` (telegram.constants.ChatID method), 696
- `is_integer()` (telegram.constants.ChatInviteLinkLimit method), 701
- `is_integer()` (telegram.constants.ChatLimit method), 705
- `is_integer()` (telegram.constants.ChatPhotoSize method), 715
- `is_integer()` (telegram.constants.ContactLimit method), 725
- `is_integer()` (telegram.constants.CustomEmojiStickerLimit method), 729
- `is_integer()` (telegram.constants.DiceLimit method), 740
- `is_integer()` (telegram.constants.FileSizeLimit method), 745
- `is_integer()` (telegram.constants.FloodLimit method), 749
- `is_integer()` (telegram.constants.ForumIconColor method), 753
- `is_integer()` (telegram.constants.ForumTopicLimit method), 758
- `is_integer()` (telegram.constants.GiveawayLimit method), 762
- `is_integer()` (telegram.constants.InlineKeyboardButtonLimit method), 766
- `is_integer()` (telegram.constants.InlineKeyboardMarkupLimit method), 770
- `is_integer()` (telegram.constants.InlineQueryLimit method), 774
- `is_integer()` (telegram.constants.InlineQueryResultLimit method), 778

method), 779

`is_integer()` (telegram.constants.InlineQueryResultsButtonLimit method), 790

`is_integer()` (telegram.constants.InvoiceLimit method), 801

`is_integer()` (telegram.constants.KeyboardButtonRequestUsersLimit method), 805

`is_integer()` (telegram.constants.LocationLimit method), 811

`is_integer()` (telegram.constants.MediaGroupLimit method), 821

`is_integer()` (telegram.constants.MessageLimit method), 846

`is_integer()` (telegram.constants.PollingLimit method), 884

`is_integer()` (telegram.constants.PollLimit method), 874

`is_integer()` (telegram.constants.ReplyLimit method), 907

`is_integer()` (telegram.constants.StickerLimit method), 918

`is_integer()` (telegram.constants.StickerSetLimit method), 923

`is_integer()` (telegram.constants.UserProfilePhotosLimit method), 940

`is_integer()` (telegram.constants.WebhookLimit method), 945

`is_manual` (telegram.TextQuote attribute), 376

`is_member` (telegram.ChatMemberRestricted attribute), 226

`is_persistent` (telegram.ReplyKeyboardMarkup attribute), 364

`is_premium` (telegram.User attribute), 386

`is_primary` (telegram.ChatInviteLink attribute), 213

`is_revoked` (telegram.ChatInviteLink attribute), 213

`IS_TOPIC_MESSAGE` (in module telegram.ext.filters), 591

`is_topic_message` (telegram.Message attribute), 306

`is_unclaimed` (telegram.ChatBoostSourceGiveaway attribute), 210

`is_video` (telegram.Sticker attribute), 419

`is_video` (telegram.StickerSet attribute), 422

`isalnum()` (telegram.constants.BotCommandScopeType method), 660

`isalnum()` (telegram.constants.ChatAction method), 683

`isalnum()` (telegram.constants.ChatBoostSources method), 689

`isalnum()` (telegram.constants.ChatMemberStatus method), 708

`isalnum()` (telegram.constants.ChatType method), 718

`isalnum()` (telegram.constants.DiceEmoji method), 732

`isalnum()` (telegram.constants.InlineQueryResultType method), 782

`isalnum()` (telegram.constants.InputMediaType method), 793

`isalnum()` (telegram.constants.MaskPosition method), 814

`isalnum()` (telegram.constants.MenuButtonType method), 824

`isalnum()` (telegram.constants.MessageAttachmentType method), 831

`isalnum()` (telegram.constants.MessageEntityType method), 838

`isalnum()` (telegram.constants.MessageOriginType method), 849

`isalnum()` (telegram.constants.MessageType method), 860

`isalnum()` (telegram.constants.ParseMode method), 866

`isalpha()` (telegram.constants.BotCommandScopeType method), 660

`isalpha()` (telegram.constants.ChatAction method), 683

`isalpha()` (telegram.constants.ChatBoostSources method), 689

`isalpha()` (telegram.constants.ChatMemberStatus method), 708

`isalpha()` (telegram.constants.ChatType method), 718

`isalpha()` (telegram.constants.DiceEmoji method), 733

`isalpha()` (telegram.constants.InlineQueryResultType method), 783

`isalpha()` (telegram.constants.InputMediaType method), 793

`isalpha()` (telegram.constants.MaskPosition method), 814

`isalpha()` (telegram.constants.MenuButtonType method), 824

`isalpha()` (telegram.constants.MessageAttachmentType method), 831

`isalpha()` (telegram.constants.MessageEntityType method), 838

`isalpha()` (telegram.constants.MessageOriginType method), 849

`isalpha()` (telegram.constants.MessageType method), 860

`isalpha()` (telegram.constants.ParseMode method), 866

- `isalpha()` (*telegram.constants.PollType* method), 876
- `isalpha()` (*telegram.constants.ReactionEmoji* method), 894
- `isalpha()` (*telegram.constants.ReactionType* method), 900
- `isalpha()` (*telegram.constants.StickerFormat* method), 910
- `isalpha()` (*telegram.constants.StickerType* method), 926
- `isalpha()` (*telegram.constants.UpdateType* method), 933
- `isascii()` (*telegram.constants.BotCommandScopeType* method), 660
- `isascii()` (*telegram.constants.ChatAction* method), 683
- `isascii()` (*telegram.constants.ChatBoostSources* method), 689
- `isascii()` (*telegram.constants.ChatMemberStatus* method), 708
- `isascii()` (*telegram.constants.ChatType* method), 718
- `isascii()` (*telegram.constants.DiceEmoji* method), 733
- `isascii()` (*telegram.constants.InlineQueryResultType* method), 783
- `isascii()` (*telegram.constants.InputMediaType* method), 793
- `isascii()` (*telegram.constants.MaskPosition* method), 814
- `isascii()` (*telegram.constants.MenuButtonType* method), 824
- `isascii()` (*telegram.constants.MessageAttachmentType* method), 831
- `isascii()` (*telegram.constants.MessageEntityType* method), 838
- `isascii()` (*telegram.constants.MessageOriginType* method), 849
- `isascii()` (*telegram.constants.MessageType* method), 860
- `isascii()` (*telegram.constants.ParseMode* method), 866
- `isascii()` (*telegram.constants.PollType* method), 876
- `isascii()` (*telegram.constants.ReactionEmoji* method), 894
- `isascii()` (*telegram.constants.ReactionType* method), 900
- `isascii()` (*telegram.constants.StickerFormat* method), 910
- `isascii()` (*telegram.constants.StickerType* method), 926
- `isascii()` (*telegram.constants.UpdateType* method), 933
- `isdecimal()` (*telegram.constants.BotCommandScopeType* method), 660
- `isdecimal()` (*telegram.constants.ChatAction* method), 683
- `isdecimal()` (*telegram.constants.ChatBoostSources* method), 689
- `isdecimal()` (*telegram.constants.ChatMemberStatus* method), 708
- `isdecimal()` (*telegram.constants.ChatType* method), 718
- `isdecimal()` (*telegram.constants.DiceEmoji* method), 733
- `isdecimal()` (*telegram.constants.InlineQueryResultType* method), 783
- `isdecimal()` (*telegram.constants.InputMediaType* method), 793
- `isdecimal()` (*telegram.constants.MaskPosition* method), 814
- `isdecimal()` (*telegram.constants.MenuButtonType* method), 824
- `isdecimal()` (*telegram.constants.MessageAttachmentType* method), 831
- `isdecimal()` (*telegram.constants.ChatMemberStatus* method), 708
- `isdecimal()` (*telegram.constants.ChatType* method), 718
- `isdecimal()` (*telegram.constants.DiceEmoji* method), 733
- `isdecimal()` (*telegram.constants.InlineQueryResultType* method), 783
- `isdecimal()` (*telegram.constants.InputMediaType* method), 793
- `isdecimal()` (*telegram.constants.MaskPosition* method), 814
- `isdecimal()` (*telegram.constants.MenuButtonType* method), 824
- `isdecimal()` (*telegram.constants.MessageAttachmentType* method), 831
- `isdecimal()` (*telegram.constants.MessageEntityType* method), 839
- `isdecimal()` (*telegram.constants.MessageOriginType* method), 849
- `isdecimal()` (*telegram.constants.MessageType* method), 860
- `isdecimal()` (*telegram.constants.ParseMode* method), 866
- `isdecimal()` (*telegram.constants.PollType* method), 877
- `isdecimal()` (*telegram.constants.ReactionEmoji* method), 895
- `isdecimal()` (*telegram.constants.ReactionType* method), 900
- `isdecimal()` (*telegram.constants.StickerFormat* method), 910
- `isdecimal()` (*telegram.constants.StickerType* method), 926
- `isdecimal()` (*telegram.constants.UpdateType* method), 933
- `isdigit()` (*telegram.constants.BotCommandScopeType* method), 660
- `isdigit()` (*telegram.constants.ChatAction* method), 683
- `isdigit()` (*telegram.constants.ChatBoostSources* method), 689
- `isdigit()` (*telegram.constants.ChatMemberStatus* method), 708
- `isdigit()` (*telegram.constants.ChatType* method), 718
- `isdigit()` (*telegram.constants.DiceEmoji* method), 733
- `isdigit()` (*telegram.constants.InlineQueryResultType* method), 783
- `isdigit()` (*telegram.constants.InputMediaType* method), 793
- `isdigit()` (*telegram.constants.MaskPosition* method), 814
- `isdigit()` (*telegram.constants.MenuButtonType* method), 824
- `isdigit()` (*telegram.constants.MessageAttachmentType* method), 831

- `isdigit()` (*telegram.constants.MessageEntityType method*), 839
- `isdigit()` (*telegram.constants.MessageOriginType method*), 849
- `isdigit()` (*telegram.constants.MessageType method*), 860
- `isdigit()` (*telegram.constants.ParseMode method*), 866
- `isdigit()` (*telegram.constants.PollType method*), 877
- `isdigit()` (*telegram.constants.ReactionEmoji method*), 895
- `isdigit()` (*telegram.constants.ReactionType method*), 900
- `isdigit()` (*telegram.constants.StickerFormat method*), 911
- `isdigit()` (*telegram.constants.StickerType method*), 926
- `isdigit()` (*telegram.constants.UpdateType method*), 933
- `isidentifier()` (*telegram.constants.BotCommandScopeType method*), 660
- `isidentifier()` (*telegram.constants.ChatAction method*), 683
- `isidentifier()` (*telegram.constants.ChatBoostSources method*), 689
- `isidentifier()` (*telegram.constants.ChatMemberStatus method*), 708
- `isidentifier()` (*telegram.constants.ChatType method*), 719
- `isidentifier()` (*telegram.constants.DiceEmoji method*), 733
- `isidentifier()` (*telegram.constants.InlineQueryResultType method*), 783
- `isidentifier()` (*telegram.constants.InputMediaType method*), 793
- `isidentifier()` (*telegram.constants.MaskPosition method*), 814
- `isidentifier()` (*telegram.constants.MenuButtonType method*), 824
- `isidentifier()` (*telegram.constants.MessageAttachmentType method*), 831
- `isidentifier()` (*telegram.constants.MessageEntityType method*), 839
- `isidentifier()` (*telegram.constants.MessageOriginType method*), 850
- `isidentifier()` (*telegram.constants.MessageType method*), 860
- `isidentifier()` (*telegram.constants.ParseMode method*), 866
- `isidentifier()` (*telegram.constants.PollType method*), 877
- `isidentifier()` (*telegram.constants.ReactionEmoji method*), 895
- `isidentifier()` (*telegram.constants.ReactionType method*), 901
- `isidentifier()` (*telegram.constants.StickerFormat method*), 911
- `isidentifier()` (*telegram.constants.StickerType method*), 926
- `isidentifier()` (*telegram.constants.UpdateType method*), 934
- `isnumeric()` (*telegram.constants.BotCommandScopeType method*), 660
- `isnumeric()` (*telegram.constants.ChatAction method*), 683
- `isnumeric()` (*telegram.constants.ChatBoostSources*
- `isidentifier()` (*telegram.constants.PollType method*), 877
- `isidentifier()` (*telegram.constants.ReactionEmoji method*), 895
- `isidentifier()` (*telegram.constants.ReactionType method*), 901
- `isidentifier()` (*telegram.constants.StickerFormat method*), 911
- `isidentifier()` (*telegram.constants.StickerType method*), 926
- `isidentifier()` (*telegram.constants.UpdateType method*), 933
- `islower()` (*telegram.constants.BotCommandScopeType method*), 660
- `islower()` (*telegram.constants.ChatAction method*), 683
- `islower()` (*telegram.constants.ChatBoostSources method*), 689
- `islower()` (*telegram.constants.ChatMemberStatus method*), 708
- `islower()` (*telegram.constants.ChatType method*), 719
- `islower()` (*telegram.constants.DiceEmoji method*), 733
- `islower()` (*telegram.constants.InlineQueryResultType method*), 783
- `islower()` (*telegram.constants.InputMediaType method*), 793
- `islower()` (*telegram.constants.MaskPosition method*), 814
- `islower()` (*telegram.constants.MenuButtonType method*), 824
- `islower()` (*telegram.constants.MessageAttachmentType method*), 831
- `islower()` (*telegram.constants.MessageEntityType method*), 839
- `islower()` (*telegram.constants.MessageOriginType method*), 850
- `islower()` (*telegram.constants.MessageType method*), 860
- `islower()` (*telegram.constants.ParseMode method*), 866
- `islower()` (*telegram.constants.PollType method*), 877
- `islower()` (*telegram.constants.ReactionEmoji method*), 895
- `islower()` (*telegram.constants.ReactionType method*), 901
- `islower()` (*telegram.constants.StickerFormat method*), 911
- `islower()` (*telegram.constants.StickerType method*), 926
- `islower()` (*telegram.constants.UpdateType method*), 934

- method), 689
- isnumeric() (*telegram.constants.ChatMemberStatus* method), 709
- isnumeric() (*telegram.constants.ChatType* method), 719
- isnumeric() (*telegram.constants.DiceEmoji* method), 733
- isnumeric() (*telegram.constants.InlineQueryResultType* method), 783
- isnumeric() (*telegram.constants.InputMediaType* method), 793
- isnumeric() (*telegram.constants.MaskPosition* method), 814
- isnumeric() (*telegram.constants.MenuButtonType* method), 824
- isnumeric() (*telegram.constants.MessageAttachmentType* method), 832
- isnumeric() (*telegram.constants.MessageEntityType* method), 839
- isnumeric() (*telegram.constants.MessageOriginType* method), 850
- isnumeric() (*telegram.constants.MessageType* method), 860
- isnumeric() (*telegram.constants.ParseMode* method), 866
- isnumeric() (*telegram.constants.PollType* method), 877
- isnumeric() (*telegram.constants.ReactionEmoji* method), 895
- isnumeric() (*telegram.constants.ReactionType* method), 901
- isnumeric() (*telegram.constants.StickerFormat* method), 911
- isnumeric() (*telegram.constants.StickerType* method), 926
- isnumeric() (*telegram.constants.UpdateType* method), 934
- isprintable() (*telegram.constants.BotCommandScopeType* method), 660
- isprintable() (*telegram.constants.ChatAction* method), 683
- isprintable() (*telegram.constants.ChatBoostSources* method), 689
- isprintable() (*telegram.constants.ChatMemberStatus* method), 709
- isprintable() (*telegram.constants.ChatType* method), 719
- isprintable() (*telegram.constants.DiceEmoji* method), 733
- isprintable() (*telegram.constants.InlineQueryResultType* method), 783
- isprintable() (*telegram.constants.InputMediaType* method), 793
- isprintable() (*telegram.constants.MaskPosition* method), 814
- isprintable() (*telegram.constants.MenuButtonType* method), 824
- isprintable() (*telegram.constants.MessageAttachmentType* method), 832
- isprintable() (*telegram.constants.MessageEntityType* method), 839
- isprintable() (*telegram.constants.MessageOriginType* method), 850
- isprintable() (*telegram.constants.MessageType* method), 860
- isprintable() (*telegram.constants.ParseMode* method), 866
- method), 814
- isprintable() (*telegram.constants.MenuButtonType* method), 824
- isprintable() (*telegram.constants.MessageAttachmentType* method), 832
- isprintable() (*telegram.constants.MessageEntityType* method), 839
- isprintable() (*telegram.constants.MessageOriginType* method), 850
- isprintable() (*telegram.constants.MessageType* method), 860
- isprintable() (*telegram.constants.ParseMode* method), 866
- isprintable() (*telegram.constants.PollType* method), 877
- isprintable() (*telegram.constants.ReactionEmoji* method), 895
- isprintable() (*telegram.constants.ReactionType* method), 901
- isprintable() (*telegram.constants.StickerFormat* method), 911
- isprintable() (*telegram.constants.StickerType* method), 926
- isprintable() (*telegram.constants.UpdateType* method), 934
- isspace() (*telegram.constants.BotCommandScopeType* method), 661
- isspace() (*telegram.constants.ChatAction* method), 683
- isspace() (*telegram.constants.ChatBoostSources* method), 689
- isspace() (*telegram.constants.ChatMemberStatus* method), 709
- isspace() (*telegram.constants.ChatType* method), 719
- isspace() (*telegram.constants.DiceEmoji* method), 733
- isspace() (*telegram.constants.InlineQueryResultType* method), 783
- isspace() (*telegram.constants.InputMediaType* method), 793
- isspace() (*telegram.constants.MaskPosition* method), 814
- isspace() (*telegram.constants.MenuButtonType* method), 824
- isspace() (*telegram.constants.MessageAttachmentType* method), 832
- isspace() (*telegram.constants.MessageEntityType* method), 839
- isspace() (*telegram.constants.MessageOriginType* method), 850
- isspace() (*telegram.constants.MessageType* method), 860
- isspace() (*telegram.constants.ParseMode* method), 866

- `isspace()` (*telegram.constants.PollType* method), 877
 - `isspace()` (*telegram.constants.ReactionEmoji* method), 895
 - `isspace()` (*telegram.constants.ReactionType* method), 901
 - `isspace()` (*telegram.constants.StickerFormat* method), 911
 - `isspace()` (*telegram.constants.StickerType* method), 926
 - `isspace()` (*telegram.constants.UpdateType* method), 934
 - `istitle()` (*telegram.constants.BotCommandScopeType* method), 661
 - `istitle()` (*telegram.constants.ChatAction* method), 683
 - `istitle()` (*telegram.constants.ChatBoostSources* method), 689
 - `istitle()` (*telegram.constants.ChatMemberStatus* method), 709
 - `istitle()` (*telegram.constants.ChatType* method), 719
 - `istitle()` (*telegram.constants.DiceEmoji* method), 733
 - `istitle()` (*telegram.constants.InlineQueryResultType* method), 783
 - `istitle()` (*telegram.constants.InputMediaType* method), 793
 - `istitle()` (*telegram.constants.MaskPosition* method), 814
 - `istitle()` (*telegram.constants.MenuButtonType* method), 824
 - `istitle()` (*telegram.constants.MessageAttachmentType* method), 832
 - `istitle()` (*telegram.constants.MessageEntityType* method), 839
 - `istitle()` (*telegram.constants.MessageOriginType* method), 850
 - `istitle()` (*telegram.constants.MessageType* method), 860
 - `istitle()` (*telegram.constants.ParseMode* method), 866
 - `istitle()` (*telegram.constants.PollType* method), 877
 - `istitle()` (*telegram.constants.ReactionEmoji* method), 895
 - `istitle()` (*telegram.constants.ReactionType* method), 901
 - `istitle()` (*telegram.constants.StickerFormat* method), 911
 - `istitle()` (*telegram.constants.StickerType* method), 926
 - `istitle()` (*telegram.constants.UpdateType* method), 934
 - `isupper()` (*telegram.constants.BotCommandScopeType* method), 661
 - `isupper()` (*telegram.constants.ChatAction* method), 684
 - `isupper()` (*telegram.constants.ChatBoostSources* method), 690
 - `isupper()` (*telegram.constants.ChatMemberStatus* method), 709
 - `isupper()` (*telegram.constants.ChatType* method), 719
 - `isupper()` (*telegram.constants.DiceEmoji* method), 733
 - `isupper()` (*telegram.constants.InlineQueryResultType* method), 783
 - `isupper()` (*telegram.constants.InputMediaType* method), 794
 - `isupper()` (*telegram.constants.MaskPosition* method), 815
 - `isupper()` (*telegram.constants.MenuButtonType* method), 825
 - `isupper()` (*telegram.constants.MessageAttachmentType* method), 832
 - `isupper()` (*telegram.constants.MessageEntityType* method), 839
 - `isupper()` (*telegram.constants.MessageOriginType* method), 850
 - `isupper()` (*telegram.constants.MessageType* method), 861
 - `isupper()` (*telegram.constants.ParseMode* method), 867
 - `isupper()` (*telegram.constants.PollType* method), 877
 - `isupper()` (*telegram.constants.ReactionEmoji* method), 895
 - `isupper()` (*telegram.constants.ReactionType* method), 901
 - `isupper()` (*telegram.constants.StickerFormat* method), 911
 - `isupper()` (*telegram.constants.StickerType* method), 927
 - `isupper()` (*telegram.constants.UpdateType* method), 934
 - `ITALIC` (*telegram.constants.MessageEntityType* attribute), 835
 - `ITALIC` (*telegram.MessageEntity* attribute), 343
- ## J
- `JACK_O_LANTERN` (*telegram.constants.ReactionEmoji* attribute), 888
 - `Job` (class in *telegram.ext*), 559
 - `job` (*telegram.ext.CallbackContext* attribute), 549
 - `job` (*telegram.ext.Job* property), 561
 - `job_callback()` (*telegram.ext.JobQueue* static method), 563
 - `job_queue` (*telegram.ext.Application* property), 523
 - `job_queue` (*telegram.ext.CallbackContext* property), 551
 - `job_queue()` (*telegram.ext.ApplicationBuilder* method), 538
 - `JobQueue` (class in *telegram.ext*), 562
 - `jobs()` (*telegram.ext.JobQueue* method), 563
 - `join()` (*telegram.constants.BotCommandScopeType* method), 661
 - `join()` (*telegram.constants.ChatAction* method), 684

join() (*telegram.constants.ChatBoostSources* method), 690
 join() (*telegram.constants.ChatMemberStatus* method), 709
 join() (*telegram.constants.ChatType* method), 719
 join() (*telegram.constants.DiceEmoji* method), 734
 join() (*telegram.constants.InlineQueryResultType* method), 784
 join() (*telegram.constants.InputMediaType* method), 794
 join() (*telegram.constants.MaskPosition* method), 815
 join() (*telegram.constants.MenuButtonType* method), 825
 join() (*telegram.constants.MessageAttachmentType* method), 832
 join() (*telegram.constants.MessageEntityType* method), 839
 join() (*telegram.constants.MessageOriginType* method), 850
 join() (*telegram.constants.MessageType* method), 861
 join() (*telegram.constants.ParseMode* method), 867
 join() (*telegram.constants.PollType* method), 877
 join() (*telegram.constants.ReactionEmoji* method), 895
 join() (*telegram.constants.ReactionType* method), 901
 join() (*telegram.constants.StickerFormat* method), 911
 join() (*telegram.constants.StickerType* method), 927
 join() (*telegram.constants.UpdateType* method), 934
 join_by_request (*telegram.Chat* attribute), 178
 join_to_send_messages (*telegram.Chat* attribute), 178
 JPG (*telegram.ext.filters.Document* attribute), 600
 json_parameters (*telegram.request.RequestData* property), 954
 json_payload (*telegram.request.RequestData* property), 954

K

keyboard (*telegram.ReplyKeyboardMarkup* attribute), 364
 KeyboardButton (class in *telegram*), 278
 KeyboardButtonPollType (class in *telegram*), 281
 KeyboardButtonRequestChat (class in *telegram*), 281
 KeyboardButtonRequestUser (class in *telegram*), 283
 KeyboardButtonRequestUsers (class in *telegram*), 283
 KeyboardButtonRequestUsersLimit (class in *telegram.constants*), 801
 keywords (*telegram.InputSticker* attribute), 278
 KISS_MARK (*telegram.constants.ReactionEmoji* attribute), 888

L

label (*telegram.LabeledPrice* attribute), 483
 LabeledPrice (class in *telegram*), 483
 Language (class in *telegram.ext.filters*), 603
 language (*telegram.MessageEntity* attribute), 341
 language_code (*telegram.User* attribute), 386
 last_error_date (*telegram.WebhookInfo* attribute), 414
 last_error_message (*telegram.WebhookInfo* attribute), 415
 last_name (*telegram.Bot* property), 81
 last_name (*telegram.Chat* attribute), 177
 last_name (*telegram.Contact* attribute), 235
 last_name (*telegram.InlineQueryResultContact* attribute), 446
 last_name (*telegram.InputContactMessageContent* attribute), 476
 last_name (*telegram.PersonalDetails* attribute), 510
 last_name (*telegram.User* attribute), 386
 last_name_native (*telegram.PersonalDetails* attribute), 511
 last_synchronization_error_date (*telegram.WebhookInfo* attribute), 415
 latitude (*telegram.InlineQueryResultLocation* attribute), 454
 latitude (*telegram.InlineQueryResultVenue* attribute), 463
 latitude (*telegram.InputLocationMessageContent* attribute), 472
 latitude (*telegram.InputVenueMessageContent* attribute), 474
 latitude (*telegram.Location* attribute), 286
 leave() (*telegram.Chat* method), 189
 leave_chat() (*telegram.Bot* method), 82
 leaveChat() (*telegram.Bot* method), 81
 LEFT (*telegram.ChatMember* attribute), 218
 LEFT (*telegram.constants.ChatMemberStatus* attribute), 706
 LEFT_CHAT_MEMBER (*telegram.constants.MessageType* attribute), 855
 LEFT_CHAT_MEMBER (*telegram.ext.filters.StatusUpdate* attribute), 608
 left_chat_member (*telegram.Message* attribute), 303
 length (*telegram.MessageEntity* attribute), 341
 length (*telegram.VideoNote* attribute), 410
 link (*telegram.Bot* property), 82
 link (*telegram.Chat* property), 189
 link (*telegram.Message* property), 319
 link (*telegram.User* property), 391
 link_preview_options (*telegram.ext.Defaults* property), 556
 link_preview_options (*telegram.ExternalReplyInfo* attribute), 241
 link_preview_options (*telegram.InputTextMessageContent* attribute), 471
 link_preview_options (*telegram.Message* attribute), 301

- `linked_chat_id` (*telegram.Chat* attribute), 178
- `LinkPreviewOptions` (class in *telegram*), 284
- `live_period` (*telegram.InlineQueryResultLocation* attribute), 454
- `live_period` (*telegram.InputLocationMessageContent* attribute), 472
- `live_period` (*telegram.Location* attribute), 286
- `ljust()` (*telegram.constants.BotCommandScopeType* method), 661
- `ljust()` (*telegram.constants.ChatAction* method), 684
- `ljust()` (*telegram.constants.ChatBoostSources* method), 690
- `ljust()` (*telegram.constants.ChatMemberStatus* method), 709
- `ljust()` (*telegram.constants.ChatType* method), 719
- `ljust()` (*telegram.constants.DiceEmoji* method), 734
- `ljust()` (*telegram.constants.InlineQueryResultType* method), 784
- `ljust()` (*telegram.constants.InputMediaType* method), 794
- `ljust()` (*telegram.constants.MaskPosition* method), 815
- `ljust()` (*telegram.constants.MenuButtonType* method), 825
- `ljust()` (*telegram.constants.MessageAttachmentType* method), 832
- `ljust()` (*telegram.constants.MessageEntityType* method), 839
- `ljust()` (*telegram.constants.MessageOriginType* method), 850
- `ljust()` (*telegram.constants.MessageType* method), 861
- `ljust()` (*telegram.constants.ParseMode* method), 867
- `ljust()` (*telegram.constants.PollType* method), 877
- `ljust()` (*telegram.constants.ReactionEmoji* method), 895
- `ljust()` (*telegram.constants.ReactionType* method), 901
- `ljust()` (*telegram.constants.StickerFormat* method), 911
- `ljust()` (*telegram.constants.StickerType* method), 927
- `ljust()` (*telegram.constants.UpdateType* method), 934
- `load_persistence_data()` (*telegram.ext.CallbackDataCache* method), 645
- `local_mode` (*telegram.Bot* property), 82
- `local_mode()` (*telegram.ext.ApplicationBuilder* method), 539
- `Location` (class in *telegram*), 286
- `LOCATION` (in module *telegram.ext.filters*), 591
- `location` (*telegram.Chat* attribute), 178
- `location` (*telegram.ChatLocation* attribute), 216
- `location` (*telegram.ChosenInlineResult* attribute), 423
- `LOCATION` (*telegram.constants.InlineQueryResultType* attribute), 780
- `LOCATION` (*telegram.constants.MessageAttachmentType* attribute), 828
- `LOCATION` (*telegram.constants.MessageType* attribute), 855
- `location` (*telegram.ExternalReplyInfo* attribute), 243
- `location` (*telegram.InlineQuery* attribute), 425
- `location` (*telegram.Message* attribute), 303
- `location` (*telegram.Venue* attribute), 404
- `LocationLimit` (class in *telegram.constants*), 806
- `log_out()` (*telegram.Bot* method), 82
- `login_url` (*telegram.InlineKeyboardButton* attribute), 259
- `LoginUrl` (class in *telegram*), 287
- `logout()` (*telegram.Bot* method), 82
- `longitude` (*telegram.InlineQueryResultLocation* attribute), 454
- `longitude` (*telegram.InlineQueryResultVenue* attribute), 463
- `longitude` (*telegram.InputLocationMessageContent* attribute), 472
- `longitude` (*telegram.InputVenueMessageContent* attribute), 474
- `longitude` (*telegram.Location* attribute), 286
- `LOUDLY_CRYING_FACE` (*telegram.constants.ReactionEmoji* attribute), 889
- `lower()` (*telegram.constants.BotCommandScopeType* method), 661
- `lower()` (*telegram.constants.ChatAction* method), 684
- `lower()` (*telegram.constants.ChatBoostSources* method), 690
- `lower()` (*telegram.constants.ChatMemberStatus* method), 709
- `lower()` (*telegram.constants.ChatType* method), 719
- `lower()` (*telegram.constants.DiceEmoji* method), 734
- `lower()` (*telegram.constants.InlineQueryResultType* method), 784
- `lower()` (*telegram.constants.InputMediaType* method), 794
- `lower()` (*telegram.constants.MaskPosition* method), 815
- `lower()` (*telegram.constants.MenuButtonType* method), 825
- `lower()` (*telegram.constants.MessageAttachmentType* method), 832
- `lower()` (*telegram.constants.MessageEntityType* method), 840
- `lower()` (*telegram.constants.MessageOriginType* method), 850
- `lower()` (*telegram.constants.MessageType* method), 861
- `lower()` (*telegram.constants.ParseMode* method), 867
- `lower()` (*telegram.constants.PollType* method), 878
- `lower()` (*telegram.constants.ReactionEmoji* method), 896
- `lower()` (*telegram.constants.ReactionType* method), 901
- `lower()` (*telegram.constants.StickerFormat* method), 911
- `lower()` (*telegram.constants.StickerType* method), 927
- `lower()` (*telegram.constants.UpdateType* method), 934

- [lstrip\(\)](#) (*telegram.constants.BotCommandScopeType* method), 661
[lstrip\(\)](#) (*telegram.constants.ChatAction* method), 684
[lstrip\(\)](#) (*telegram.constants.ChatBoostSources* method), 690
[lstrip\(\)](#) (*telegram.constants.ChatMemberStatus* method), 709
[lstrip\(\)](#) (*telegram.constants.ChatType* method), 719
[lstrip\(\)](#) (*telegram.constants.DiceEmoji* method), 734
[lstrip\(\)](#) (*telegram.constants.InlineQueryResultType* method), 784
[lstrip\(\)](#) (*telegram.constants.InputMediaType* method), 794
[lstrip\(\)](#) (*telegram.constants.MaskPosition* method), 815
[lstrip\(\)](#) (*telegram.constants.MenuButtonType* method), 825
[lstrip\(\)](#) (*telegram.constants.MessageAttachmentType* method), 832
[lstrip\(\)](#) (*telegram.constants.MessageEntityType* method), 840
[lstrip\(\)](#) (*telegram.constants.MessageOriginType* method), 850
[lstrip\(\)](#) (*telegram.constants.MessageType* method), 861
[lstrip\(\)](#) (*telegram.constants.ParseMode* method), 867
[lstrip\(\)](#) (*telegram.constants.PollType* method), 878
[lstrip\(\)](#) (*telegram.constants.ReactionEmoji* method), 896
[lstrip\(\)](#) (*telegram.constants.ReactionType* method), 901
[lstrip\(\)](#) (*telegram.constants.StickerFormat* method), 912
[lstrip\(\)](#) (*telegram.constants.StickerType* method), 927
[lstrip\(\)](#) (*telegram.constants.UpdateType* method), 934
- M**
- [maketrans\(\)](#) (*telegram.constants.BotCommandScopeType* static method), 661
[maketrans\(\)](#) (*telegram.constants.ChatAction* static method), 684
[maketrans\(\)](#) (*telegram.constants.ChatBoostSources* static method), 690
[maketrans\(\)](#) (*telegram.constants.ChatMemberStatus* static method), 709
[maketrans\(\)](#) (*telegram.constants.ChatType* static method), 719
[maketrans\(\)](#) (*telegram.constants.DiceEmoji* static method), 734
[maketrans\(\)](#) (*telegram.constants.InlineQueryResultType* static method), 784
[maketrans\(\)](#) (*telegram.constants.InputMediaType* static method), 794
[maketrans\(\)](#) (*telegram.constants.MaskPosition* static method), 815
[maketrans\(\)](#) (*telegram.constants.MenuButtonType* static method), 825
[maketrans\(\)](#) (*telegram.constants.MessageAttachmentType* static method), 832
[maketrans\(\)](#) (*telegram.constants.MessageEntityType* static method), 840
[maketrans\(\)](#) (*telegram.constants.MessageOriginType* static method), 850
[maketrans\(\)](#) (*telegram.constants.MessageType* static method), 861
[maketrans\(\)](#) (*telegram.constants.ParseMode* static method), 867
[maketrans\(\)](#) (*telegram.constants.PollType* static method), 878
[maketrans\(\)](#) (*telegram.constants.ReactionEmoji* static method), 896
[maketrans\(\)](#) (*telegram.constants.ReactionType* static method), 901
[maketrans\(\)](#) (*telegram.constants.StickerFormat* static method), 912
[maketrans\(\)](#) (*telegram.constants.StickerType* static method), 927
[maketrans\(\)](#) (*telegram.constants.UpdateType* static method), 934
[MAN_SHRUGGING](#) (*telegram.constants.ReactionEmoji* attribute), 889
[MAN_TECHNOLOGIST](#) (*telegram.constants.ReactionEmoji* attribute), 889
[map_to_parent](#) (*telegram.ext.ConversationHandler* property), 589
[mark_data_for_update_persistence\(\)](#) (*telegram.ext.Application* method), 523
[MARKDOWN](#) (*telegram.constants.ParseMode* attribute), 863
[MARKDOWN_V2](#) (*telegram.constants.ParseMode* attribute), 864
[MASK](#) (*telegram.constants.StickerType* attribute), 923
[MASK](#) (*telegram.Sticker* attribute), 420
[mask_position](#) (*telegram.InputSticker* attribute), 278
[mask_position](#) (*telegram.Sticker* attribute), 420
[MaskPosition](#) (class in *telegram*), 416
[MaskPosition](#) (class in *telegram.constants*), 811
[match](#) (*telegram.ext.CallbackContext* property), 551
[matches](#) (*telegram.ext.CallbackContext* attribute), 548
[MAX_ADDRESS](#) (*telegram.ChatLocation* attribute), 216
[MAX_ANIMATED_STICKERS](#) (*telegram.constants.StickerSetLimit* attribute), 919
[MAX_ANIMATED_THUMBNAIL_SIZE](#) (*telegram.constants.StickerSetLimit* attribute), 919
[MAX_ANSWER_TEXT_LENGTH](#) (*telegram.CallbackQuery* attribute), 167
[MAX_CALLBACK_DATA](#) (*telegram.constants.InlineKeyboardButtonLimit*

<i>attribute</i>), 762	
MAX_CALLBACK_DATA (<i>telegram.InlineKeyboardButton attribute</i>), 260	MAX_INITIAL_STICKERS (<i>telegram.constants.StickerSetLimit attribute</i>), 919
MAX_CHAT_LOCATION_ADDRESS (<i>telegram.constants.LocationLimit attribute</i>), 806	MAX_INPUT_FIELD_PLACEHOLDER (<i>telegram.constants.ReplyLimit attribute</i>), 904
MAX_CHAT_TITLE_LENGTH (<i>telegram.constants.ChatLimit attribute</i>), 701	MAX_INPUT_FIELD_PLACEHOLDER (<i>telegram.ForceReply attribute</i>), 247
MAX_COMMAND (<i>telegram.BotCommand attribute</i>), 159	MAX_INPUT_FIELD_PLACEHOLDER (<i>telegram.ReplyKeyboardMarkup attribute</i>), 364
MAX_COMMAND (<i>telegram.constants.BotCommandLimit attribute</i>), 653	MAX_KEYWORD_LENGTH (<i>telegram.constants.StickerLimit attribute</i>), 914
MAX_COMMAND_NUMBER (<i>telegram.constants.BotCommandLimit attribute</i>), 653	MAX_LENGTH (<i>telegram.BotName attribute</i>), 165
max_concurrent_updates (<i>telegram.ext.BaseUpdateProcessor property</i>), 547	MAX_LENGTH (<i>telegram.PollOption attribute</i>), 357
max_connections (<i>telegram.WebhookInfo attribute</i>), 415	MAX_LIMIT (<i>telegram.constants.BulkRequestLimit attribute</i>), 671
MAX_CONNECTIONS_LIMIT (<i>telegram.constants.WebhookLimit attribute</i>), 941	MAX_LIMIT (<i>telegram.constants.PollingLimit attribute</i>), 880
MAX_DESCRIPTION (<i>telegram.BotCommand attribute</i>), 159	MAX_LIMIT (<i>telegram.constants.UserProfilePhotosLimit attribute</i>), 937
MAX_DESCRIPTION (<i>telegram.constants.BotCommandLimit attribute</i>), 653	MAX_LIVE_PERIOD (<i>telegram.constants.LocationLimit attribute</i>), 806
MAX_DESCRIPTION_LENGTH (<i>telegram.constants.BotDescriptionLimit attribute</i>), 663	MAX_LIVE_PERIOD (<i>telegram.InlineQueryResultLocation attribute</i>), 455
MAX_DESCRIPTION_LENGTH (<i>telegram.constants.InvoiceLimit attribute</i>), 796	MAX_LIVE_PERIOD (<i>telegram.InputLocationMessageContent attribute</i>), 473
MAX_DESCRIPTION_LENGTH (<i>telegram.Invoice attribute</i>), 482	MAX_MEDIA_LENGTH (<i>telegram.constants.MediaGroupLimit attribute</i>), 817
MAX_EMOJI_STICKERS (<i>telegram.constants.StickerSetLimit attribute</i>), 919	MAX_MEMBER_LIMIT (<i>telegram.constants.ChatInviteLinkLimit attribute</i>), 697
MAX_EXPLANATION_LENGTH (<i>telegram.constants.PollLimit attribute</i>), 869	MAX_NAME_AND_TITLE (<i>telegram.constants.StickerLimit attribute</i>), 914
MAX_EXPLANATION_LENGTH (<i>telegram.Poll attribute</i>), 354	MAX_NAME_LENGTH (<i>telegram.constants.BotNameLimit attribute</i>), 667
MAX_EXPLANATION_LINE_FEEDS (<i>telegram.constants.PollLimit attribute</i>), 869	MAX_NAME_LENGTH (<i>telegram.constants.ForumTopicLimit attribute</i>), 754
MAX_EXPLANATION_LINE_FEEDS (<i>telegram.Poll attribute</i>), 354	MAX_OFFSET_LENGTH (<i>telegram.constants.InlineQueryLimit attribute</i>), 771
MAX_HEADING (<i>telegram.constants.LocationLimit attribute</i>), 806	MAX_OFFSET_LENGTH (<i>telegram.InlineQuery attribute</i>), 425
MAX_HEADING (<i>telegram.InlineQueryResultLocation attribute</i>), 455	MAX_OPEN_PERIOD (<i>telegram.constants.PollLimit attribute</i>), 869
MAX_HEADING (<i>telegram.InputLocationMessageContent attribute</i>), 473	MAX_OPEN_PERIOD (<i>telegram.Poll attribute</i>), 354
MAX_HEADING (<i>telegram.Location attribute</i>), 287	MAX_OPTION_LENGTH (<i>telegram.constants.PollLimit attribute</i>), 870
MAX_ID_LENGTH (<i>telegram.constants.InlineQueryResultLimit attribute</i>), 775	MAX_OPTION_LENGTH (<i>telegram.Poll attribute</i>), 354
MAX_ID_LENGTH (<i>telegram.InlineQueryResult attribute</i>), 427	MAX_OPTION_NUMBER (<i>telegram.constants.PollLimit attribute</i>), 870
	MAX_OPTION_NUMBER (<i>telegram.Poll attribute</i>), 354
	MAX_PAYLOAD_LENGTH (<i>telegram</i>), 914

<code>gram.constants.InvoiceLimit</code> attribute), 796	<code>max_tip_amount</code> (<code>telegram.InputInvoiceMessageContent</code> attribute), 479
<code>MAX_PAYLOAD_LENGTH</code> (<code>telegram.Invoice</code> attribute), 482	<code>MAX_TIP_AMOUNTS</code> (<code>telegram.constants.InvoiceLimit</code> attribute), 797
<code>MAX_PROXIMITY_ALERT_RADIUS</code> (<code>telegram.constants.LocationLimit</code> attribute), 806	<code>MAX_TIP_AMOUNTS</code> (<code>telegram.Invoice</code> attribute), 482
<code>MAX_PROXIMITY_ALERT_RADIUS</code> (<code>telegram.InlineQueryResultLocation</code> attribute), 455	<code>MAX_TITLE_LENGTH</code> (<code>telegram.constants.InvoiceLimit</code> attribute), 797
<code>MAX_PROXIMITY_ALERT_RADIUS</code> (<code>telegram.InputLocationMessageContent</code> attribute), 473	<code>MAX_TITLE_LENGTH</code> (<code>telegram.Invoice</code> attribute), 482
<code>MAX_QUANTITY</code> (<code>telegram.constants.KeyboardButtonRequestUsersLimit</code> attribute), 801	<code>MAX_VALUE_BASKETBALL</code> (<code>telegram.constants.DiceLimit</code> attribute), 736
<code>max_quantity</code> (<code>telegram.KeyboardButtonRequestUsers</code> attribute), 284	<code>MAX_VALUE_BASKETBALL</code> (<code>telegram.Dice</code> attribute), 237
<code>MAX_QUERY_LENGTH</code> (<code>telegram.constants.InlineQueryLimit</code> attribute), 771	<code>MAX_VALUE_BOWLING</code> (<code>telegram.constants.DiceLimit</code> attribute), 736
<code>MAX_QUERY_LENGTH</code> (<code>telegram.InlineQuery</code> attribute), 426	<code>MAX_VALUE_BOWLING</code> (<code>telegram.Dice</code> attribute), 237
<code>MAX_QUESTION_LENGTH</code> (<code>telegram.constants.PollLimit</code> attribute), 870	<code>MAX_VALUE_DARTS</code> (<code>telegram.constants.DiceLimit</code> attribute), 736
<code>MAX_QUESTION_LENGTH</code> (<code>telegram.Poll</code> attribute), 355	<code>MAX_VALUE_DARTS</code> (<code>telegram.Dice</code> attribute), 237
<code>MAX_RESULTS</code> (<code>telegram.InlineQuery</code> attribute), 426	<code>MAX_VALUE_DICE</code> (<code>telegram.constants.DiceLimit</code> attribute), 736
<code>MAX_SEARCH_KEYWORDS</code> (<code>telegram.constants.StickerLimit</code> attribute), 914	<code>MAX_VALUE_DICE</code> (<code>telegram.Dice</code> attribute), 237
<code>MAX_SECRET_TOKEN_LENGTH</code> (<code>telegram.constants.WebhookLimit</code> attribute), 941	<code>MAX_VALUE_DICE</code> (<code>telegram.Dice</code> attribute), 237
<code>MAX_SHORT_DESCRIPTION_LENGTH</code> (<code>telegram.constants.BotDescriptionLimit</code> attribute), 663	<code>MAX_VALUE_FOOTBALL</code> (<code>telegram.constants.DiceLimit</code> attribute), 736
<code>MAX_START_PARAMETER_LENGTH</code> (<code>telegram.constants.InlineQueryResultsButtonLimit</code> attribute), 786	<code>MAX_VALUE_FOOTBALL</code> (<code>telegram.Dice</code> attribute), 237
<code>MAX_START_PARAMETER_LENGTH</code> (<code>telegram.InlineQueryResultsButton</code> attribute), 461	<code>MAX_VALUE_SLOT_MACHINE</code> (<code>telegram.constants.DiceLimit</code> attribute), 737
<code>MAX_STATIC_STICKERS</code> (<code>telegram.constants.StickerSetLimit</code> attribute), 919	<code>MAX_VALUE_SLOT_MACHINE</code> (<code>telegram.Dice</code> attribute), 238
<code>MAX_STATIC_THUMBNAIL_SIZE</code> (<code>telegram.constants.StickerSetLimit</code> attribute), 919	<code>MAX_WINNERS</code> (<code>telegram.constants.GiveawayLimit</code> attribute), 758
<code>MAX_STICKER_EMOJI</code> (<code>telegram.constants.StickerLimit</code> attribute), 914	<code>maxsize</code> (<code>telegram.ext.CallbackDataCache</code> property), 645
<code>MAX_SWITCH_PM_TEXT_LENGTH</code> (<code>telegram.constants.InlineQueryLimit</code> attribute), 771	<code>MaybeInaccessibleMessage</code> (class in <code>telegram</code>), 288
<code>MAX_SWITCH_PM_TEXT_LENGTH</code> (<code>telegram.InlineQuery</code> attribute), 426	<code>media</code> (<code>telegram.InputMedia</code> attribute), 266
<code>MAX_TEXT_LENGTH</code> (<code>telegram.constants.MessageLimit</code> attribute), 842	<code>media</code> (<code>telegram.InputMediaAnimation</code> attribute), 268
	<code>media</code> (<code>telegram.InputMediaAudio</code> attribute), 270
	<code>media</code> (<code>telegram.InputMediaDocument</code> attribute), 272
	<code>media</code> (<code>telegram.InputMediaPhoto</code> attribute), 274
	<code>media</code> (<code>telegram.InputMediaVideo</code> attribute), 276
	<code>media_group_id</code> (<code>telegram.Message</code> attribute), 301
	<code>MediaGroupLimit</code> (class in <code>telegram.constants</code>), 817
	<code>MEMBER</code> (<code>telegram.ChatMember</code> attribute), 218
	<code>MEMBER</code> (<code>telegram.constants.ChatMemberStatus</code> attribute), 706
	<code>member_limit</code> (<code>telegram.ChatInviteLink</code> attribute), 213
	<code>Mention</code> (class in <code>telegram.ext.filters</code>), 603
	<code>MENTION</code> (<code>telegram.constants.MessageEntityType</code> attribute), 835
	<code>MENTION</code> (<code>telegram.MessageEntity</code> attribute), 343
	<code>mention_button()</code> (<code>telegram.User</code> method), 391
	<code>mention_html()</code> (in module <code>telegram.helpers</code>), 949
	<code>mention_html()</code> (<code>telegram.Chat</code> method), 189
	<code>mention_html()</code> (<code>telegram.User</code> method), 391
	<code>mention_markdown()</code> (in module <code>telegram.helpers</code>), 949
	<code>mention_markdown()</code> (<code>telegram.Chat</code> method), 190

- `mention_markdown()` (*telegram.User* method), 391
- `mention_markdown_v2()` (*telegram.Chat* method), 190
- `mention_markdown_v2()` (*telegram.User* method), 391
- `MenuButton` (class in *telegram*), 290
- `MenuButtonCommands` (class in *telegram*), 291
- `MenuButtonDefault` (class in *telegram*), 291
- `MenuButtonType` (class in *telegram.constants*), 821
- `MenuButtonWebApp` (class in *telegram*), 292
- `Message` (class in *telegram*), 293
- `message` (*telegram.CallbackQuery* attribute), 167
- `MESSAGE` (*telegram.constants.UpdateType* attribute), 930
- `MESSAGE` (*telegram.ext.filters.UpdateType* attribute), 612
- `message` (*telegram.PassportElementError* attribute), 501
- `message` (*telegram.PassportElementErrorDataField* attribute), 502
- `message` (*telegram.PassportElementErrorFile* attribute), 503
- `message` (*telegram.PassportElementErrorFiles* attribute), 503
- `message` (*telegram.PassportElementErrorFrontSide* attribute), 504
- `message` (*telegram.PassportElementErrorReverseSide* attribute), 505
- `message` (*telegram.PassportElementErrorSelfie* attribute), 505
- `message` (*telegram.PassportElementErrorTranslationFile* attribute), 506
- `message` (*telegram.PassportElementErrorTranslationFiles* attribute), 507
- `message` (*telegram.PassportElementErrorUnspecified* attribute), 508
- `MESSAGE` (*telegram.Update* attribute), 382
- `message` (*telegram.Update* attribute), 378
- `message_auto_delete_time` (*telegram.Chat* attribute), 178
- `message_auto_delete_time` (*telegram.MessageAutoDeleteTimerChanged* attribute), 339
- `MESSAGE_AUTO_DELETE_TIMER_CHANGED` (*telegram.constants.MessageType* attribute), 855
- `MESSAGE_AUTO_DELETE_TIMER_CHANGED` (*telegram.ext.filters.StatusUpdate* attribute), 608
- `message_auto_delete_timer_changed` (*telegram.Message* attribute), 304
- `MESSAGE_ENTITIES` (*telegram.constants.MessageLimit* attribute), 843
- `message_id` (*telegram.ExternalReplyInfo* attribute), 241
- `message_id` (*telegram.InaccessibleMessage* attribute), 256
- `message_id` (*telegram.MaybeInaccessibleMessage* attribute), 289
- `message_id` (*telegram.Message* attribute), 300
- `message_id` (*telegram.MessageId* attribute), 344
- `message_id` (*telegram.MessageOriginChannel* attribute), 346
- `message_id` (*telegram.MessageReactionCountUpdated* attribute), 348
- `message_id` (*telegram.MessageReactionUpdated* attribute), 350
- `message_id` (*telegram.ReplyParameters* attribute), 369
- `MESSAGE_REACTION` (*telegram.constants.UpdateType* attribute), 930
- `MESSAGE_REACTION` (*telegram.ext.MessageReactionHandler* attribute), 619
- `MESSAGE_REACTION` (*telegram.Update* attribute), 382
- `message_reaction` (*telegram.Update* attribute), 380
- `MESSAGE_REACTION_COUNT` (*telegram.constants.UpdateType* attribute), 930
- `MESSAGE_REACTION_COUNT` (*telegram.Update* attribute), 382
- `message_reaction_count` (*telegram.Update* attribute), 381
- `MESSAGE_REACTION_COUNT_UPDATED` (*telegram.ext.MessageReactionHandler* attribute), 619
- `message_reaction_types` (*telegram.ext.MessageReactionHandler* attribute), 619
- `MESSAGE_REACTION_UPDATED` (*telegram.ext.MessageReactionHandler* attribute), 619
- `message_text` (*telegram.InputTextMessageContent* attribute), 470
- `message_thread_id` (*telegram.ForumTopic* attribute), 248
- `message_thread_id` (*telegram.Message* attribute), 306
- `MessageAttachmentType` (class in *telegram.constants*), 827
- `MessageAutoDeleteTimerChanged` (class in *telegram*), 339
- `MessageEntity` (class in *telegram*), 340
- `MessageEntityType` (class in *telegram.constants*), 834
- `MessageFilter` (class in *telegram.ext.filters*), 603
- `MessageHandler` (class in *telegram.ext*), 616
- `MessageId` (class in *telegram*), 344
- `MessageLimit` (class in *telegram.constants*), 842
- `MessageOrigin` (class in *telegram*), 344
- `MessageOriginChannel` (class in *telegram*), 345
- `MessageOriginChat` (class in *telegram*), 346
- `MessageOriginHiddenUser` (class in *telegram*), 347
- `MessageOriginType` (class in *telegram.constants*), 847
- `MessageOriginUser` (class in *telegram*), 347

- [MessageReactionCountUpdated \(class in telegram\), 348](#)
[MessageReactionHandler \(class in telegram.ext\), 618](#)
[MessageReactionUpdated \(class in telegram\), 349](#)
[MESSAGES \(telegram.ext.filters.UpdateType attribute\), 612](#)
[MESSAGES_PER_MINUTE_PER_GROUP \(telegram.constants.FloodLimit attribute\), 745](#)
[MESSAGES_PER_SECOND \(telegram.constants.FloodLimit attribute\), 745](#)
[MESSAGES_PER_SECOND_PER_CHAT \(telegram.constants.FloodLimit attribute\), 745](#)
[MessageType \(class in telegram.constants\), 852](#)
[middle_name \(telegram.PersonalDetails attribute\), 510](#)
[middle_name_native \(telegram.PersonalDetails attribute\), 511](#)
[MIGRATE \(telegram.ext.filters.StatusUpdate attribute\), 608](#)
[migrate_chat_data\(\) \(telegram.ext.Application method\), 523](#)
[migrate_from_chat_id \(telegram.Message attribute\), 304](#)
[MIGRATE_TO_CHAT_ID \(telegram.constants.MessageType attribute\), 855](#)
[migrate_to_chat_id \(telegram.Message attribute\), 304](#)
[mime_type \(telegram.Animation attribute\), 156](#)
[mime_type \(telegram.Audio attribute\), 158](#)
[mime_type \(telegram.Document attribute\), 239](#)
[mime_type \(telegram.InlineQueryResultDocument attribute\), 448](#)
[mime_type \(telegram.InlineQueryResultVideo attribute\), 466](#)
[mime_type \(telegram.Video attribute\), 407](#)
[mime_type \(telegram.Voice attribute\), 411](#)
[mimetype \(telegram.InputFile attribute\), 265](#)
[MIN_ADDRESS \(telegram.ChatLocation attribute\), 216](#)
[MIN_CALLBACK_DATA \(telegram.constants.InlineKeyboardButtonLimit attribute\), 762](#)
[MIN_CALLBACK_DATA \(telegram.InlineKeyboardButton attribute\), 260](#)
[MIN_CHAT_LOCATION_ADDRESS \(telegram.constants.LocationLimit attribute\), 807](#)
[MIN_CHAT_TITLE_LENGTH \(telegram.constants.ChatLimit attribute\), 701](#)
[MIN_COMMAND \(telegram.BotCommand attribute\), 159](#)
[MIN_COMMAND \(telegram.constants.BotCommandLimit attribute\), 653](#)
[MIN_CONNECTIONS_LIMIT \(telegram.constants.WebhookLimit attribute\), 941](#)
[MIN_DESCRIPTION \(telegram.BotCommand attribute\), 159](#)
[MIN_DESCRIPTION \(telegram.constants.BotCommandLimit attribute\), 653](#)
[MIN_DESCRIPTION_LENGTH \(telegram.constants.InvoiceLimit attribute\), 797](#)
[MIN_DESCRIPTION_LENGTH \(telegram.Invoice attribute\), 482](#)
[MIN_HEADING \(telegram.constants.LocationLimit attribute\), 807](#)
[MIN_HEADING \(telegram.InlineQueryResultLocation attribute\), 455](#)
[MIN_HEADING \(telegram.InputLocationMessageContent attribute\), 473](#)
[MIN_HEADING \(telegram.Location attribute\), 287](#)
[MIN_ID_LENGTH \(telegram.constants.InlineQueryResultLimit attribute\), 775](#)
[MIN_ID_LENGTH \(telegram.InlineQueryResult attribute\), 427](#)
[MIN_INITIAL_STICKERS \(telegram.constants.StickerSetLimit attribute\), 919](#)
[MIN_INPUT_FIELD_PLACEHOLDER \(telegram.constants.ReplyLimit attribute\), 904](#)
[MIN_INPUT_FIELD_PLACEHOLDER \(telegram.ForceReply attribute\), 247](#)
[MIN_INPUT_FIELD_PLACEHOLDER \(telegram.ReplyKeyboardMarkup attribute\), 364](#)
[MIN_LENGTH \(telegram.PollOption attribute\), 358](#)
[MIN_LIMIT \(telegram.constants.BulkRequestLimit attribute\), 672](#)
[MIN_LIMIT \(telegram.constants.PollingLimit attribute\), 880](#)
[MIN_LIMIT \(telegram.constants.UserProfilePhotosLimit attribute\), 937](#)
[MIN_LIVE_PERIOD \(telegram.constants.LocationLimit attribute\), 807](#)
[MIN_LIVE_PERIOD \(telegram.InlineQueryResultLocation attribute\), 455](#)
[MIN_LIVE_PERIOD \(telegram.InputLocationMessageContent attribute\), 474](#)
[MIN_MEDIA_LENGTH \(telegram.constants.MediaGroupLimit attribute\), 817](#)
[MIN_MEMBER_LIMIT \(telegram.constants.ChatInviteLinkLimit attribute\), 697](#)
[MIN_NAME_AND_TITLE \(telegram.constants.StickerLimit attribute\), 914](#)
[MIN_NAME_LENGTH \(telegram.constants.StickerLimit attribute\), 914](#)

- gram.constants.ForumTopicLimit* attribute), 754
- MIN_OPEN_PERIOD* (*telegram.constants.PollLimit* attribute), 870
- MIN_OPEN_PERIOD* (*telegram.Poll* attribute), 355
- MIN_OPTION_LENGTH* (*telegram.constants.PollLimit* attribute), 870
- MIN_OPTION_LENGTH* (*telegram.Poll* attribute), 355
- MIN_OPTION_NUMBER* (*telegram.constants.PollLimit* attribute), 870
- MIN_OPTION_NUMBER* (*telegram.Poll* attribute), 355
- MIN_PAYLOAD_LENGTH* (*telegram.constants.InvoiceLimit* attribute), 797
- MIN_PAYLOAD_LENGTH* (*telegram.Invoice* attribute), 482
- MIN_PROXIMITY_ALERT_RADIUS* (*telegram.constants.LocationLimit* attribute), 807
- MIN_PROXIMITY_ALERT_RADIUS* (*telegram.InlineQueryResultLocation* attribute), 455
- MIN_PROXIMITY_ALERT_RADIUS* (*telegram.InputLocationMessageContent* attribute), 474
- MIN_QUANTITY* (*telegram.constants.KeyboardButtonRequestUsersLimit* attribute), 802
- MIN_QUESTION_LENGTH* (*telegram.constants.PollLimit* attribute), 870
- MIN_QUESTION_LENGTH* (*telegram.Poll* attribute), 355
- MIN_SECRET_TOKEN_LENGTH* (*telegram.constants.WebhookLimit* attribute), 941
- MIN_START_PARAMETER_LENGTH* (*telegram.constants.InlineQueryResultsButtonLimit* attribute), 786
- MIN_START_PARAMETER_LENGTH* (*telegram.InlineQueryResultsButton* attribute), 461
- MIN_STICKER_EMOJI* (*telegram.constants.StickerLimit* attribute), 915
- MIN_SWITCH_PM_TEXT_LENGTH* (*telegram.constants.InlineQueryLimit* attribute), 771
- MIN_SWITCH_PM_TEXT_LENGTH* (*telegram.InlineQuery* attribute), 426
- MIN_TEXT_LENGTH* (*telegram.constants.MessageLimit* attribute), 843
- MIN_TITLE_LENGTH* (*telegram.constants.InvoiceLimit* attribute), 797
- MIN_TITLE_LENGTH* (*telegram.Invoice* attribute), 482
- MIN_VALUE* (*telegram.constants.DiceLimit* attribute), 737
- MIN_VALUE* (*telegram.Dice* attribute), 238
- module
 - telegram*, 21
 - telegram.constants*, 651
 - telegram.error*, 945
 - telegram.ext*, 516
 - telegram.ext.filters*, 589
 - telegram.helpers*, 948
 - telegram.warnings*, 957
- MOUTH* (*telegram.constants.MaskPosition* attribute), 812
- MOUTH* (*telegram.MaskPosition* attribute), 417
- MOYAI* (*telegram.constants.ReactionEmoji* attribute), 889
- MP3* (*telegram.ext.filters.Document* attribute), 601
- MP4* (*telegram.ext.filters.Document* attribute), 600
- mpeg4_duration* (*telegram.InlineQueryResultMpeg4Gif* attribute), 457
- mpeg4_file_id* (*telegram.InlineQueryResultCachedMpeg4Gif* attribute), 437
- mpeg4_height* (*telegram.InlineQueryResultMpeg4Gif* attribute), 457
- mpeg4_url* (*telegram.InlineQueryResultMpeg4Gif* attribute), 457
- mpeg4_width* (*telegram.InlineQueryResultMpeg4Gif* attribute), 457
- MPEG4GIF* (*telegram.constants.InlineQueryResultType* attribute), 780
- multipart_data* (*telegram.request.RequestData* property), 954
- MY_CHAT_MEMBER* (*telegram.constants.UpdateType* attribute), 930
- MY_CHAT_MEMBER* (*telegram.ext.ChatMemberHandler* attribute), 581
- MY_CHAT_MEMBER* (*telegram.Update* attribute), 382
- my_chat_member* (*telegram.Update* attribute), 380
- N**
 - NAIL_POLISH* (*telegram.constants.ReactionEmoji* attribute), 889
 - name* (*telegram.Bot* property), 83
 - name* (*telegram.BotName* attribute), 165
 - name* (*telegram.ChatInviteLink* attribute), 213
 - name* (*telegram.ext.ConversationHandler* property), 589
 - name* (*telegram.ext.filters.BaseFilter* property), 593
 - name* (*telegram.ext.filters.Chat* property), 596
 - name* (*telegram.ext.filters.ForwardedFrom* property), 602
 - name* (*telegram.ext.filters.SenderChat* property), 606
 - name* (*telegram.ext.filters.User* property), 612
 - name* (*telegram.ext.filters.ViaBot* property), 614
 - name* (*telegram.ext.Job* attribute), 560
 - name* (*telegram.ForumTopic* attribute), 248
 - name* (*telegram.ForumTopicCreated* attribute), 249
 - name* (*telegram.ForumTopicEdited* attribute), 250
 - name* (*telegram.OrderInfo* attribute), 484
 - name* (*telegram.StickerSet* attribute), 421
 - name* (*telegram.User* property), 392

- NAME_LENGTH (*telegram.constants.ChatInviteLinkLimit* attribute), 697
- need_email (*telegram.InputInvoiceMessageContent* attribute), 480
- need_name (*telegram.InputInvoiceMessageContent* attribute), 480
- need_phone_number (*telegram.InputInvoiceMessageContent* attribute), 480
- need_shipping_address (*telegram.InputInvoiceMessageContent* attribute), 480
- needs_repainting (*telegram.Sticker* attribute), 420
- NERD_FACE (*telegram.constants.ReactionEmoji* attribute), 889
- NetworkError, 946
- NEUTRAL_FACE (*telegram.constants.ReactionEmoji* attribute), 889
- new_chat_id (*telegram.error.ChatMigrated* attribute), 945
- new_chat_member (*telegram.ChatMemberUpdated* attribute), 229
- NEW_CHAT_MEMBERS (*telegram.constants.MessageType* attribute), 855
- NEW_CHAT_MEMBERS (*telegram.ext.filters.StatusUpdate* attribute), 608
- new_chat_members (*telegram.Message* attribute), 303
- NEW_CHAT_PHOTO (*telegram.constants.MessageType* attribute), 855
- NEW_CHAT_PHOTO (*telegram.ext.filters.StatusUpdate* attribute), 608
- new_chat_photo (*telegram.Message* attribute), 304
- NEW_CHAT_TITLE (*telegram.constants.MessageType* attribute), 855
- NEW_CHAT_TITLE (*telegram.ext.filters.StatusUpdate* attribute), 608
- new_chat_title (*telegram.Message* attribute), 303
- NEW_MOON_WITH_FACE (*telegram.constants.ReactionEmoji* attribute), 889
- new_reaction (*telegram.MessageReactionUpdated* attribute), 350
- next_t (*telegram.ext.Job* property), 561
- no_permissions() (*telegram.ChatPermissions* class method), 233
- no_rights() (*telegram.ChatAdministratorRights* class method), 206
- nonce (*telegram.Credentials* attribute), 493
- numerator (*telegram.constants.BotCommandLimit* attribute), 657
- numerator (*telegram.constants.BotDescriptionLimit* attribute), 667
- numerator (*telegram.constants.BotNameLimit* attribute), 671
- numerator (*telegram.constants.BulkRequestLimit* attribute), 675
- numerator (*telegram.constants.CallbackQueryLimit* attribute), 679
- numerator (*telegram.constants.ChatID* attribute), 696
- numerator (*telegram.constants.ChatInviteLinkLimit* attribute), 701
- numerator (*telegram.constants.ChatLimit* attribute), 705
- numerator (*telegram.constants.ChatPhotoSize* attribute), 715
- numerator (*telegram.constants.ContactLimit* attribute), 725
- numerator (*telegram.constants.CustomEmojiStickerLimit* attribute), 729
- numerator (*telegram.constants.DiceLimit* attribute), 740
- numerator (*telegram.constants.FileSizeLimit* attribute), 745
- numerator (*telegram.constants.FloodLimit* attribute), 749
- numerator (*telegram.constants.ForumIconColor* attribute), 753
- numerator (*telegram.constants.ForumTopicLimit* attribute), 758
- numerator (*telegram.constants.GiveawayLimit* attribute), 762
- numerator (*telegram.constants.InlineKeyboardButtonLimit* attribute), 766
- numerator (*telegram.constants.InlineKeyboardMarkupLimit* attribute), 770
- numerator (*telegram.constants.InlineQueryLimit* attribute), 775
- numerator (*telegram.constants.InlineQueryResultLimit* attribute), 779
- numerator (*telegram.constants.InlineQueryResultsButtonLimit* attribute), 790
- numerator (*telegram.constants.InvoiceLimit* attribute), 801
- numerator (*telegram.constants.KeyboardButtonRequestUsersLimit* attribute), 805
- numerator (*telegram.constants.LocationLimit* attribute), 811
- numerator (*telegram.constants.MediaGroupLimit* attribute), 821
- numerator (*telegram.constants.MessageLimit* attribute), 846
- numerator (*telegram.constants.PollingLimit* attribute), 884
- numerator (*telegram.constants.PollLimit* attribute), 874
- numerator (*telegram.constants.ReplyLimit* attribute), 907
- numerator (*telegram.constants.StickerLimit* attribute), 918
- numerator (*telegram.constants.StickerSetLimit* attribute), 923
- numerator (*telegram.constants.UserProfilePhotosLimit* attribute), 940
- numerator (*telegram.constants.WebhookLimit* attribute), 945

O

`offset` (*telegram.InlineQuery* attribute), 425
`offset` (*telegram.MessageEntity* attribute), 340
`OK_HAND_SIGN` (*telegram.constants.ReactionEmoji* attribute), 889
`old_chat_member` (*telegram.ChatMemberUpdated* attribute), 229
`old_reaction` (*telegram.MessageReactionUpdated* attribute), 350
`on_flush` (*telegram.ext.PicklePersistence* attribute), 641
`one_time_keyboard` (*telegram.ReplyKeyboardMarkup* attribute), 364
`only_new_members` (*telegram.Giveaway* attribute), 252
`only_new_members` (*telegram.GiveawayWinners* attribute), 255
`open_period` (*telegram.Poll* attribute), 354
`option_ids` (*telegram.PollAnswer* attribute), 357
`options` (*telegram.Poll* attribute), 353
`order_info` (*telegram.PreCheckoutQuery* attribute), 486
`order_info` (*telegram.SuccessfulPayment* attribute), 490
`OrderInfo` (class in *telegram*), 484
`origin` (*telegram.ExternalReplyInfo* attribute), 241
`OWNER` (*telegram.ChatMember* attribute), 218
`OWNER` (*telegram.constants.ChatMemberStatus* attribute), 706

P

`parameters` (*telegram.request.RequestData* property), 954
`parametrized_url()` (*telegram.request.RequestData* method), 954
`parse_caption_entities()` (*telegram.Message* method), 320
`parse_caption_entity()` (*telegram.Message* method), 320
`parse_entities()` (*telegram.Message* method), 320
`parse_entity()` (*telegram.Message* method), 321
`parse_explanation_entities()` (*telegram.Poll* method), 355
`parse_explanation_entity()` (*telegram.Poll* method), 355
`parse_json_payload()` (*telegram.request.BaseRequest* static method), 952
`parse_mode` (*telegram.ext.Defaults* property), 556
`parse_mode` (*telegram.InlineQueryResultAudio* attribute), 431
`parse_mode` (*telegram.InlineQueryResultCachedAudio* attribute), 432
`parse_mode` (*telegram.InlineQueryResultCachedDocument* attribute), 434
`parse_mode` (*telegram.InlineQueryResultCachedGif* attribute), 436
`parse_mode` (*telegram.InlineQueryResultCachedMpeg4Gif* attribute), 438
`parse_mode` (*telegram.InlineQueryResultCachedPhoto* attribute), 440
`parse_mode` (*telegram.InlineQueryResultCachedVideo* attribute), 442
`parse_mode` (*telegram.InlineQueryResultCachedVoice* attribute), 444
`parse_mode` (*telegram.InlineQueryResultDocument* attribute), 448
`parse_mode` (*telegram.InlineQueryResultGif* attribute), 452
`parse_mode` (*telegram.InlineQueryResultMpeg4Gif* attribute), 457
`parse_mode` (*telegram.InlineQueryResultPhoto* attribute), 460
`parse_mode` (*telegram.InlineQueryResultVideo* attribute), 466
`parse_mode` (*telegram.InlineQueryResultVoice* attribute), 468
`parse_mode` (*telegram.InputMedia* attribute), 266
`parse_mode` (*telegram.InputMediaAnimation* attribute), 268
`parse_mode` (*telegram.InputMediaAudio* attribute), 270
`parse_mode` (*telegram.InputMediaDocument* attribute), 273
`parse_mode` (*telegram.InputMediaPhoto* attribute), 274
`parse_mode` (*telegram.InputMediaVideo* attribute), 276
`parse_mode` (*telegram.InputTextMessageContent* attribute), 470
`parse_text_entities()` (*telegram.Game* method), 492
`parse_text_entity()` (*telegram.Game* method), 492
`ParseMode` (class in *telegram.constants*), 863
`partition()` (*telegram.constants.BotCommandScopeType* method), 661
`partition()` (*telegram.constants.ChatAction* method), 684
`partition()` (*telegram.constants.ChatBoostSources* method), 690
`partition()` (*telegram.constants.ChatMemberStatus* method), 709
`partition()` (*telegram.constants.ChatType* method), 720
`partition()` (*telegram.constants.DiceEmoji* method), 734
`partition()` (*telegram.constants.InlineQueryResultType* method), 784
`partition()` (*telegram.constants.InputMediaType* method), 794
`partition()` (*telegram.constants.MaskPosition* method), 815
`partition()` (*telegram.constants.MenuButtonType* method), 825
`partition()` (*telegram.constants.MessageAttachmentType*

- method*), 832
- `partition()` (*telegram.constants.MessageEntityType method*), 840
- `partition()` (*telegram.constants.MessageOriginType method*), 851
- `partition()` (*telegram.constants.MessageType method*), 861
- `partition()` (*telegram.constants.ParseMode method*), 867
- `partition()` (*telegram.constants.PollType method*), 878
- `partition()` (*telegram.constants.ReactionEmoji method*), 896
- `partition()` (*telegram.constants.ReactionType method*), 902
- `partition()` (*telegram.constants.StickerFormat method*), 912
- `partition()` (*telegram.constants.StickerType method*), 927
- `partition()` (*telegram.constants.UpdateType method*), 935
- `PARTY_POPPER` (*telegram.constants.ReactionEmoji attribute*), 889
- `passport` (*telegram.SecureData attribute*), 513
- `PASSPORT_DATA` (in module *telegram.ext.filters*), 591
- `PASSPORT_DATA` (*telegram.constants.MessageAttachmentType attribute*), 828
- `PASSPORT_DATA` (*telegram.constants.MessageType attribute*), 856
- `passport_data` (*telegram.Message attribute*), 305
- `passport_registration` (*telegram.SecureData attribute*), 514
- `PassportData` (class in *telegram*), 499
- `PassportDecryptionError`, 947
- `PassportElementError` (class in *telegram*), 501
- `PassportElementErrorDataField` (class in *telegram*), 501
- `PassportElementErrorFile` (class in *telegram*), 502
- `PassportElementErrorFiles` (class in *telegram*), 503
- `PassportElementErrorFrontSide` (class in *telegram*), 504
- `PassportElementErrorReverseSide` (class in *telegram*), 504
- `PassportElementErrorSelfie` (class in *telegram*), 505
- `PassportElementErrorTranslationFile` (class in *telegram*), 506
- `PassportElementErrorTranslationFiles` (class in *telegram*), 507
- `PassportElementErrorUnspecified` (class in *telegram*), 507
- `PassportFile` (class in *telegram*), 508
- `pattern` (*telegram.ext.CallbackQueryHandler attribute*), 577
- `pattern` (*telegram.ext.ChosenInlineResultHandler attribute*), 582
- `pattern` (*telegram.ext.InlineQueryHandler attribute*), 615
- `pattern` (*telegram.ext.PreCheckoutQueryHandler attribute*), 622
- `pattern` (*telegram.ext.StringRegexHandler attribute*), 628
- `pay` (*telegram.InlineKeyboardButton attribute*), 259
- `payload` (*telegram.InputInvoiceMessageContent attribute*), 479
- `PDF` (*telegram.ext.filters.Document attribute*), 601
- `pending_join_request_count` (*telegram.ChatInviteLink attribute*), 213
- `pending_update_count` (*telegram.WebhookInfo attribute*), 414
- `per_chat` (*telegram.ext.ConversationHandler property*), 589
- `per_message` (*telegram.ext.ConversationHandler property*), 589
- `per_user` (*telegram.ext.ConversationHandler property*), 589
- `performer` (*telegram.Audio attribute*), 158
- `performer` (*telegram.InlineQueryResultAudio attribute*), 430
- `performer` (*telegram.InputMediaAudio attribute*), 271
- `permissions` (*telegram.Chat attribute*), 177
- `persistence` (*telegram.ext.Application attribute*), 519
- `persistence()` (*telegram.ext.ApplicationBuilder method*), 539
- `persistence_data` (*telegram.ext.CallbackDataCache property*), 645
- `PersistenceInput` (class in *telegram.ext*), 639
- `persistent` (*telegram.ext.ConversationHandler property*), 589
- `PERSON_WITH_FOLDED_HANDS` (*telegram.constants.ReactionEmoji attribute*), 889
- `personal_details` (*telegram.SecureData attribute*), 513
- `PersonalDetails` (class in *telegram*), 510
- `PHONE_NUMBER` (*telegram.constants.MessageEntityType attribute*), 836
- `phone_number` (*telegram.Contact attribute*), 235
- `phone_number` (*telegram.EncryptedPassportElement attribute*), 497
- `phone_number` (*telegram.InlineQueryResultContact attribute*), 446
- `phone_number` (*telegram.InputContactMessageContent attribute*), 476
- `PHONE_NUMBER` (*telegram.MessageEntity attribute*), 343
- `phone_number` (*telegram.OrderInfo attribute*), 484
- `PHOTO` (in module *telegram.ext.filters*), 591
- `photo` (*telegram.Chat attribute*), 177
- `PHOTO` (*telegram.constants.InlineQueryResultType attribute*), 780

- PHOTO (*telegram.constants.InputMediaType* attribute), 790
- PHOTO (*telegram.constants.MessageAttachmentType* attribute), 828
- PHOTO (*telegram.constants.MessageType* attribute), 856
- photo (*telegram.ExternalReplyInfo* attribute), 241
- photo (*telegram.Game* attribute), 491
- photo (*telegram.Message* attribute), 302
- photo_file_id (*telegram.InlineQueryResultCachedPhoto* attribute), 439
- photo_height (*telegram.InlineQueryResultPhoto* attribute), 460
- photo_height (*telegram.InputInvoiceMessageContent* attribute), 480
- photo_size (*telegram.InputInvoiceMessageContent* attribute), 480
- photo_url (*telegram.InlineQueryResultPhoto* attribute), 459
- photo_url (*telegram.InputInvoiceMessageContent* attribute), 480
- photo_width (*telegram.InlineQueryResultPhoto* attribute), 459
- photo_width (*telegram.InputInvoiceMessageContent* attribute), 480
- photos (*telegram.UserProfilePhotos* attribute), 402
- PhotoSize (class in *telegram*), 350
- PHOTOSIZE_UPLOAD (*telegram.constants.FileSizeLimit* attribute), 741
- PicklePersistence (class in *telegram.ext*), 640
- PILE_OF_POO (*telegram.constants.ReactionEmoji* attribute), 890
- PILL (*telegram.constants.ReactionEmoji* attribute), 890
- pin() (*telegram.Message* method), 321
- pin_chat_message() (*telegram.Bot* method), 83
- pin_message() (*telegram.CallbackQuery* method), 171
- pin_message() (*telegram.Chat* method), 190
- pin_message() (*telegram.User* method), 392
- pinChatMessage() (*telegram.Bot* method), 83
- PINK (*telegram.constants.ForumIconColor* attribute), 750
- pinned_message (*telegram.Chat* attribute), 177
- PINNED_MESSAGE (*telegram.constants.MessageType* attribute), 856
- PINNED_MESSAGE (*telegram.ext.filters.StatusUpdate* attribute), 608
- pinned_message (*telegram.Message* attribute), 304
- point (*telegram.MaskPosition* attribute), 417
- Poll (class in *telegram*), 352
- POLL (in module *telegram.ext.filters*), 591
- POLL (*telegram.constants.MessageAttachmentType* attribute), 828
- POLL (*telegram.constants.MessageType* attribute), 856
- POLL (*telegram.constants.UpdateType* attribute), 931
- poll (*telegram.ExternalReplyInfo* attribute), 243
- poll (*telegram.Message* attribute), 305
- POLL (*telegram.Update* attribute), 382
- poll (*telegram.Update* attribute), 379
- POLL_ANSWER (*telegram.constants.UpdateType* attribute), 931
- POLL_ANSWER (*telegram.Update* attribute), 382
- poll_answer (*telegram.Update* attribute), 380
- poll_id (*telegram.PollAnswer* attribute), 356
- PollAnswer (class in *telegram*), 356
- PollAnswerHandler (class in *telegram.ext*), 620
- PollHandler (class in *telegram.ext*), 621
- PollingLimit (class in *telegram.constants*), 880
- PollLimit (class in *telegram.constants*), 869
- PollOption (class in *telegram*), 357
- PollType (class in *telegram.constants*), 874
- pool_timeout() (*telegram.ext.ApplicationBuilder* method), 540
- position (*telegram.GameHighScore* attribute), 493
- position (*telegram.TextQuote* attribute), 376
- post() (*telegram.request.BaseRequest* method), 952
- post_code (*telegram.ResidentialAddress* attribute), 512
- post_code (*telegram.ShippingAddress* attribute), 487
- post_init (*telegram.ext.Application* attribute), 519
- post_init() (*telegram.ext.ApplicationBuilder* method), 540
- post_shutdown (*telegram.ext.Application* attribute), 519
- post_shutdown() (*telegram.ext.ApplicationBuilder* method), 541
- post_stop (*telegram.ext.Application* attribute), 519
- post_stop() (*telegram.ext.ApplicationBuilder* method), 542
- POUTING_FACE (*telegram.constants.ReactionEmoji* attribute), 890
- PRE (*telegram.constants.MessageEntityType* attribute), 836
- PRE (*telegram.MessageEntity* attribute), 343
- PRE_CHECKOUT_QUERY (*telegram.constants.UpdateType* attribute), 931
- PRE_CHECKOUT_QUERY (*telegram.Update* attribute), 382
- pre_checkout_query (*telegram.Update* attribute), 379
- PreCheckoutQuery (class in *telegram*), 485
- PreCheckoutQueryHandler (class in *telegram.ext*), 622
- prefer_large_media (*telegram.LinkPreviewOptions* attribute), 285
- prefer_small_media (*telegram.LinkPreviewOptions* attribute), 285
- PrefixHandler (class in *telegram.ext*), 623
- PREMIUM (*telegram.ChatBoostSource* attribute), 208
- PREMIUM (*telegram.constants.ChatBoostSources* attribute), 686
- PREMIUM (*telegram.ext.filters.Sticker* attribute), 609
- premium_animation (*telegram.Sticker* attribute), 420

- premium_subscription_month_count (telegram.Giveaway attribute), 252
- premium_subscription_month_count (telegram.GiveawayWinners attribute), 255
- PREMIUM_USER (in module telegram.ext.filters), 591
- prices (telegram.InputInvoiceMessageContent attribute), 479
- prices (telegram.ShippingOption attribute), 488
- PRIVATE (telegram.Chat attribute), 180
- PRIVATE (telegram.constants.ChatType attribute), 716
- PRIVATE (telegram.ext.filters.ChatType attribute), 596
- private_key (telegram.Bot property), 84
- private_key() (telegram.ext.ApplicationBuilder method), 542
- prize_description (telegram.Giveaway attribute), 252
- prize_description (telegram.GiveawayWinners attribute), 255
- process_callback_query() (telegram.ext.CallbackDataCache method), 645
- process_error() (telegram.ext.Application method), 524
- process_keyboard() (telegram.ext.CallbackDataCache method), 646
- process_message() (telegram.ext.CallbackDataCache method), 646
- process_request() (telegram.ext.AIORateLimiter method), 650
- process_request() (telegram.ext.BaseRateLimiter method), 648
- process_update() (telegram.ext.Application method), 524
- process_update() (telegram.ext.BaseUpdateProcessor method), 547
- profile_accent_color_id (telegram.Chat attribute), 179
- profile_background_custom_emoji_id (telegram.Chat attribute), 180
- ProfileAccentColor (class in telegram.constants), 884
- promote_chat_member() (telegram.Bot method), 84
- promote_member() (telegram.Chat method), 191
- promoteChatMember() (telegram.Bot method), 84
- protect_content (telegram.ext.Defaults property), 556
- provider_data (telegram.InputInvoiceMessageContent attribute), 480
- provider_payment_charge_id (telegram.SuccessfulPayment attribute), 490
- provider_token (telegram.InputInvoiceMessageContent attribute), 479
- proximity_alert_radius (telegram.InlineQueryResultLocation attribute), 454
- proximity_alert_radius (telegram.InputLocationMessageContent attribute), 472
- proximity_alert_radius (telegram.Location attribute), 286
- PROXIMITY_ALERT_TRIGGERED (telegram.constants.MessageType attribute), 856
- PROXIMITY_ALERT_TRIGGERED (telegram.ext.filters.StatusUpdate attribute), 608
- proximity_alert_triggered (telegram.Message attribute), 305
- ProximityAlertTriggered (class in telegram), 358
- proxy() (telegram.ext.ApplicationBuilder method), 543
- proxy_url() (telegram.ext.ApplicationBuilder method), 543
- PTBDeprecationWarning, 957
- PTBRuntimeWarning, 957
- PTBUserWarning, 957
- PURPLE (telegram.constants.ForumIconColor attribute), 750
- PY (telegram.ext.filters.Document attribute), 601
- ## Q
- query (telegram.ChosenInlineResult attribute), 423
- query (telegram.InlineQuery attribute), 425
- query (telegram.SwitchInlineQueryChosenChat attribute), 371
- question (telegram.Poll attribute), 353
- QUIZ (telegram.constants.PollType attribute), 874
- QUIZ (telegram.Poll attribute), 355
- quote (telegram.ext.Defaults property), 556
- quote (telegram.Message attribute), 308
- quote (telegram.ReplyParameters attribute), 369
- quote_entities (telegram.ReplyParameters attribute), 370
- quote_parse_mode (telegram.ext.Defaults property), 556
- quote_parse_mode (telegram.ReplyParameters attribute), 370
- quote_position (telegram.ReplyParameters attribute), 370
- ## R
- rate_limiter (telegram.ext.ExtBot property), 558
- rate_limiter() (telegram.ext.ApplicationBuilder method), 543
- ReactionCount (class in telegram), 359
- ReactionEmoji (class in telegram.constants), 885
- reactions (telegram.MessageReactionCountUpdated attribute), 349
- ReactionType (class in telegram), 359
- ReactionType (class in telegram.constants), 898
- ReactionTypeCustomEmoji (class in telegram), 360

- `ReactionTypeEmoji` (class in `telegram`), 361
- `read_timeout` (`telegram.request.BaseRequest` property), 953
- `read_timeout` (`telegram.request.HTTPXRequest` property), 956
- `read_timeout()` (`telegram.ext.ApplicationBuilder` method), 544
- `real` (`telegram.constants.BotCommandLimit` attribute), 657
- `real` (`telegram.constants.BotDescriptionLimit` attribute), 667
- `real` (`telegram.constants.BotNameLimit` attribute), 671
- `real` (`telegram.constants.BulkRequestLimit` attribute), 675
- `real` (`telegram.constants.CallbackQueryLimit` attribute), 679
- `real` (`telegram.constants.ChatID` attribute), 696
- `real` (`telegram.constants.ChatInviteLinkLimit` attribute), 701
- `real` (`telegram.constants.ChatLimit` attribute), 705
- `real` (`telegram.constants.ChatPhotoSize` attribute), 715
- `real` (`telegram.constants.ContactLimit` attribute), 725
- `real` (`telegram.constants.CustomEmojiStickerLimit` attribute), 729
- `real` (`telegram.constants.DiceLimit` attribute), 740
- `real` (`telegram.constants.FileSizeLimit` attribute), 745
- `real` (`telegram.constants.FloodLimit` attribute), 749
- `real` (`telegram.constants.ForumIconColor` attribute), 753
- `real` (`telegram.constants.ForumTopicLimit` attribute), 758
- `real` (`telegram.constants.GiveawayLimit` attribute), 762
- `real` (`telegram.constants.InlineKeyboardButtonLimit` attribute), 766
- `real` (`telegram.constants.InlineKeyboardMarkupLimit` attribute), 770
- `real` (`telegram.constants.InlineQueryLimit` attribute), 775
- `real` (`telegram.constants.InlineQueryResultLimit` attribute), 779
- `real` (`telegram.constants.InlineQueryResultsButtonLimit` attribute), 790
- `real` (`telegram.constants.InvoiceLimit` attribute), 801
- `real` (`telegram.constants.KeyboardButtonRequestUsersLimit` attribute), 805
- `real` (`telegram.constants.LocationLimit` attribute), 811
- `real` (`telegram.constants.MediaGroupLimit` attribute), 821
- `real` (`telegram.constants.MessageLimit` attribute), 846
- `real` (`telegram.constants.PollingLimit` attribute), 884
- `real` (`telegram.constants.PollLimit` attribute), 874
- `real` (`telegram.constants.ReplyLimit` attribute), 907
- `real` (`telegram.constants.StickerLimit` attribute), 918
- `real` (`telegram.constants.StickerSetLimit` attribute), 923
- `real` (`telegram.constants.UserProfilePhotosLimit` attribute), 940
- `real` (`telegram.constants.WebhookLimit` attribute), 945
- `RECORD_VIDEO` (`telegram.constants.ChatAction` attribute), 680
- `RECORD_VIDEO_NOTE` (`telegram.constants.ChatAction` attribute), 680
- `RECORD_VOICE` (`telegram.constants.ChatAction` attribute), 680
- `RED` (`telegram.constants.ForumIconColor` attribute), 750
- `RED_HEART` (`telegram.constants.ReactionEmoji` attribute), 890
- `refresh_bot_data()` (`telegram.ext.BasePersistence` method), 633
- `refresh_bot_data()` (`telegram.ext.DictPersistence` method), 638
- `refresh_bot_data()` (`telegram.ext.PicklePersistence` method), 642
- `refresh_chat_data()` (`telegram.ext.BasePersistence` method), 633
- `refresh_chat_data()` (`telegram.ext.DictPersistence` method), 638
- `refresh_chat_data()` (`telegram.ext.PicklePersistence` method), 643
- `refresh_data()` (`telegram.ext.CallbackContext` method), 551
- `refresh_user_data()` (`telegram.ext.BasePersistence` method), 633
- `refresh_user_data()` (`telegram.ext.DictPersistence` method), 638
- `refresh_user_data()` (`telegram.ext.PicklePersistence` method), 643
- `Regex` (class in `telegram.ext.filters`), 604
- `REGULAR` (`telegram.constants.PollType` attribute), 874
- `REGULAR` (`telegram.constants.StickerType` attribute), 924
- `REGULAR` (`telegram.Poll` attribute), 355
- `REGULAR` (`telegram.Sticker` attribute), 420
- `remove_bot_ids()` (`telegram.ext.filters.ViaBot` method), 614
- `remove_chat_ids()` (`telegram.ext.filters.Chat` method), 595
- `remove_chat_ids()` (`telegram.ext.filters.ForwardedFrom` method), 602
- `remove_chat_ids()` (`telegram.ext.filters.SenderChat` method), 606
- `remove_date` (`telegram.ChatBoostRemoved` attribute), 207
- `remove_error_handler()` (`telegram.ext.Application` method), 525
- `remove_handler()` (`telegram.ext.Application` method), 525
- `remove_keyboard` (`telegram.ReplyKeyboardRemove` attribute), 367
- `remove_user_ids()` (`telegram.ext.filters.User` method), 613
- `remove_usernames()` (`telegram.ext.filters.Chat` method), 596

`remove_usernames()` (*telegram.ext.filters.ForwardedFrom* method), 603
`remove_usernames()` (*telegram.ext.filters.SenderChat* method), 606
`remove_usernames()` (*telegram.ext.filters.User* method), 612
`remove_usernames()` (*telegram.ext.filters.ViaBot* method), 614
`removed` (*telegram.ext.Job* property), 561
`REMOVED_CHAT_BOOST` (*telegram.constants.UpdateType* attribute), 931
`REMOVED_CHAT_BOOST` (*telegram.ext.ChatBoostHandler* attribute), 578
`REMOVED_CHAT_BOOST` (*telegram.Update* attribute), 382
`removed_chat_boost` (*telegram.Update* attribute), 380
`removeprefix()` (*telegram.constants.BotCommandScopeType* method), 661
`removeprefix()` (*telegram.constants.ChatAction* method), 684
`removeprefix()` (*telegram.constants.ChatBoostSources* method), 690
`removeprefix()` (*telegram.constants.ChatMemberStatus* method), 710
`removeprefix()` (*telegram.constants.ChatType* method), 720
`removeprefix()` (*telegram.constants.DiceEmoji* method), 734
`removeprefix()` (*telegram.constants.InlineQueryResultType* method), 784
`removeprefix()` (*telegram.constants.InputMediaType* method), 794
`removeprefix()` (*telegram.constants.MaskPosition* method), 815
`removeprefix()` (*telegram.constants.MenuButtonType* method), 825
`removeprefix()` (*telegram.constants.MessageAttachmentType* method), 833
`removeprefix()` (*telegram.constants.MessageEntityType* method), 840
`removeprefix()` (*telegram.constants.MessageOriginType* method), 851
`removeprefix()` (*telegram.constants.MessageType* method), 861
`removeprefix()` (*telegram.constants.ParseMode* method), 867
`removeprefix()` (*telegram.constants.PollType* method), 878
`removeprefix()` (*telegram.constants.ReactionEmoji* method), 896
`removeprefix()` (*telegram.constants.ReactionType* method), 902
`removeprefix()` (*telegram.constants.StickerFormat* method), 912
`removeprefix()` (*telegram.constants.StickerType* method), 927
`removeprefix()` (*telegram.constants.UpdateType* method), 935
`removesuffix()` (*telegram.constants.BotCommandScopeType* method), 661
`removesuffix()` (*telegram.constants.ChatAction* method), 684
`removesuffix()` (*telegram.constants.ChatBoostSources* method), 690
`removesuffix()` (*telegram.constants.ChatMemberStatus* method), 710
`removesuffix()` (*telegram.constants.ChatType* method), 720
`removesuffix()` (*telegram.constants.DiceEmoji* method), 734
`removesuffix()` (*telegram.constants.InlineQueryResultType* method), 784
`removesuffix()` (*telegram.constants.InputMediaType* method), 794
`removesuffix()` (*telegram.constants.MaskPosition* method), 815
`removesuffix()` (*telegram.constants.MenuButtonType* method), 825
`removesuffix()` (*telegram.constants.MessageAttachmentType* method), 833
`removesuffix()` (*telegram.constants.MessageEntityType* method), 840
`removesuffix()` (*telegram.constants.MessageOriginType* method), 851
`removesuffix()` (*telegram.constants.MessageType* method), 861
`removesuffix()` (*telegram.constants.ParseMode* method), 867
`removesuffix()` (*telegram.constants.PollType* method), 878
`removesuffix()` (*telegram.constants.ReactionEmoji* method), 896
`removesuffix()` (*telegram.constants.ReactionType* method), 902

- `removesuffix()` (*telegram.constants.StickerFormat method*), 912
- `removesuffix()` (*telegram.constants.StickerType method*), 927
- `removesuffix()` (*telegram.constants.UpdateType method*), 935
- `rental_agreement` (*telegram.SecureData attribute*), 514
- `reopen_forum_topic()` (*telegram.Bot method*), 86
- `reopen_forum_topic()` (*telegram.Chat method*), 191
- `reopen_forum_topic()` (*telegram.Message method*), 321
- `reopen_general_forum_topic()` (*telegram.Bot method*), 87
- `reopen_general_forum_topic()` (*telegram.Chat method*), 191
- `reopenForumTopic()` (*telegram.Bot method*), 86
- `reopenGeneralForumTopic()` (*telegram.Bot method*), 86
- `replace()` (*telegram.constants.BotCommandScopeType method*), 662
- `replace()` (*telegram.constants.ChatAction method*), 684
- `replace()` (*telegram.constants.ChatBoostSources method*), 690
- `replace()` (*telegram.constants.ChatMemberStatus method*), 710
- `replace()` (*telegram.constants.ChatType method*), 720
- `replace()` (*telegram.constants.DiceEmoji method*), 734
- `replace()` (*telegram.constants.InlineQueryResultType method*), 784
- `replace()` (*telegram.constants.InputMediaType method*), 794
- `replace()` (*telegram.constants.MaskPosition method*), 815
- `replace()` (*telegram.constants.MenuButtonType method*), 825
- `replace()` (*telegram.constants.MessageAttachmentType method*), 833
- `replace()` (*telegram.constants.MessageEntityType method*), 840
- `replace()` (*telegram.constants.MessageOriginType method*), 851
- `replace()` (*telegram.constants.MessageType method*), 861
- `replace()` (*telegram.constants.ParseMode method*), 867
- `replace()` (*telegram.constants.PollType method*), 878
- `replace()` (*telegram.constants.ReactionEmoji method*), 896
- `replace()` (*telegram.constants.ReactionType method*), 902
- `replace()` (*telegram.constants.StickerFormat method*), 912
- `replace()` (*telegram.constants.StickerType method*), 927
- `replace()` (*telegram.constants.UpdateType method*), 935
- `REPLY` (in module *telegram.ext.filters*), 591
- `reply_animation()` (*telegram.Message method*), 322
- `reply_audio()` (*telegram.Message method*), 322
- `reply_chat_action()` (*telegram.Message method*), 323
- `reply_contact()` (*telegram.Message method*), 323
- `reply_copy()` (*telegram.Message method*), 323
- `reply_dice()` (*telegram.Message method*), 324
- `reply_document()` (*telegram.Message method*), 325
- `reply_game()` (*telegram.Message method*), 325
- `reply_html()` (*telegram.Message method*), 326
- `reply_invoice()` (*telegram.Message method*), 326
- `reply_location()` (*telegram.Message method*), 327
- `reply_markdown()` (*telegram.Message method*), 328
- `reply_markdown_v2()` (*telegram.Message method*), 329
- `reply_markup` (*telegram.InlineQueryResultArticle attribute*), 428
- `reply_markup` (*telegram.InlineQueryResultAudio attribute*), 431
- `reply_markup` (*telegram.InlineQueryResultCachedAudio attribute*), 432
- `reply_markup` (*telegram.InlineQueryResultCachedDocument attribute*), 434
- `reply_markup` (*telegram.InlineQueryResultCachedGif attribute*), 436
- `reply_markup` (*telegram.InlineQueryResultCachedMpeg4Gif attribute*), 438
- `reply_markup` (*telegram.InlineQueryResultCachedPhoto attribute*), 440
- `reply_markup` (*telegram.InlineQueryResultCachedSticker attribute*), 441
- `reply_markup` (*telegram.InlineQueryResultCachedVideo attribute*), 443
- `reply_markup` (*telegram.InlineQueryResultCachedVoice attribute*), 444
- `reply_markup` (*telegram.InlineQueryResultContact attribute*), 446
- `reply_markup` (*telegram.InlineQueryResultDocument attribute*), 448
- `reply_markup` (*telegram.InlineQueryResultGame attribute*), 450
- `reply_markup` (*telegram.InlineQueryResultGif attribute*), 452
- `reply_markup` (*telegram.InlineQueryResultLocation attribute*), 454
- `reply_markup` (*telegram.InlineQueryResultMpeg4Gif attribute*), 458

- `reply_markup` (*telegram.InlineQueryResultPhoto* attribute), 460
- `reply_markup` (*telegram.InlineQueryResultVenue* attribute), 464
- `reply_markup` (*telegram.InlineQueryResultVideo* attribute), 467
- `reply_markup` (*telegram.InlineQueryResultVoice* attribute), 469
- `reply_markup` (*telegram.Message* attribute), 306
- `reply_media_group()` (*telegram.Message* method), 329
- `reply_photo()` (*telegram.Message* method), 330
- `reply_poll()` (*telegram.Message* method), 330
- `reply_sticker()` (*telegram.Message* method), 331
- `reply_text()` (*telegram.Message* method), 331
- `reply_to_message` (*telegram.Message* attribute), 300
- `reply_venue()` (*telegram.Message* method), 332
- `reply_video()` (*telegram.Message* method), 332
- `reply_video_note()` (*telegram.Message* method), 333
- `reply_voice()` (*telegram.Message* method), 334
- `ReplyKeyboardMarkup` (class in *telegram*), 361
- `ReplyKeyboardRemove` (class in *telegram*), 367
- `ReplyLimit` (class in *telegram.constants*), 904
- `ReplyParameters` (class in *telegram*), 368
- `request` (*telegram.Bot* property), 87
- `request()` (*telegram.ext.ApplicationBuilder* method), 544
- `request_chat` (*telegram.KeyboardButton* attribute), 280
- `request_contact` (*telegram.KeyboardButton* attribute), 279
- `request_id` (*telegram.ChatShared* attribute), 235
- `request_id` (*telegram.KeyboardButtonRequestChat* attribute), 282
- `request_id` (*telegram.KeyboardButtonRequestUsers* attribute), 284
- `request_id` (*telegram.UsersShared* attribute), 403
- `request_location` (*telegram.KeyboardButton* attribute), 280
- `request_poll` (*telegram.KeyboardButton* attribute), 280
- `request_user` (*telegram.KeyboardButton* property), 280
- `request_users` (*telegram.KeyboardButton* attribute), 280
- `request_write_access` (*telegram.LoginUrl* attribute), 288
- `RequestData` (class in *telegram.request*), 954
- `residence_country_code` (*telegram.PersonalDetails* attribute), 511
- `ResidentialAddress` (class in *telegram*), 512
- `resize_keyboard` (*telegram.ReplyKeyboardMarkup* attribute), 364
- `restrict_chat_member()` (*telegram.Bot* method), 88
- `restrict_member()` (*telegram.Chat* method), 192
- `restrictChatMember()` (*telegram.Bot* method), 87
- `RESTRICTED` (*telegram.ChatMember* attribute), 218
- `RESTRICTED` (*telegram.constants.ChatMemberStatus* attribute), 706
- `result_id` (*telegram.ChosenInlineResult* attribute), 423
- `RESULTS` (*telegram.constants.InlineQueryLimit* attribute), 771
- `retrieve()` (*telegram.request.BaseRequest* method), 953
- `retry_after` (*telegram.error.RetryAfter* attribute), 947
- `RetryAfter`, 947
- `reverse_side` (*telegram.EncryptedPassportElement* attribute), 497
- `reverse_side` (*telegram.SecureValue* attribute), 515
- `REVERSED_HAND_WITH_MIDDLE_FINGER_EXTENDED` (*telegram.constants.ReactionEmoji* attribute), 890
- `revoke_chat_invite_link()` (*telegram.Bot* method), 89
- `revoke_invite_link()` (*telegram.Chat* method), 192
- `revokeChatInviteLink()` (*telegram.Bot* method), 89
- `rfind()` (*telegram.constants.BotCommandScopeType* method), 662
- `rfind()` (*telegram.constants.ChatAction* method), 685
- `rfind()` (*telegram.constants.ChatBoostSources* method), 691
- `rfind()` (*telegram.constants.ChatMemberStatus* method), 710
- `rfind()` (*telegram.constants.ChatType* method), 720
- `rfind()` (*telegram.constants.DiceEmoji* method), 734
- `rfind()` (*telegram.constants.InlineQueryResultType* method), 784
- `rfind()` (*telegram.constants.InputMediaType* method), 795
- `rfind()` (*telegram.constants.MaskPosition* method), 816
- `rfind()` (*telegram.constants.MenuButtonType* method), 826
- `rfind()` (*telegram.constants.MessageAttachmentType* method), 833
- `rfind()` (*telegram.constants.MessageEntityType* method), 840
- `rfind()` (*telegram.constants.MessageOriginType* method), 851
- `rfind()` (*telegram.constants.MessageType* method), 862
- `rfind()` (*telegram.constants.ParseMode* method), 868
- `rfind()` (*telegram.constants.PollType* method), 878
- `rfind()` (*telegram.constants.ReactionEmoji* method), 896
- `rfind()` (*telegram.constants.ReactionType* method), 902
- `rfind()` (*telegram.constants.StickerFormat* method), 912
- `rfind()` (*telegram.constants.StickerType* method), 928
- `rfind()` (*telegram.constants.UpdateType* method), 935
- `rindex()` (*telegram.constants.BotCommandScopeType* method), 662

`rindex()` (*telegram.constants.ChatAction* method), 685
`rindex()` (*telegram.constants.ChatBoostSources* method), 691
`rindex()` (*telegram.constants.ChatMemberStatus* method), 710
`rindex()` (*telegram.constants.ChatType* method), 720
`rindex()` (*telegram.constants.DiceEmoji* method), 735
`rindex()` (*telegram.constants.InlineQueryResultType* method), 785
`rindex()` (*telegram.constants.InputMediaType* method), 795
`rindex()` (*telegram.constants.MaskPosition* method), 816
`rindex()` (*telegram.constants.MenuButtonType* method), 826
`rindex()` (*telegram.constants.MessageAttachmentType* method), 833
`rindex()` (*telegram.constants.MessageEntityType* method), 840
`rindex()` (*telegram.constants.MessageOriginType* method), 851
`rindex()` (*telegram.constants.MessageType* method), 862
`rindex()` (*telegram.constants.ParseMode* method), 868
`rindex()` (*telegram.constants.PollType* method), 878
`rindex()` (*telegram.constants.ReactionEmoji* method), 896
`rindex()` (*telegram.constants.ReactionType* method), 902
`rindex()` (*telegram.constants.StickerFormat* method), 912
`rindex()` (*telegram.constants.StickerType* method), 928
`rindex()` (*telegram.constants.UpdateType* method), 935
`rjust()` (*telegram.constants.BotCommandScopeType* method), 662
`rjust()` (*telegram.constants.ChatAction* method), 685
`rjust()` (*telegram.constants.ChatBoostSources* method), 691
`rjust()` (*telegram.constants.ChatMemberStatus* method), 710
`rjust()` (*telegram.constants.ChatType* method), 720
`rjust()` (*telegram.constants.DiceEmoji* method), 735
`rjust()` (*telegram.constants.InlineQueryResultType* method), 785
`rjust()` (*telegram.constants.InputMediaType* method), 795
`rjust()` (*telegram.constants.MaskPosition* method), 816
`rjust()` (*telegram.constants.MenuButtonType* method), 826
`rjust()` (*telegram.constants.MessageAttachmentType* method), 833
`rjust()` (*telegram.constants.MessageEntityType* method), 840
`rjust()` (*telegram.constants.MessageOriginType* method), 851
`rjust()` (*telegram.constants.MessageType* method), 862
`rjust()` (*telegram.constants.ParseMode* method), 868
`rjust()` (*telegram.constants.PollType* method), 878
`rjust()` (*telegram.constants.ReactionEmoji* method), 896
`rjust()` (*telegram.constants.ReactionType* method), 902
`rjust()` (*telegram.constants.StickerFormat* method), 912
`rjust()` (*telegram.constants.StickerType* method), 928
`rjust()` (*telegram.constants.UpdateType* method), 935
`ROLLING_ON_THE_FLOOR_LAUGHING` (*telegram.constants.ReactionEmoji* attribute), 890
`rpartition()` (*telegram.constants.BotCommandScopeType* method), 662
`rpartition()` (*telegram.constants.ChatAction* method), 685
`rpartition()` (*telegram.constants.ChatBoostSources* method), 691
`rpartition()` (*telegram.constants.ChatMemberStatus* method), 710
`rpartition()` (*telegram.constants.ChatType* method), 720
`rpartition()` (*telegram.constants.DiceEmoji* method), 735
`rpartition()` (*telegram.constants.InlineQueryResultType* method), 785
`rpartition()` (*telegram.constants.InputMediaType* method), 795
`rpartition()` (*telegram.constants.MaskPosition* method), 816
`rpartition()` (*telegram.constants.MenuButtonType* method), 826
`rpartition()` (*telegram.constants.MessageAttachmentType* method), 833
`rpartition()` (*telegram.constants.MessageEntityType* method), 841
`rpartition()` (*telegram.constants.MessageOriginType* method), 851
`rpartition()` (*telegram.constants.MessageType* method), 862
`rpartition()` (*telegram.constants.ParseMode* method), 868
`rpartition()` (*telegram.constants.PollType* method), 879
`rpartition()` (*telegram.constants.ReactionEmoji* method), 897
`rpartition()` (*telegram.constants.ReactionType*

- `method`), 902
 - `rpartition()` (*telegram.constants.StickerFormat method*), 912
 - `rpartition()` (*telegram.constants.StickerType method*), 928
 - `rpartition()` (*telegram.constants.UpdateType method*), 935
 - `rsplit()` (*telegram.constants.BotCommandScopeType method*), 662
 - `rsplit()` (*telegram.constants.ChatAction method*), 685
 - `rsplit()` (*telegram.constants.ChatBoostSources method*), 691
 - `rsplit()` (*telegram.constants.ChatMemberStatus method*), 710
 - `rsplit()` (*telegram.constants.ChatType method*), 720
 - `rsplit()` (*telegram.constants.DiceEmoji method*), 735
 - `rsplit()` (*telegram.constants.InlineQueryResultType method*), 785
 - `rsplit()` (*telegram.constants.InputMediaType method*), 795
 - `rsplit()` (*telegram.constants.MaskPosition method*), 816
 - `rsplit()` (*telegram.constants.MenuButtonType method*), 826
 - `rsplit()` (*telegram.constants.MessageAttachmentType method*), 833
 - `rsplit()` (*telegram.constants.MessageEntityType method*), 841
 - `rsplit()` (*telegram.constants.MessageOriginType method*), 851
 - `rsplit()` (*telegram.constants.MessageType method*), 862
 - `rsplit()` (*telegram.constants.ParseMode method*), 868
 - `rsplit()` (*telegram.constants.PollType method*), 879
 - `rsplit()` (*telegram.constants.ReactionEmoji method*), 897
 - `rsplit()` (*telegram.constants.ReactionType method*), 902
 - `rsplit()` (*telegram.constants.StickerFormat method*), 913
 - `rsplit()` (*telegram.constants.StickerType method*), 928
 - `rsplit()` (*telegram.constants.UpdateType method*), 935
 - `rstrip()` (*telegram.constants.BotCommandScopeType method*), 662
 - `rstrip()` (*telegram.constants.ChatAction method*), 685
 - `rstrip()` (*telegram.constants.ChatBoostSources method*), 691
 - `rstrip()` (*telegram.constants.ChatMemberStatus method*), 710
 - `rstrip()` (*telegram.constants.ChatType method*), 721
 - `rstrip()` (*telegram.constants.DiceEmoji method*), 735
 - `rstrip()` (*telegram.constants.InlineQueryResultType method*), 785
 - `rstrip()` (*telegram.constants.InputMediaType method*), 795
 - `rstrip()` (*telegram.constants.MaskPosition method*), 816
 - `rstrip()` (*telegram.constants.MenuButtonType method*), 826
 - `rstrip()` (*telegram.constants.MessageAttachmentType method*), 833
 - `rstrip()` (*telegram.constants.MessageEntityType method*), 841
 - `rstrip()` (*telegram.constants.MessageOriginType method*), 852
 - `rstrip()` (*telegram.constants.MessageType method*), 862
 - `rstrip()` (*telegram.constants.ParseMode method*), 868
 - `rstrip()` (*telegram.constants.PollType method*), 879
 - `rstrip()` (*telegram.constants.ReactionEmoji method*), 897
 - `rstrip()` (*telegram.constants.ReactionType method*), 903
 - `rstrip()` (*telegram.constants.StickerFormat method*), 913
 - `rstrip()` (*telegram.constants.StickerType method*), 928
 - `rstrip()` (*telegram.constants.UpdateType method*), 936
 - `run()` (*telegram.ext.Job method*), 562
 - `run_custom()` (*telegram.ext.JobQueue method*), 564
 - `run_daily()` (*telegram.ext.JobQueue method*), 564
 - `run_monthly()` (*telegram.ext.JobQueue method*), 565
 - `run_once()` (*telegram.ext.JobQueue method*), 566
 - `run_polling()` (*telegram.ext.Application method*), 525
 - `run_repeating()` (*telegram.ext.JobQueue method*), 566
 - `run_webhook()` (*telegram.ext.Application method*), 527
 - `running` (*telegram.ext.Application property*), 529
- ## S
- `SALUTING_FACE` (*telegram.constants.ReactionEmoji attribute*), 890
 - `scale` (*telegram.MaskPosition attribute*), 417
 - `schedule_removal()` (*telegram.ext.Job method*), 562
 - `scheduler` (*telegram.ext.JobQueue attribute*), 562
 - `scheduler_configuration` (*telegram.ext.JobQueue property*), 568
 - `score` (*telegram.GameHighScore attribute*), 493
 - `secret` (*telegram.DataCredentials attribute*), 494
 - `secret` (*telegram.EncryptedCredentials attribute*), 495
 - `secret` (*telegram.FileCredentials attribute*), 498
 - `secure_data` (*telegram.Credentials attribute*), 493
 - `SecureData` (*class in telegram*), 513
 - `SecureValue` (*class in telegram*), 515
 - `SEE_NO_EVIL_MONKEY` (*telegram.constants.ReactionEmoji attribute*), 890

- `selective` (*telegram.ForceReply* attribute), 247
- `selective` (*telegram.ReplyKeyboardMarkup* attribute), 364
- `selective` (*telegram.ReplyKeyboardRemove* attribute), 367
- `selfie` (*telegram.EncryptedPassportElement* attribute), 497
- `selfie` (*telegram.SecureValue* attribute), 515
- `send_action()` (*telegram.Chat* method), 192
- `send_action()` (*telegram.User* method), 392
- `send_animation()` (*telegram.Bot* method), 92
- `send_animation()` (*telegram.Chat* method), 192
- `send_animation()` (*telegram.User* method), 392
- `send_audio()` (*telegram.Bot* method), 94
- `send_audio()` (*telegram.Chat* method), 192
- `send_audio()` (*telegram.User* method), 392
- `send_chat_action()` (*telegram.Bot* method), 97
- `send_chat_action()` (*telegram.Chat* method), 193
- `send_chat_action()` (*telegram.User* method), 393
- `send_contact()` (*telegram.Bot* method), 98
- `send_contact()` (*telegram.Chat* method), 193
- `send_contact()` (*telegram.User* method), 393
- `send_copies()` (*telegram.Chat* method), 193
- `send_copies()` (*telegram.User* method), 394
- `send_copy()` (*telegram.Chat* method), 194
- `send_copy()` (*telegram.User* method), 394
- `send_dice()` (*telegram.Bot* method), 99
- `send_dice()` (*telegram.Chat* method), 194
- `send_dice()` (*telegram.User* method), 394
- `send_document()` (*telegram.Bot* method), 101
- `send_document()` (*telegram.Chat* method), 194
- `send_document()` (*telegram.User* method), 395
- `send_email_to_provider` (*telegram.InputInvoiceMessageContent* attribute), 481
- `send_game()` (*telegram.Bot* method), 103
- `send_game()` (*telegram.Chat* method), 195
- `send_game()` (*telegram.User* method), 395
- `send_invoice()` (*telegram.Bot* method), 104
- `send_invoice()` (*telegram.Chat* method), 195
- `send_invoice()` (*telegram.User* method), 396
- `send_location()` (*telegram.Bot* method), 107
- `send_location()` (*telegram.Chat* method), 195
- `send_location()` (*telegram.User* method), 396
- `send_media_group()` (*telegram.Bot* method), 109
- `send_media_group()` (*telegram.Chat* method), 196
- `send_media_group()` (*telegram.User* method), 397
- `send_message()` (*telegram.Bot* method), 111
- `send_message()` (*telegram.Chat* method), 196
- `send_message()` (*telegram.User* method), 397
- `send_phone_number_to_provider` (*telegram.InputInvoiceMessageContent* attribute), 480
- `send_photo()` (*telegram.Bot* method), 112
- `send_photo()` (*telegram.Chat* method), 196
- `send_photo()` (*telegram.User* method), 397
- `send_poll()` (*telegram.Bot* method), 115
- `send_poll()` (*telegram.Chat* method), 197
- `send_poll()` (*telegram.User* method), 398
- `send_sticker()` (*telegram.Bot* method), 117
- `send_sticker()` (*telegram.Chat* method), 197
- `send_sticker()` (*telegram.User* method), 398
- `send_venue()` (*telegram.Bot* method), 119
- `send_venue()` (*telegram.Chat* method), 197
- `send_venue()` (*telegram.User* method), 399
- `send_video()` (*telegram.Bot* method), 120
- `send_video()` (*telegram.Chat* method), 197
- `send_video()` (*telegram.User* method), 399
- `send_video_note()` (*telegram.Bot* method), 123
- `send_video_note()` (*telegram.Chat* method), 198
- `send_video_note()` (*telegram.User* method), 399
- `send_voice()` (*telegram.Bot* method), 125
- `send_voice()` (*telegram.Chat* method), 198
- `send_voice()` (*telegram.User* method), 400
- `sendAnimation()` (*telegram.Bot* method), 89
- `sendAudio()` (*telegram.Bot* method), 89
- `sendChatAction()` (*telegram.Bot* method), 90
- `sendContact()` (*telegram.Bot* method), 90
- `sendDice()` (*telegram.Bot* method), 90
- `sendDocument()` (*telegram.Bot* method), 90
- `SENDER` (*telegram.Chat* attribute), 180
- `SENDER` (*telegram.constants.ChatType* attribute), 716
- `sender_chat` (*telegram.Message* attribute), 300
- `sender_chat` (*telegram.MessageOriginChat* attribute), 346
- `sender_user` (*telegram.MessageOriginUser* attribute), 348
- `sender_user_name` (*telegram.MessageOriginHiddenUser* attribute), 347
- `SenderChat` (class in *telegram.ext.filters*), 605
- `sendGame()` (*telegram.Bot* method), 90
- `sendInvoice()` (*telegram.Bot* method), 90
- `sendLocation()` (*telegram.Bot* method), 90
- `sendMediaGroup()` (*telegram.Bot* method), 91
- `sendMessage()` (*telegram.Bot* method), 91
- `sendPhoto()` (*telegram.Bot* method), 91
- `sendPoll()` (*telegram.Bot* method), 91
- `sendSticker()` (*telegram.Bot* method), 91
- `sendVenue()` (*telegram.Bot* method), 91
- `sendVideo()` (*telegram.Bot* method), 91
- `sendVideoNote()` (*telegram.Bot* method), 92
- `sendVoice()` (*telegram.Bot* method), 92
- `SentWebAppMessage` (class in *telegram*), 370
- `SERIOUS_FACE_WITH_SYMBOLS_COVERING_MOUTH` (*telegram.constants.ReactionEmoji* attribute), 890
- `SERVICE_CHAT` (*telegram.constants.ChatID* attribute), 693
- `set_administrator_custom_title()` (*telegram.Chat* method), 198
- `set_application()` (*telegram.ext.JobQueue* method), 568
- `set_bot()` (*telegram.ext.BasePersistence* method), 634
- `set_bot()` (*telegram.TelegramObject* method), 375

- `set_chat_administrator_custom_title()` (*telegram.Bot method*), 129
- `set_chat_description()` (*telegram.Bot method*), 130
- `set_chat_menu_button()` (*telegram.Bot method*), 130
- `set_chat_permissions()` (*telegram.Bot method*), 131
- `set_chat_photo()` (*telegram.Bot method*), 132
- `set_chat_sticker_set()` (*telegram.Bot method*), 133
- `set_chat_title()` (*telegram.Bot method*), 133
- `set_credentials()` (*telegram.File method*), 246
- `set_custom_emoji_sticker_set_thumbnail()` (*telegram.Bot method*), 134
- `set_description()` (*telegram.Chat method*), 199
- `set_game_score()` (*telegram.Bot method*), 135
- `set_game_score()` (*telegram.CallbackQuery method*), 171
- `set_game_score()` (*telegram.Message method*), 334
- `set_menu_button()` (*telegram.Chat method*), 199
- `set_menu_button()` (*telegram.User method*), 400
- `set_message_reaction()` (*telegram.Bot method*), 136
- `set_message_reaction()` (*telegram.Chat method*), 199
- `set_my_commands()` (*telegram.Bot method*), 137
- `set_my_default_administrator_rights()` (*telegram.Bot method*), 137
- `set_my_description()` (*telegram.Bot method*), 138
- `set_my_name()` (*telegram.Bot method*), 139
- `set_my_short_description()` (*telegram.Bot method*), 139
- `set_name` (*telegram.Sticker attribute*), 419
- `set_passport_data_errors()` (*telegram.Bot method*), 140
- `set_permissions()` (*telegram.Chat method*), 200
- `set_photo()` (*telegram.Chat method*), 200
- `set_reaction()` (*telegram.Message method*), 335
- `set_sticker_emoji_list()` (*telegram.Bot method*), 141
- `set_sticker_keywords()` (*telegram.Bot method*), 141
- `set_sticker_mask_position()` (*telegram.Bot method*), 142
- `set_sticker_position_in_set()` (*telegram.Bot method*), 143
- `set_sticker_set_thumbnail()` (*telegram.Bot method*), 143
- `set_sticker_set_title()` (*telegram.Bot method*), 144
- `set_title()` (*telegram.Chat method*), 200
- `set_webhook()` (*telegram.Bot method*), 144
- `setChatAdministratorCustomTitle()` (*telegram.Bot method*), 127
- `setChatDescription()` (*telegram.Bot method*), 127
- `setChatMenuButton()` (*telegram.Bot method*), 127
- `setChatPermissions()` (*telegram.Bot method*), 127
- `setChatPhoto()` (*telegram.Bot method*), 128
- `setChatStickerSet()` (*telegram.Bot method*), 128
- `setChatTitle()` (*telegram.Bot method*), 128
- `setCustomEmojiStickerSetThumbnail()` (*telegram.Bot method*), 128
- `setGameScore()` (*telegram.Bot method*), 128
- `setMessageReaction()` (*telegram.Bot method*), 128
- `setMyCommands()` (*telegram.Bot method*), 128
- `setMyDefaultAdministratorRights()` (*telegram.Bot method*), 128
- `setMyDescription()` (*telegram.Bot method*), 128
- `setMyName()` (*telegram.Bot method*), 128
- `setMyShortDescription()` (*telegram.Bot method*), 128
- `setPassportDataErrors()` (*telegram.Bot method*), 128
- `setStickerEmojiList()` (*telegram.Bot method*), 129
- `setStickerKeywords()` (*telegram.Bot method*), 129
- `setStickerMaskPosition()` (*telegram.Bot method*), 129
- `setStickerPositionInSet()` (*telegram.Bot method*), 129
- `setStickerSetThumbnail()` (*telegram.Bot method*), 129
- `setStickerSetTitle()` (*telegram.Bot method*), 129
- `setWebhook()` (*telegram.Bot method*), 129
- `shipping_address` (*telegram.OrderInfo attribute*), 484
- `shipping_address` (*telegram.ShippingQuery attribute*), 489
- `shipping_option_id` (*telegram.PreCheckoutQuery attribute*), 485
- `shipping_option_id` (*telegram.SuccessfulPayment attribute*), 490
- `SHIPPING_QUERY` (*telegram.constants.UpdateType attribute*), 931
- `SHIPPING_QUERY` (*telegram.Update attribute*), 382
- `shipping_query` (*telegram.Update attribute*), 379
- `ShippingAddress` (*class in telegram*), 486
- `ShippingOption` (*class in telegram*), 487
- `ShippingQuery` (*class in telegram*), 488
- `ShippingQueryHandler` (*class in telegram.ext*), 625
- `SHOCKED_FACE_WITH_EXPLODING_HEAD` (*telegram.constants.ReactionEmoji attribute*), 890
- `short_description` (*telegram.BotShortDescription attribute*), 165
- `show_above_text` (*telegram.LinkPreviewOptions attribute*), 285
- `SHRUG` (*telegram.constants.ReactionEmoji attribute*), 890
- `shutdown()` (*telegram.Bot method*), 146
- `shutdown()` (*telegram.ext.AIORateLimiter method*), 650
- `shutdown()` (*telegram.ext.Application method*), 529
- `shutdown()` (*telegram.ext.BaseRateLimiter method*), 649
- `shutdown()` (*telegram.ext.BaseUpdateProcessor*

- method*), 547
- `shutdown()` (*telegram.ext.ExtBot method*), 559
- `shutdown()` (*telegram.ext.SimpleUpdateProcessor method*), 569
- `shutdown()` (*telegram.ext.Updater method*), 570
- `shutdown()` (*telegram.request.BaseRequest method*), 954
- `shutdown()` (*telegram.request.HTTPXRequest method*), 956
- `SimpleUpdateProcessor` (class in *telegram.ext*), 569
- `single_file` (*telegram.ext.PicklePersistence attribute*), 641
- `SIZE_BIG` (*telegram.ChatPhoto attribute*), 234
- `SIZE_SMALL` (*telegram.ChatPhoto attribute*), 234
- `SLEEPING_FACE` (*telegram.constants.ReactionEmoji attribute*), 891
- `SLOT_MACHINE` (*telegram.constants.DiceEmoji attribute*), 730
- `SLOT_MACHINE` (*telegram.Dice attribute*), 238
- `SLOT_MACHINE` (*telegram.ext.filters.Dice attribute*), 598
- `slow_mode_delay` (*telegram.Chat attribute*), 177
- `SMALL` (*telegram.constants.ChatPhotoSize attribute*), 712
- `small_file_id` (*telegram.ChatPhoto attribute*), 233
- `small_file_unique_id` (*telegram.ChatPhoto attribute*), 233
- `SMILING_FACE_WITH_HALO` (*telegram.constants.ReactionEmoji attribute*), 891
- `SMILING_FACE_WITH_HEART_SHAPED_EYES` (*telegram.constants.ReactionEmoji attribute*), 891
- `SMILING_FACE_WITH_HEARTS` (*telegram.constants.ReactionEmoji attribute*), 891
- `SMILING_FACE_WITH_HORNS` (*telegram.constants.ReactionEmoji attribute*), 891
- `SMILING_FACE_WITH_SUNGLASSES` (*telegram.constants.ReactionEmoji attribute*), 891
- `SNOWMAN` (*telegram.constants.ReactionEmoji attribute*), 891
- `socket_options()` (*telegram.ext.ApplicationBuilder method*), 544
- `source` (*telegram.ChatBoost attribute*), 207
- `source` (*telegram.ChatBoostRemoved attribute*), 207
- `source` (*telegram.ChatBoostSource attribute*), 208
- `source` (*telegram.ChatBoostSourceGiftCode attribute*), 209
- `source` (*telegram.ChatBoostSourceGiveaway attribute*), 209
- `source` (*telegram.ChatBoostSourcePremium attribute*), 210
- `source` (*telegram.PassportElementError attribute*), 501
- `SPEAK_NO_EVIL_MONKEY` (*telegram.constants.ReactionEmoji attribute*), 891
- `split()` (*telegram.constants.BotCommandScopeType method*), 662
- `split()` (*telegram.constants.ChatAction method*), 685
- `split()` (*telegram.constants.ChatBoostSources method*), 691
- `split()` (*telegram.constants.ChatMemberStatus method*), 711
- `split()` (*telegram.constants.ChatType method*), 721
- `split()` (*telegram.constants.DiceEmoji method*), 735
- `split()` (*telegram.constants.InlineQueryResultType method*), 785
- `split()` (*telegram.constants.InputMediaType method*), 795
- `split()` (*telegram.constants.MaskPosition method*), 816
- `split()` (*telegram.constants.MenuButtonType method*), 826
- `split()` (*telegram.constants.MessageAttachmentType method*), 834
- `split()` (*telegram.constants.MessageEntityType method*), 841
- `split()` (*telegram.constants.MessageOriginType method*), 852
- `split()` (*telegram.constants.MessageType method*), 862
- `split()` (*telegram.constants.ParseMode method*), 868
- `split()` (*telegram.constants.PollType method*), 879
- `split()` (*telegram.constants.ReactionEmoji method*), 897
- `split()` (*telegram.constants.ReactionType method*), 903
- `split()` (*telegram.constants.StickerFormat method*), 913
- `split()` (*telegram.constants.StickerType method*), 928
- `split()` (*telegram.constants.UpdateType method*), 936
- `splitlines()` (*telegram.constants.BotCommandScopeType method*), 663
- `splitlines()` (*telegram.constants.ChatAction method*), 685
- `splitlines()` (*telegram.constants.ChatBoostSources method*), 691
- `splitlines()` (*telegram.constants.ChatMemberStatus method*), 711
- `splitlines()` (*telegram.constants.ChatType method*), 721
- `splitlines()` (*telegram.constants.DiceEmoji method*), 735
- `splitlines()` (*telegram.constants.InlineQueryResultType method*), 785
- `splitlines()` (*telegram.constants.InputMediaType method*), 795
- `splitlines()` (*telegram.constants.MaskPosition method*), 816

- `splitlines()` (*telegram.constants.MenuButtonType method*), 826
- `splitlines()` (*telegram.constants.MessageAttachmentType method*), 834
- `splitlines()` (*telegram.constants.MessageEntityType method*), 841
- `splitlines()` (*telegram.constants.MessageOriginType method*), 852
- `splitlines()` (*telegram.constants.MessageType method*), 862
- `splitlines()` (*telegram.constants.ParseMode method*), 868
- `splitlines()` (*telegram.constants.PollType method*), 879
- `splitlines()` (*telegram.constants.ReactionEmoji method*), 897
- `splitlines()` (*telegram.constants.ReactionType method*), 903
- `splitlines()` (*telegram.constants.StickerFormat method*), 913
- `splitlines()` (*telegram.constants.StickerType method*), 928
- `splitlines()` (*telegram.constants.UpdateType method*), 936
- `SPOILER` (*telegram.constants.MessageEntityType attribute*), 836
- `SPOILER` (*telegram.MessageEntity attribute*), 343
- `SPOUTING_WHALE` (*telegram.constants.ReactionEmoji attribute*), 891
- `SQUARED_COOL` (*telegram.constants.ReactionEmoji attribute*), 891
- `start()` (*telegram.ext.Application method*), 529
- `start()` (*telegram.ext.JobQueue method*), 568
- `start_date` (*telegram.VideoChatScheduled attribute*), 408
- `start_parameter` (*telegram.InlineQueryResultsButton attribute*), 461
- `start_parameter` (*telegram.Invoice attribute*), 481
- `start_polling()` (*telegram.ext.Updater method*), 571
- `start_webhook()` (*telegram.ext.Updater method*), 572
- `startswith()` (*telegram.constants.BotCommandScopeType method*), 663
- `startswith()` (*telegram.constants.ChatAction method*), 686
- `startswith()` (*telegram.constants.ChatBoostSources method*), 692
- `startswith()` (*telegram.constants.ChatMemberStatus method*), 711
- `startswith()` (*telegram.constants.ChatType method*), 721
- `startswith()` (*telegram.constants.DiceEmoji method*), 735
- `startswith()` (*telegram.constants.InlineQueryResultType method*), 785
- `startswith()` (*telegram.constants.InputMediaType method*), 796
- `startswith()` (*telegram.constants.MaskPosition method*), 817
- `startswith()` (*telegram.constants.MenuButtonType method*), 827
- `startswith()` (*telegram.constants.MessageAttachmentType method*), 834
- `startswith()` (*telegram.constants.MessageEntityType method*), 841
- `startswith()` (*telegram.constants.MessageOriginType method*), 852
- `startswith()` (*telegram.constants.MessageType method*), 863
- `startswith()` (*telegram.constants.ParseMode method*), 869
- `startswith()` (*telegram.constants.PollType method*), 879
- `startswith()` (*telegram.constants.ReactionEmoji method*), 897
- `startswith()` (*telegram.constants.ReactionType method*), 903
- `startswith()` (*telegram.constants.StickerFormat method*), 913
- `startswith()` (*telegram.constants.StickerType method*), 929
- `startswith()` (*telegram.constants.UpdateType method*), 936
- `state` (*telegram.ext.ApplicationHandlerStop attribute*), 546
- `state` (*telegram.ResidentialAddress attribute*), 512
- `state` (*telegram.ShippingAddress attribute*), 486
- `states` (*telegram.ext.ConversationHandler property*), 589
- `STATIC` (*telegram.constants.StickerFormat attribute*), 908
- `STATIC` (*telegram.ext.filters.Sticker attribute*), 609
- `STATIC_THUMB_DIMENSIONS` (*telegram.constants.StickerSetLimit attribute*), 919
- `status` (*telegram.ChatMember attribute*), 217
- `status` (*telegram.ChatMemberAdministrator attribute*), 219
- `status` (*telegram.ChatMemberBanned attribute*), 222
- `status` (*telegram.ChatMemberLeft attribute*), 223
- `status` (*telegram.ChatMemberMember attribute*), 223
- `status` (*telegram.ChatMemberOwner attribute*), 224
- `status` (*telegram.ChatMemberRestricted attribute*), 226
- `StatusUpdate` (*class in telegram.ext.filters*), 607

- `Sticker` (class in `telegram`), 417
- `Sticker` (class in `telegram.ext.filters`), 609
- `STICKER` (`telegram.constants.InlineQueryResultType` attribute), 780
- `STICKER` (`telegram.constants.MessageAttachmentType` attribute), 828
- `STICKER` (`telegram.constants.MessageType` attribute), 856
- `sticker` (`telegram.ExternalReplyInfo` attribute), 242
- `sticker` (`telegram.InputSticker` attribute), 278
- `sticker` (`telegram.Message` attribute), 302
- `sticker_file_id` (`telegram.InlineQueryResultCachedSticker` attribute), 441
- `sticker_set_name` (`telegram.Chat` attribute), 178
- `sticker_type` (`telegram.StickerSet` attribute), 422
- `StickerFormat` (class in `telegram.constants`), 908
- `StickerLimit` (class in `telegram.constants`), 914
- `stickers` (`telegram.StickerSet` attribute), 422
- `StickerSet` (class in `telegram`), 421
- `StickerSetLimit` (class in `telegram.constants`), 918
- `StickerType` (class in `telegram.constants`), 923
- `stop()` (`telegram.ext.Application` method), 530
- `stop()` (`telegram.ext.JobQueue` method), 568
- `stop()` (`telegram.ext.Updater` method), 573
- `stop_live_location()` (`telegram.Message` method), 335
- `stop_message_live_location()` (`telegram.Bot` method), 146
- `stop_message_live_location()` (`telegram.CallbackQuery` method), 172
- `stop_poll()` (`telegram.Bot` method), 147
- `stop_poll()` (`telegram.Message` method), 335
- `stop_running()` (`telegram.ext.Application` method), 530
- `stopMessageLiveLocation()` (`telegram.Bot` method), 146
- `stopPoll()` (`telegram.Bot` method), 146
- `store_data` (`telegram.ext.BasePersistence` attribute), 631
- `store_data` (`telegram.ext.DictPersistence` attribute), 636
- `store_data` (`telegram.ext.PicklePersistence` attribute), 641
- `Story` (class in `telegram`), 371
- `STORY` (in module `telegram.ext.filters`), 591
- `STORY` (`telegram.constants.MessageAttachmentType` attribute), 828
- `STORY` (`telegram.constants.MessageType` attribute), 856
- `story` (`telegram.ExternalReplyInfo` attribute), 242
- `story` (`telegram.Message` attribute), 302
- `STRAWBERRY` (`telegram.constants.ReactionEmoji` attribute), 891
- `street_line1` (`telegram.ResidentialAddress` attribute), 512
- `street_line1` (`telegram.ShippingAddress` attribute), 487
- `street_line2` (`telegram.ResidentialAddress` attribute), 512
- `street_line2` (`telegram.ShippingAddress` attribute), 487
- `strict` (`telegram.ext.TypeHandler` attribute), 629
- `STRIKETHROUGH` (`telegram.constants.MessageEntityType` attribute), 836
- `STRIKETHROUGH` (`telegram.MessageEntity` attribute), 343
- `StringCommandHandler` (class in `telegram.ext`), 626
- `StringRegexHandler` (class in `telegram.ext`), 627
- `strip()` (`telegram.constants.BotCommandScopeType` method), 663
- `strip()` (`telegram.constants.ChatAction` method), 686
- `strip()` (`telegram.constants.ChatBoostSources` method), 692
- `strip()` (`telegram.constants.ChatMemberStatus` method), 711
- `strip()` (`telegram.constants.ChatType` method), 721
- `strip()` (`telegram.constants.DiceEmoji` method), 736
- `strip()` (`telegram.constants.InlineQueryResultType` method), 786
- `strip()` (`telegram.constants.InputMediaType` method), 796
- `strip()` (`telegram.constants.MaskPosition` method), 817
- `strip()` (`telegram.constants.MenuButtonType` method), 827
- `strip()` (`telegram.constants.MessageAttachmentType` method), 834
- `strip()` (`telegram.constants.MessageEntityType` method), 841
- `strip()` (`telegram.constants.MessageOriginType` method), 852
- `strip()` (`telegram.constants.MessageType` method), 863
- `strip()` (`telegram.constants.ParseMode` method), 869
- `strip()` (`telegram.constants.PollType` method), 879
- `strip()` (`telegram.constants.ReactionEmoji` method), 897
- `strip()` (`telegram.constants.ReactionType` method), 903
- `strip()` (`telegram.constants.StickerFormat` method), 913
- `strip()` (`telegram.constants.StickerType` method), 929
- `strip()` (`telegram.constants.UpdateType` method), 936
- `SUCCESSFUL_PAYMENT` (in module `telegram.ext.filters`), 591
- `SUCCESSFUL_PAYMENT` (`telegram.constants.MessageAttachmentType` attribute), 828
- `SUCCESSFUL_PAYMENT` (`telegram.constants.MessageType` attribute), 856
- `successful_payment` (`telegram.Message` attribute), 305
- `SuccessfulPayment` (class in `telegram`), 489

- SuccessfulPayment (class in telegram.ext.filters), 610
- suggested_tip_amounts (telegram.InputInvoiceMessageContent attribute), 479
- SUPER_GROUP (telegram.ext.filters.SenderChat attribute), 606
- SUPERGROUP (telegram.Chat attribute), 181
- SUPERGROUP (telegram.constants.ChatType attribute), 716
- SUPERGROUP (telegram.ext.filters.ChatType attribute), 596
- SUPERGROUP_CHAT_CREATED (telegram.constants.MessageType attribute), 856
- supergroup_chat_created (telegram.Message attribute), 304
- SUPPORTED_WEBHOOK_PORTS (in module telegram.constants), 908
- supports_inline_queries (telegram.Bot property), 148
- supports_inline_queries (telegram.User attribute), 386
- supports_streaming (telegram.InputMediaVideo attribute), 277
- SVG (telegram.ext.filters.Document attribute), 601
- swapcase() (telegram.constants.BotCommandScopeType method), 663
- swapcase() (telegram.constants.ChatAction method), 686
- swapcase() (telegram.constants.ChatBoostSources method), 692
- swapcase() (telegram.constants.ChatMemberStatus method), 711
- swapcase() (telegram.constants.ChatType method), 721
- swapcase() (telegram.constants.DiceEmoji method), 736
- swapcase() (telegram.constants.InlineQueryResultType method), 786
- swapcase() (telegram.constants.InputMediaType method), 796
- swapcase() (telegram.constants.MaskPosition method), 817
- swapcase() (telegram.constants.MenuButtonType method), 827
- swapcase() (telegram.constants.MessageAttachmentType method), 834
- swapcase() (telegram.constants.MessageEntityType method), 841
- swapcase() (telegram.constants.MessageOriginType method), 852
- swapcase() (telegram.constants.MessageType method), 863
- swapcase() (telegram.constants.ParseMode method), 869
- swapcase() (telegram.constants.PollType method), 879
- swapcase() (telegram.constants.ReactionEmoji method), 897
- swapcase() (telegram.constants.ReactionType method), 903
- swapcase() (telegram.constants.StickerFormat method), 913
- swapcase() (telegram.constants.StickerType method), 929
- swapcase() (telegram.constants.UpdateType method), 936
- switch_inline_query (telegram.InlineKeyboardButton attribute), 259
- switch_inline_query_chosen_chat (telegram.InlineKeyboardButton attribute), 260
- switch_inline_query_current_chat (telegram.InlineKeyboardButton attribute), 259
- SwitchInlineQueryChosenChat (class in telegram), 371
- ## T
- TARGZ (telegram.ext.filters.Document attribute), 601
- telegram module, 21
- telegram.constants module, 651
- telegram.error module, 945
- telegram.ext module, 516
- telegram.ext.filters module, 589
- telegram.helpers module, 948
- telegram.warnings module, 957
- telegram_payment_charge_id (telegram.SuccessfulPayment attribute), 490
- TelegramError, 947
- TelegramObject (class in telegram), 372
- temporary_registration (telegram.SecureData attribute), 514
- Text (class in telegram.ext.filters), 610
- TEXT (in module telegram.ext.filters), 591
- TEXT (telegram.constants.MessageType attribute), 856
- TEXT (telegram.ext.filters.Document attribute), 599
- text (telegram.Game attribute), 491
- text (telegram.InlineKeyboardButton attribute), 258
- text (telegram.InlineQueryResultsButton attribute), 461
- text (telegram.KeyboardButton attribute), 279
- text (telegram.MenuButtonWebApp attribute), 292
- text (telegram.Message attribute), 301
- text (telegram.PollOption attribute), 357
- text (telegram.TextQuote attribute), 376
- text_entities (telegram.Game attribute), 491
- text_html (telegram.Message property), 335

- `text_html_urled` (*telegram.Message* property), 336
- `TEXT_LINK` (*telegram.constants.MessageEntityType* attribute), 836
- `TEXT_LINK` (*telegram.MessageEntity* attribute), 343
- `text_markdown` (*telegram.Message* property), 336
- `text_markdown_urled` (*telegram.Message* property), 337
- `text_markdown_v2` (*telegram.Message* property), 337
- `text_markdown_v2_urled` (*telegram.Message* property), 338
- `TEXT_MENTION` (*telegram.constants.MessageEntityType* attribute), 836
- `TEXT_MENTION` (*telegram.MessageEntity* attribute), 343
- `TextQuote` (class in *telegram*), 376
- `THINKING_FACE` (*telegram.constants.ReactionEmoji* attribute), 892
- `thumbnail` (*telegram.Animation* attribute), 156
- `thumbnail` (*telegram.Audio* attribute), 158
- `thumbnail` (*telegram.Document* attribute), 239
- `thumbnail` (*telegram.InputMediaAnimation* attribute), 269
- `thumbnail` (*telegram.InputMediaAudio* attribute), 271
- `thumbnail` (*telegram.InputMediaDocument* attribute), 273
- `thumbnail` (*telegram.InputMediaVideo* attribute), 277
- `thumbnail` (*telegram.Sticker* attribute), 420
- `thumbnail` (*telegram.StickerSet* attribute), 422
- `thumbnail` (*telegram.Video* attribute), 407
- `thumbnail` (*telegram.VideoNote* attribute), 410
- `thumbnail_height` (*telegram.InlineQueryResultArticle* attribute), 429
- `thumbnail_height` (*telegram.InlineQueryResultContact* attribute), 446
- `thumbnail_height` (*telegram.InlineQueryResultDocument* attribute), 449
- `thumbnail_height` (*telegram.InlineQueryResultLocation* attribute), 455
- `thumbnail_height` (*telegram.InlineQueryResultVenue* attribute), 464
- `thumbnail_mime_type` (*telegram.InlineQueryResultGif* attribute), 452
- `thumbnail_mime_type` (*telegram.InlineQueryResultMpeg4Gif* attribute), 457
- `thumbnail_url` (*telegram.InlineQueryResultArticle* attribute), 429
- `thumbnail_url` (*telegram.InlineQueryResultContact* attribute), 446
- `thumbnail_url` (*telegram.InlineQueryResultDocument* attribute), 449
- `thumbnail_url` (*telegram.InlineQueryResultGif* attribute), 451
- `thumbnail_url` (*telegram.InlineQueryResultLocation* attribute), 454
- `thumbnail_url` (*telegram.InlineQueryResultMpeg4Gif* attribute), 457
- `thumbnail_url` (*telegram.InlineQueryResultPhoto* attribute), 459
- `thumbnail_url` (*telegram.InlineQueryResultVenue* attribute), 464
- `thumbnail_url` (*telegram.InlineQueryResultVideo* attribute), 466
- `thumbnail_width` (*telegram.InlineQueryResultArticle* attribute), 429
- `thumbnail_width` (*telegram.InlineQueryResultContact* attribute), 446
- `thumbnail_width` (*telegram.InlineQueryResultDocument* attribute), 449
- `thumbnail_width` (*telegram.InlineQueryResultLocation* attribute), 455
- `thumbnail_width` (*telegram.InlineQueryResultVenue* attribute), 464
- `THUMBS_DOWN` (*telegram.constants.ReactionEmoji* attribute), 892
- `THUMBS_UP` (*telegram.constants.ReactionEmoji* attribute), 892
- `TimedOut`, 948
- `TIMEOUT` (*telegram.ext.ConversationHandler* attribute), 587
- `title` (*telegram.Audio* attribute), 158
- `title` (*telegram.Chat* attribute), 176
- `title` (*telegram.Game* attribute), 491
- `title` (*telegram.InlineQueryResultArticle* attribute), 428
- `title` (*telegram.InlineQueryResultAudio* attribute), 430
- `title` (*telegram.InlineQueryResultCachedDocument* attribute), 434
- `title` (*telegram.InlineQueryResultCachedGif* attribute), 436
- `title` (*telegram.InlineQueryResultCachedMpeg4Gif* attribute), 438
- `title` (*telegram.InlineQueryResultCachedPhoto* attribute), 439
- `title` (*telegram.InlineQueryResultCachedVideo* attribute), 442
- `title` (*telegram.InlineQueryResultCachedVoice* attribute), 444
- `title` (*telegram.InlineQueryResultDocument* attribute), 448
- `title` (*telegram.InlineQueryResultGif* attribute), 452
- `title` (*telegram.InlineQueryResultLocation* attribute), 454

- 454
- `title` (*telegram.InlineQueryResultMpeg4Gif* attribute), 457
- `title` (*telegram.InlineQueryResultPhoto* attribute), 460
- `title` (*telegram.InlineQueryResultVenue* attribute), 463
- `title` (*telegram.InlineQueryResultVideo* attribute), 466
- `title` (*telegram.InlineQueryResultVoice* attribute), 468
- `title` (*telegram.InputInvoiceMessageContent* attribute), 479
- `title` (*telegram.InputMediaAudio* attribute), 271
- `title` (*telegram.InputVenueMessageContent* attribute), 474
- `title` (*telegram.Invoice* attribute), 481
- `title` (*telegram.ShippingOption* attribute), 488
- `title` (*telegram.StickerSet* attribute), 422
- `title` (*telegram.Venue* attribute), 404
- `title()` (*telegram.constants.BotCommandScopeType* method), 663
- `title()` (*telegram.constants.ChatAction* method), 686
- `title()` (*telegram.constants.ChatBoostSources* method), 692
- `title()` (*telegram.constants.ChatMemberStatus* method), 711
- `title()` (*telegram.constants.ChatType* method), 721
- `title()` (*telegram.constants.DiceEmoji* method), 736
- `title()` (*telegram.constants.InlineQueryResultType* method), 786
- `title()` (*telegram.constants.InputMediaType* method), 796
- `title()` (*telegram.constants.MaskPosition* method), 817
- `title()` (*telegram.constants.MenuButtonType* method), 827
- `title()` (*telegram.constants.MessageAttachmentType* method), 834
- `title()` (*telegram.constants.MessageEntityType* method), 841
- `title()` (*telegram.constants.MessageOriginType* method), 852
- `title()` (*telegram.constants.MessageType* method), 863
- `title()` (*telegram.constants.ParseMode* method), 869
- `title()` (*telegram.constants.PollType* method), 879
- `title()` (*telegram.constants.ReactionEmoji* method), 897
- `title()` (*telegram.constants.ReactionType* method), 903
- `title()` (*telegram.constants.StickerFormat* method), 913
- `title()` (*telegram.constants.StickerType* method), 929
- `title()` (*telegram.constants.UpdateType* method), 936
- `to_bytes()` (*telegram.constants.BotCommandLimit* method), 657
- `to_bytes()` (*telegram.constants.BotDescriptionLimit* method), 667
- `to_bytes()` (*telegram.constants.BotNameLimit* method), 671
- `to_bytes()` (*telegram.constants.BulkRequestLimit* method), 675
- `to_bytes()` (*telegram.constants.CallbackQueryLimit* method), 679
- `to_bytes()` (*telegram.constants.ChatID* method), 696
- `to_bytes()` (*telegram.constants.ChatInviteLinkLimit* method), 701
- `to_bytes()` (*telegram.constants.ChatLimit* method), 705
- `to_bytes()` (*telegram.constants.ChatPhotoSize* method), 715
- `to_bytes()` (*telegram.constants.ContactLimit* method), 725
- `to_bytes()` (*telegram.constants.CustomEmojiStickerLimit* method), 730
- `to_bytes()` (*telegram.constants.DiceLimit* method), 740
- `to_bytes()` (*telegram.constants.FileSizeLimit* method), 745
- `to_bytes()` (*telegram.constants.FloodLimit* method), 749
- `to_bytes()` (*telegram.constants.ForumIconColor* method), 754
- `to_bytes()` (*telegram.constants.ForumTopicLimit* method), 758
- `to_bytes()` (*telegram.constants.GiveawayLimit* method), 762
- `to_bytes()` (*telegram.constants.InlineKeyboardButtonLimit* method), 766
- `to_bytes()` (*telegram.constants.InlineKeyboardMarkupLimit* method), 770
- `to_bytes()` (*telegram.constants.InlineQueryLimit* method), 775
- `to_bytes()` (*telegram.constants.InlineQueryResultLimit* method), 779
- `to_bytes()` (*telegram.constants.InlineQueryResultsButtonLimit* method), 790
- `to_bytes()` (*telegram.constants.InvoiceLimit* method), 801
- `to_bytes()` (*telegram.constants.KeyboardButtonRequestUsersLimit* method), 805
- `to_bytes()` (*telegram.constants.LocationLimit* method), 811
- `to_bytes()` (*telegram.constants.MediaGroupLimit* method), 821
- `to_bytes()` (*telegram.constants.MessageLimit* method), 846
- `to_bytes()` (*telegram.constants.PollingLimit* method), 884
- `to_bytes()` (*telegram.constants.PollLimit* method), 874
- `to_bytes()` (*telegram.constants.ReplyLimit* method), 907
- `to_bytes()` (*telegram.constants.StickerLimit* method), 918

`to_bytes()` (*telegram.constants.StickerSetLimit* method), 923

`to_bytes()` (*telegram.constants.UserProfilePhotosLimit* method), 940

`to_bytes()` (*telegram.constants.WebhookLimit* method), 945

`to_dict()` (*telegram.Bot* method), 148

`to_dict()` (*telegram.PassportElementErrorFiles* method), 503

`to_dict()` (*telegram.PassportElementErrorTranslationFiles* method), 507

`to_dict()` (*telegram.PassportFile* method), 510

`to_dict()` (*telegram.TelegramObject* method), 375

`to_json()` (*telegram.TelegramObject* method), 375

`token` (*telegram.Bot* property), 148

`token()` (*telegram.ext.ApplicationBuilder* method), 544

`total_amount` (*telegram.Invoice* attribute), 482

`total_amount` (*telegram.PreCheckoutQuery* attribute), 485

`total_amount` (*telegram.SuccessfulPayment* attribute), 490

`TOTAL_BUTTON_NUMBER` (*telegram.constants.InlineKeyboardMarkupLimit* attribute), 766

`total_count` (*telegram.ReactionCount* attribute), 359

`total_count` (*telegram.UserProfilePhotos* attribute), 402

`total_voter_count` (*telegram.Poll* attribute), 353

`translate()` (*telegram.constants.BotCommandScopeType* method), 663

`translate()` (*telegram.constants.ChatAction* method), 686

`translate()` (*telegram.constants.ChatBoostSources* method), 692

`translate()` (*telegram.constants.ChatMemberStatus* method), 711

`translate()` (*telegram.constants.ChatType* method), 721

`translate()` (*telegram.constants.DiceEmoji* method), 736

`translate()` (*telegram.constants.InlineQueryResultType* method), 786

`translate()` (*telegram.constants.InputMediaType* method), 796

`translate()` (*telegram.constants.MaskPosition* method), 817

`translate()` (*telegram.constants.MenuButtonType* method), 827

`translate()` (*telegram.constants.MessageAttachmentType* method), 834

`translate()` (*telegram.constants.MessageEntityType* method), 842

`translate()` (*telegram.constants.MessageOriginType* method), 852

`translate()` (*telegram.constants.MessageType* method), 863

`translate()` (*telegram.constants.ParseMode* method), 869

`translate()` (*telegram.constants.PollType* method), 880

`translate()` (*telegram.constants.ReactionEmoji* method), 898

`translate()` (*telegram.constants.ReactionType* method), 903

`translate()` (*telegram.constants.StickerFormat* method), 913

`translate()` (*telegram.constants.StickerType* method), 929

`translate()` (*telegram.constants.UpdateType* method), 936

`translation` (*telegram.EncryptedPassportElement* attribute), 498

`translation` (*telegram.SecureValue* attribute), 515

`traveler` (*telegram.ProximityAlertTriggered* attribute), 358

`TROPHY` (*telegram.constants.ReactionEmoji* attribute), 892

`TXT` (*telegram.ext.filters.Document* attribute), 601

`type` (*telegram.BotCommandScope* attribute), 160

`type` (*telegram.BotCommandScopeAllChatAdministrators* attribute), 161

`type` (*telegram.BotCommandScopeAllGroupChats* attribute), 161

`type` (*telegram.BotCommandScopeAllPrivateChats* attribute), 162

`type` (*telegram.BotCommandScopeChat* attribute), 162

`type` (*telegram.BotCommandScopeChatAdministrators* attribute), 163

`type` (*telegram.BotCommandScopeChatMember* attribute), 163

`type` (*telegram.BotCommandScopeDefault* attribute), 164

`type` (*telegram.Chat* attribute), 176

`type` (*telegram.EncryptedPassportElement* attribute), 497

`type` (*telegram.ext.TypeHandler* attribute), 629

`type` (*telegram.InlineQueryResult* attribute), 427

`type` (*telegram.InlineQueryResultArticle* attribute), 428

`type` (*telegram.InlineQueryResultAudio* attribute), 430

`type` (*telegram.InlineQueryResultCachedAudio* attribute), 432

`type` (*telegram.InlineQueryResultCachedDocument* attribute), 433

`type` (*telegram.InlineQueryResultCachedGif* attribute), 435

`type` (*telegram.InlineQueryResultCachedMpeg4Gif* attribute), 437

`type` (*telegram.InlineQueryResultCachedPhoto* attribute), 439

`type` (*telegram.InlineQueryResultCachedSticker* attribute), 441

`type` (*telegram.InlineQueryResultCachedVideo* attribute), 442

`type` (*telegram.InlineQueryResultCachedVoice* attribute), 443

tribute), 444
 type (telegram.InlineQueryResultContact attribute), 445
 type (telegram.InlineQueryResultDocument attribute), 447
 type (telegram.InlineQueryResultGame attribute), 449
 type (telegram.InlineQueryResultGif attribute), 451
 type (telegram.InlineQueryResultLocation attribute), 453
 type (telegram.InlineQueryResultMpeg4Gif attribute), 456
 type (telegram.InlineQueryResultPhoto attribute), 459
 type (telegram.InlineQueryResultVenue attribute), 463
 type (telegram.InlineQueryResultVideo attribute), 465
 type (telegram.InlineQueryResultVoice attribute), 468
 type (telegram.InputMedia attribute), 266
 type (telegram.InputMediaAnimation attribute), 268
 type (telegram.InputMediaAudio attribute), 270
 type (telegram.InputMediaDocument attribute), 272
 type (telegram.InputMediaPhoto attribute), 274
 type (telegram.InputMediaVideo attribute), 276
 type (telegram.KeyboardButtonPollType attribute), 281
 type (telegram.MenuButton attribute), 290
 type (telegram.MenuButtonCommands attribute), 291
 type (telegram.MenuButtonDefault attribute), 291
 type (telegram.MenuButtonWebApp attribute), 292
 type (telegram.MessageEntity attribute), 340
 type (telegram.MessageOrigin attribute), 344
 type (telegram.MessageOriginChannel attribute), 345
 type (telegram.MessageOriginChat attribute), 346
 type (telegram.MessageOriginHiddenUser attribute), 347
 type (telegram.MessageOriginUser attribute), 348
 type (telegram.PassportElementError attribute), 501
 type (telegram.PassportElementErrorDataField attribute), 502
 type (telegram.PassportElementErrorFile attribute), 502
 type (telegram.PassportElementErrorFiles attribute), 503
 type (telegram.PassportElementErrorFrontSide attribute), 504
 type (telegram.PassportElementErrorReverseSide attribute), 504
 type (telegram.PassportElementErrorSelfie attribute), 505
 type (telegram.PassportElementErrorTranslationFile attribute), 506
 type (telegram.PassportElementErrorTranslationFiles attribute), 507
 type (telegram.PassportElementErrorUnspecified attribute), 508
 type (telegram.Poll attribute), 353
 type (telegram.ReactionCount attribute), 359
 type (telegram.ReactionType attribute), 360
 type (telegram.ReactionTypeCustomEmoji attribute), 360

type (telegram.ReactionTypeEmoji attribute), 361
 type (telegram.Sticker attribute), 419
 TypeHandler (class in telegram.ext), 628
 TYPING (telegram.constants.ChatAction attribute), 680
 tzinfo (telegram.ext.Defaults property), 556

U

unban_chat() (telegram.Chat method), 200
 unban_chat_member() (telegram.Bot method), 148
 unban_chat_sender_chat() (telegram.Bot method), 149
 unban_member() (telegram.Chat method), 201
 unban_sender_chat() (telegram.Chat method), 201
 unbanChatMember() (telegram.Bot method), 148
 unbanChatSenderChat() (telegram.Bot method), 148
 unclaimed_prize_count (telegram.GiveawayCompleted attribute), 253
 unclaimed_prize_count (telegram.GiveawayWinners attribute), 255
 UNDERLINE (telegram.constants.MessageEntityType attribute), 836
 UNDERLINE (telegram.MessageEntity attribute), 343
 unhide_general_forum_topic() (telegram.Bot method), 150
 unhide_general_forum_topic() (telegram.Chat method), 201
 unhideGeneralForumTopic() (telegram.Bot method), 150
 UNICORN_FACE (telegram.constants.ReactionEmoji attribute), 892
 unpin() (telegram.Message method), 338
 unpin_all_chat_messages() (telegram.Bot method), 151
 unpin_all_forum_topic_messages() (telegram.Bot method), 151
 unpin_all_forum_topic_messages() (telegram.Chat method), 201
 unpin_all_forum_topic_messages() (telegram.Message method), 339
 unpin_all_general_forum_topic_messages() (telegram.Bot method), 152
 unpin_all_general_forum_topic_messages() (telegram.Chat method), 202
 unpin_all_messages() (telegram.Chat method), 202
 unpin_all_messages() (telegram.User method), 401
 unpin_chat_message() (telegram.Bot method), 153
 unpin_message() (telegram.CallbackQuery method), 172
 unpin_message() (telegram.Chat method), 202
 unpin_message() (telegram.User method), 401
 unpinAllChatMessages() (telegram.Bot method), 151
 unpinAllForumTopicMessages() (telegram.Bot method), 151
 unpinAllGeneralForumTopicMessages() (telegram.Bot method), 151
 unpinChatMessage() (telegram.Bot method), 151

- `until_date` (*telegram.ChatMemberBanned* attribute), 222
- `until_date` (*telegram.ChatMemberRestricted* attribute), 227
- `Update` (class in *telegram*), 377
- `update()` (*telegram.ext.CallbackContext* method), 551
- `update_bot_data()` (*telegram.ext.BasePersistence* method), 634
- `update_bot_data()` (*telegram.ext.DictPersistence* method), 638
- `update_bot_data()` (*telegram.ext.PicklePersistence* method), 643
- `update_callback_data()` (*telegram.ext.BasePersistence* method), 634
- `update_callback_data()` (*telegram.ext.DictPersistence* method), 638
- `update_callback_data()` (*telegram.ext.PicklePersistence* method), 643
- `update_callback_data()` (*telegram.InlineKeyboardButton* method), 260
- `update_chat_data()` (*telegram.ext.BasePersistence* method), 634
- `update_chat_data()` (*telegram.ext.DictPersistence* method), 638
- `update_chat_data()` (*telegram.ext.PicklePersistence* method), 643
- `update_conversation()` (*telegram.ext.BasePersistence* method), 634
- `update_conversation()` (*telegram.ext.DictPersistence* method), 639
- `update_conversation()` (*telegram.ext.PicklePersistence* method), 643
- `update_id` (*telegram.Update* attribute), 378
- `update_interval` (*telegram.ext.BasePersistence* property), 635
- `update_persistence()` (*telegram.ext.Application* method), 530
- `update_processor` (*telegram.ext.Application* property), 530
- `update_queue` (*telegram.ext.Application* attribute), 518
- `update_queue` (*telegram.ext.CallbackContext* property), 552
- `update_queue` (*telegram.ext.Updater* attribute), 570
- `update_queue()` (*telegram.ext.ApplicationBuilder* method), 545
- `update_user_data()` (*telegram.ext.BasePersistence* method), 635
- `update_user_data()` (*telegram.ext.DictPersistence* method), 639
- `update_user_data()` (*telegram.ext.PicklePersistence* method), 643
- `UpdateFilter` (class in *telegram.ext.filters*), 610
- `Updater` (class in *telegram.ext*), 569
- `updater` (*telegram.ext.Application* attribute), 518
- `updater()` (*telegram.ext.ApplicationBuilder* method), 545
- `UpdateType` (class in *telegram.constants*), 929
- `UpdateType` (class in *telegram.ext.filters*), 611
- `UPLOAD_DOCUMENT` (*telegram.constants.ChatAction* attribute), 680
- `UPLOAD_PHOTO` (*telegram.constants.ChatAction* attribute), 680
- `upload_sticker_file()` (*telegram.Bot* method), 154
- `UPLOAD_VIDEO` (*telegram.constants.ChatAction* attribute), 680
- `UPLOAD_VIDEO_NOTE` (*telegram.constants.ChatAction* attribute), 680
- `UPLOAD_VOICE` (*telegram.constants.ChatAction* attribute), 681
- `uploadStickerFile()` (*telegram.Bot* method), 154
- `upper()` (*telegram.constants.BotCommandScopeType* method), 663
- `upper()` (*telegram.constants.ChatAction* method), 686
- `upper()` (*telegram.constants.ChatBoostSources* method), 692
- `upper()` (*telegram.constants.ChatMemberStatus* method), 711
- `upper()` (*telegram.constants.ChatType* method), 721
- `upper()` (*telegram.constants.DiceEmoji* method), 736
- `upper()` (*telegram.constants.InlineQueryResultType* method), 786
- `upper()` (*telegram.constants.InputMediaType* method), 796
- `upper()` (*telegram.constants.MaskPosition* method), 817
- `upper()` (*telegram.constants.MenuButtonType* method), 827
- `upper()` (*telegram.constants.MessageAttachmentType* method), 834
- `upper()` (*telegram.constants.MessageEntityType* method), 842
- `upper()` (*telegram.constants.MessageOriginType* method), 852
- `upper()` (*telegram.constants.MessageType* method), 863
- `upper()` (*telegram.constants.ParseMode* method), 869
- `upper()` (*telegram.constants.PollType* method), 880
- `upper()` (*telegram.constants.ReactionEmoji* method), 898
- `upper()` (*telegram.constants.ReactionType* method), 903
- `upper()` (*telegram.constants.StickerFormat* method), 914
- `upper()` (*telegram.constants.StickerType* method), 929
- `upper()` (*telegram.constants.UpdateType* method), 936
- `URL` (*telegram.constants.MessageEntityType* attribute), 836
- `url` (*telegram.InlineKeyboardButton* attribute), 258
- `url` (*telegram.InlineQueryResultArticle* attribute), 428
- `url` (*telegram.LinkPreviewOptions* attribute), 285
- `url` (*telegram.LoginUrl* attribute), 288
- `URL` (*telegram.MessageEntity* attribute), 343
- `url` (*telegram.MessageEntity* attribute), 341
- `url` (*telegram.WebAppInfo* attribute), 413
- `url` (*telegram.WebhookInfo* attribute), 414

- `url_encoded_parameters()` (*telegram.request.RequestData* method), 954
 - `User` (class in *telegram*), 384
 - `User` (class in *telegram.ext.filters*), 612
 - `USER` (in module *telegram.ext.filters*), 591
 - `user` (*telegram.ChatBoostSourceGiftCode* attribute), 209
 - `user` (*telegram.ChatBoostSourceGiveaway* attribute), 210
 - `user` (*telegram.ChatBoostSourcePremium* attribute), 210
 - `user` (*telegram.ChatMember* attribute), 217
 - `user` (*telegram.ChatMemberAdministrator* attribute), 220
 - `user` (*telegram.ChatMemberBanned* attribute), 222
 - `user` (*telegram.ChatMemberLeft* attribute), 223
 - `user` (*telegram.ChatMemberMember* attribute), 223
 - `user` (*telegram.ChatMemberOwner* attribute), 224
 - `user` (*telegram.ChatMemberRestricted* attribute), 226
 - `USER` (*telegram.constants.MessageOriginType* attribute), 847
 - `user` (*telegram.GameHighScore* attribute), 493
 - `user` (*telegram.MessageEntity* attribute), 341
 - `USER` (*telegram.MessageOrigin* attribute), 345
 - `user` (*telegram.MessageReactionUpdated* attribute), 350
 - `user` (*telegram.PollAnswer* attribute), 357
 - `user_administrator_rights` (*telegram.KeyboardButtonRequestChat* attribute), 282
 - `USER_AGENT` (*telegram.request.BaseRequest* attribute), 951
 - `USER_ATTACHMENT` (in module *telegram.ext.filters*), 591
 - `user_chat_id` (*telegram.ChatJoinRequest* attribute), 214
 - `user_data` (*telegram.ext.Application* attribute), 518
 - `user_data` (*telegram.ext.CallbackContext* property), 552
 - `user_data` (*telegram.ext.ContextTypes* property), 553
 - `user_data` (*telegram.ext.DictPersistence* property), 639
 - `user_data` (*telegram.ext.PersistenceInput* attribute), 639
 - `user_data_json` (*telegram.ext.DictPersistence* property), 639
 - `user_id` (*telegram.BotCommandScopeChatMember* attribute), 164
 - `user_id` (*telegram.Contact* attribute), 236
 - `user_id` (*telegram.ext.Job* attribute), 560
 - `user_id` (*telegram.UserShared* property), 403
 - `user_ids` (*telegram.ext.filters.User* property), 613
 - `user_ids` (*telegram.UsersShared* attribute), 404
 - `user_is_bot` (*telegram.KeyboardButtonRequestUsers* attribute), 284
 - `user_is_premium` (*telegram.KeyboardButtonRequestUsers* attribute), 284
 - `USER_SHARED` (*telegram.ext.filters.StatusUpdate* attribute), 608
 - `user_shared` (*telegram.Message* property), 339
 - `UserChatBoosts` (class in *telegram*), 402
 - `username` (*telegram.Bot* property), 154
 - `username` (*telegram.Chat* attribute), 176
 - `username` (*telegram.User* attribute), 386
 - `usernames` (*telegram.ext.filters.Chat* property), 596
 - `usernames` (*telegram.ext.filters.ForwardedFrom* property), 603
 - `usernames` (*telegram.ext.filters.SenderChat* property), 606
 - `usernames` (*telegram.ext.filters.User* property), 612
 - `usernames` (*telegram.ext.filters.ViaBot* property), 614
 - `UserProfilePhotos` (class in *telegram*), 402
 - `UserProfilePhotosLimit` (class in *telegram.constants*), 936
 - `users` (*telegram.VideoChatParticipantsInvited* attribute), 408
 - `USERS_SHARED` (*telegram.constants.MessageType* attribute), 856
 - `USERS_SHARED` (*telegram.ext.filters.StatusUpdate* attribute), 608
 - `users_shared` (*telegram.Message* attribute), 307
 - `UserShared` (class in *telegram*), 403
 - `UsersShared` (class in *telegram*), 403
 - `utility_bill` (*telegram.SecureData* attribute), 514
- ## V
- `value` (*telegram.Dice* attribute), 237
 - `VCARD` (*telegram.constants.ContactLimit* attribute), 722
 - `vcard` (*telegram.Contact* attribute), 236
 - `vcard` (*telegram.InlineQueryResultContact* attribute), 446
 - `vcard` (*telegram.InputContactMessageContent* attribute), 476
 - `Venue` (class in *telegram*), 404
 - `VENUE` (in module *telegram.ext.filters*), 591
 - `VENUE` (*telegram.constants.InlineQueryResultType* attribute), 780
 - `VENUE` (*telegram.constants.MessageAttachmentType* attribute), 829
 - `VENUE` (*telegram.constants.MessageType* attribute), 857
 - `venue` (*telegram.ExternalReplyInfo* attribute), 243
 - `venue` (*telegram.Message* attribute), 303
 - `VIA_BOT` (in module *telegram.ext.filters*), 592
 - `via_bot` (*telegram.Message* attribute), 305
 - `via_chat_folder_invite_link` (*telegram.ChatMemberUpdated* attribute), 229
 - `ViaBot` (class in *telegram.ext.filters*), 613
 - `Video` (class in *telegram*), 405
 - `VIDEO` (in module *telegram.ext.filters*), 592
 - `VIDEO` (*telegram.constants.InlineQueryResultType* attribute), 780
 - `VIDEO` (*telegram.constants.InputMediaType* attribute), 791
 - `VIDEO` (*telegram.constants.MessageAttachmentType* attribute), 829

- VIDEO (*telegram.constants.MessageType* attribute), 857
 - VIDEO (*telegram.constants.StickerFormat* attribute), 908
 - VIDEO (*telegram.ext.filters.Document* attribute), 599
 - VIDEO (*telegram.ext.filters.Sticker* attribute), 609
 - video (*telegram.ExternalReplyInfo* attribute), 242
 - video (*telegram.Message* attribute), 302
 - VIDEO_CHAT_ENDED (*telegram.constants.MessageType* attribute), 857
 - VIDEO_CHAT_ENDED (*telegram.ext.filters.StatusUpdate* attribute), 608
 - video_chat_ended (*telegram.Message* attribute), 306
 - VIDEO_CHAT_PARTICIPANTS_INVITED (*telegram.constants.MessageType* attribute), 857
 - VIDEO_CHAT_PARTICIPANTS_INVITED (*telegram.ext.filters.StatusUpdate* attribute), 609
 - video_chat_participants_invited (*telegram.Message* attribute), 306
 - VIDEO_CHAT_SCHEDULED (*telegram.constants.MessageType* attribute), 857
 - VIDEO_CHAT_SCHEDULED (*telegram.ext.filters.StatusUpdate* attribute), 608
 - video_chat_scheduled (*telegram.Message* attribute), 306
 - VIDEO_CHAT_STARTED (*telegram.constants.MessageType* attribute), 857
 - VIDEO_CHAT_STARTED (*telegram.ext.filters.StatusUpdate* attribute), 609
 - video_chat_started (*telegram.Message* attribute), 306
 - video_duration (*telegram.InlineQueryResultVideo* attribute), 467
 - video_file_id (*telegram.InlineQueryResultCachedVideo* attribute), 442
 - video_height (*telegram.InlineQueryResultVideo* attribute), 467
 - VIDEO_NOTE (in module *telegram.ext.filters*), 592
 - VIDEO_NOTE (*telegram.constants.MessageAttachmentType* attribute), 829
 - VIDEO_NOTE (*telegram.constants.MessageType* attribute), 857
 - video_note (*telegram.ExternalReplyInfo* attribute), 242
 - video_note (*telegram.Message* attribute), 303
 - video_url (*telegram.InlineQueryResultVideo* attribute), 466
 - video_width (*telegram.InlineQueryResultVideo* attribute), 466
 - VideoChatEnded (class in *telegram*), 407
 - VideoChatParticipantsInvited (class in *telegram*), 408
 - VideoChatScheduled (class in *telegram*), 408
 - VideoChatStarted (class in *telegram*), 409
 - VideoNote (class in *telegram*), 409
 - Voice (class in *telegram*), 411
 - VOICE (in module *telegram.ext.filters*), 592
 - VOICE (*telegram.constants.InlineQueryResultType* attribute), 780
 - VOICE (*telegram.constants.MessageAttachmentType* attribute), 829
 - VOICE (*telegram.constants.MessageType* attribute), 857
 - voice (*telegram.ExternalReplyInfo* attribute), 242
 - voice (*telegram.Message* attribute), 303
 - voice_duration (*telegram.InlineQueryResultVoice* attribute), 469
 - voice_file_id (*telegram.InlineQueryResultCachedVoice* attribute), 444
 - VOICE_NOTE_FILE_SIZE (*telegram.constants.FileSizeLimit* attribute), 741
 - voice_url (*telegram.InlineQueryResultVoice* attribute), 468
 - voter_chat (*telegram.PollAnswer* attribute), 357
 - voter_count (*telegram.PollOption* attribute), 357
- ## W
- WAITING (*telegram.ext.ConversationHandler* attribute), 588
 - was_refunded (*telegram.GiveawayWinners* attribute), 255
 - watcher (*telegram.ProximityAlertTriggered* attribute), 358
 - WAV (*telegram.ext.filters.Document* attribute), 601
 - WEB_APP (*telegram.constants.MenuButtonType* attribute), 822
 - web_app (*telegram.InlineKeyboardButton* attribute), 259
 - web_app (*telegram.InlineQueryResultsButton* attribute), 461
 - web_app (*telegram.KeyboardButton* attribute), 280
 - WEB_APP (*telegram.MenuButton* attribute), 290
 - web_app (*telegram.MenuButtonWebApp* attribute), 292
 - WEB_APP_DATA (*telegram.constants.MessageType* attribute), 857
 - WEB_APP_DATA (*telegram.ext.filters.StatusUpdate* attribute), 609
 - web_app_data (*telegram.Message* attribute), 306
 - web_app_name (*telegram.WriteAccessAllowed* attribute), 416
 - WebAppData (class in *telegram*), 412
 - WebAppInfo (class in *telegram*), 413
 - WebhookInfo (class in *telegram*), 413
 - WebhookLimit (class in *telegram.constants*), 941
 - width (*telegram.Animation* attribute), 156
 - width (*telegram.InputMediaAnimation* attribute), 269
 - width (*telegram.InputMediaVideo* attribute), 277
 - width (*telegram.PhotoSize* attribute), 351
 - width (*telegram.Sticker* attribute), 419

[width](#) (*telegram.Video* attribute), 406
[winner_count](#) (*telegram.Giveaway* attribute), 252
[winner_count](#) (*telegram.GiveawayCompleted* attribute), 253
[winner_count](#) (*telegram.GiveawayWinners* attribute), 255
[winners](#) (*telegram.GiveawayWinners* attribute), 255
[winners_selection_date](#) (*telegram.Giveaway* attribute), 252
[winners_selection_date](#) (*telegram.GiveawayWinners* attribute), 254
[WOMAN_SHRUGGING](#) (*telegram.constants.ReactionEmoji* attribute), 892
[WRITE_ACCESS_ALLOWED](#) (*telegram.constants.MessageType* attribute), 857
[WRITE_ACCESS_ALLOWED](#) (*telegram.ext.filters.StatusUpdate* attribute), 609
[write_access_allowed](#) (*telegram.Message* attribute), 307
[write_timeout\(\)](#) (*telegram.ext.ApplicationBuilder* method), 545
[WriteAccessAllowed](#) (class in *telegram*), 415
[WRITING_HAND](#) (*telegram.constants.ReactionEmoji* attribute), 892

X

[x_shift](#) (*telegram.MaskPosition* attribute), 417
[XML](#) (*telegram.ext.filters.Document* attribute), 601

Y

[y_shift](#) (*telegram.MaskPosition* attribute), 417
[YAWNING_FACE](#) (*telegram.constants.ReactionEmoji* attribute), 892
[YELLOW](#) (*telegram.constants.ForumIconColor* attribute), 750

Z

[ZERO_DATE](#) (in module *telegram.constants*), 945
[zfill\(\)](#) (*telegram.constants.BotCommandScopeType* method), 663
[zfill\(\)](#) (*telegram.constants.ChatAction* method), 686
[zfill\(\)](#) (*telegram.constants.ChatBoostSources* method), 692
[zfill\(\)](#) (*telegram.constants.ChatMemberStatus* method), 711
[zfill\(\)](#) (*telegram.constants.ChatType* method), 721
[zfill\(\)](#) (*telegram.constants.DiceEmoji* method), 736
[zfill\(\)](#) (*telegram.constants.InlineQueryResultType* method), 786
[zfill\(\)](#) (*telegram.constants.InputMediaType* method), 796
[zfill\(\)](#) (*telegram.constants.MaskPosition* method), 817
[zfill\(\)](#) (*telegram.constants.MenuButtonType* method), 827

[zfill\(\)](#) (*telegram.constants.MessageAttachmentType* method), 834
[zfill\(\)](#) (*telegram.constants.MessageEntityType* method), 842
[zfill\(\)](#) (*telegram.constants.MessageOriginType* method), 852
[zfill\(\)](#) (*telegram.constants.MessageType* method), 863
[zfill\(\)](#) (*telegram.constants.ParseMode* method), 869
[zfill\(\)](#) (*telegram.constants.PollType* method), 880
[zfill\(\)](#) (*telegram.constants.ReactionEmoji* method), 898
[zfill\(\)](#) (*telegram.constants.ReactionType* method), 903
[zfill\(\)](#) (*telegram.constants.StickerFormat* method), 914
[zfill\(\)](#) (*telegram.constants.StickerType* method), 929
[zfill\(\)](#) (*telegram.constants.UpdateType* method), 936
[ZIP](#) (*telegram.ext.filters.Document* attribute), 601