



Python Telegram Bot Documentation

Release 12.0.0b1

Leandro Toledo

Dec 31, 2020

1	telegram package	3
1.1	telegram.ext package	3
1.1.1	telegram.ext.Updater	3
1.1.2	telegram.ext.Dispatcher	6
1.1.3	telegram.ext.filters Module	8
1.1.4	telegram.ext.Job	14
1.1.5	telegram.ext.JobQueue	16
1.1.6	telegram.ext.MessageQueue	18
1.1.7	telegram.ext.DelayQueue	19
1.1.8	telegram.ext.CallbackContext	20
1.1.9	Handlers	22
1.1.10	Persistence	47
1.2	telegram.utils package	52
1.2.1	telegram.utils.helpers Module	52
1.2.2	telegram.utils.promise.Promise	53
1.2.3	telegram.utils.request.Request	54
1.3	telegram.Animation	55
1.4	telegram.Audio	56
1.5	telegram.Bot	57
1.6	telegram.CallbackQuery	93
1.7	telegram.Chat	95
1.8	telegram.ChatAction	100
1.9	telegram.ChatMember	101
1.10	telegram.ChatPhoto	103
1.11	telegram.constants Module	104
1.12	telegram.Contact	104
1.13	telegram.Document	105
1.14	telegram.error module	106
1.15	telegram.File	107
1.16	telegram.ForceReply	108
1.17	telegram.InlineKeyboardButton	109
1.18	telegram.InlineKeyboardMarkup	110
1.19	telegram.InputFile	111
1.20	telegram.InputMedia	111
1.21	telegram.InputMediaAnimation	112
1.22	telegram.InputMediaAudio	113
1.23	telegram.InputMediaDocument	114
1.24	telegram.InputMediaPhoto	115
1.25	telegram.InputMediaVideo	116
1.26	telegram.KeyboardButton	117

1.27	telegram.Location	118
1.28	telegram.Message	118
1.29	telegram.MessageEntity	131
1.30	telegram.ParseMode	133
1.31	telegram.PhotoSize	133
1.32	telegram.ReplyKeyboardRemove	134
1.33	telegram.ReplyKeyboardMarkup	135
1.34	telegram.ReplyMarkup	137
1.35	telegram.TelegramObject	137
1.36	telegram.Update	137
1.37	telegram.User	139
1.38	telegram.UserProfilePhotos	142
1.39	telegram.Venue	142
1.40	telegram.Video	143
1.41	telegram.VideoNote	144
1.42	telegram.Voice	145
1.43	telegram.WebhookInfo	146
1.44	Stickers	147
1.44.1	telegram.Sticker	147
1.44.2	telegram.StickerSet	149
1.44.3	telegram.MaskPosition	149
1.45	Inline Mode	150
1.45.1	telegram.InlineQuery	150
1.45.2	telegram.InlineQueryResult	151
1.45.3	telegram.InlineQueryResultArticle	152
1.45.4	telegram.InlineQueryResultAudio	153
1.45.5	telegram.InlineQueryResultCachedAudio	154
1.45.6	telegram.InlineQueryResultCachedDocument	156
1.45.7	telegram.InlineQueryResultCachedGif	157
1.45.8	telegram.InlineQueryResultCachedMpeg4Gif	158
1.45.9	telegram.InlineQueryResultCachedPhoto	159
1.45.10	telegram.InlineQueryResultCachedSticker	160
1.45.11	telegram.InlineQueryResultCachedVideo	161
1.45.12	telegram.InlineQueryResultCachedVoice	162
1.45.13	telegram.InlineQueryResultContact	163
1.45.14	telegram.InlineQueryResultDocument	165
1.45.15	telegram.InlineQueryResultGame	166
1.45.16	telegram.InlineQueryResultGif	167
1.45.17	telegram.InlineQueryResultLocation	168
1.45.18	telegram.InlineQueryResultMpeg4Gif	170
1.45.19	telegram.InlineQueryResultPhoto	171
1.45.20	telegram.InlineQueryResultVenue	173
1.45.21	telegram.InlineQueryResultVideo	174
1.45.22	telegram.InlineQueryResultVoice	176
1.45.23	telegram.InputMessageContent	177
1.45.24	telegram.InputTextMessageContent	177
1.45.25	telegram.InputLocationMessageContent	178
1.45.26	telegram.InputVenueMessageContent	178
1.45.27	telegram.InputContactMessageContent	179
1.45.28	telegram.ChosenInlineResult	180
1.46	Payments	181
1.46.1	telegram.LabeledPrice	181
1.46.2	telegram.Invoice	181
1.46.3	telegram.ShippingAddress	182
1.46.4	telegram.OrderInfo	183
1.46.5	telegram.ShippingOption	183
1.46.6	telegram.SuccessfulPayment	184
1.46.7	telegram.ShippingQuery	185

1.46.8	telegram.PreCheckoutQuery	186
1.47	Games	187
1.47.1	telegram.Game	187
1.47.2	telegram.Callbackgame	189
1.47.3	telegram.GameHighScore	189
1.48	Passport	189
1.48.1	telegram.PassportElementError	189
1.48.2	telegram.PassportElementErrorFile	190
1.48.3	telegram.PassportElementErrorReverseSide	190
1.48.4	telegram.PassportElementErrorFrontSide	191
1.48.5	telegram.PassportElementErrorFiles	191
1.48.6	telegram.PassportElementErrorDataField	192
1.48.7	telegram.Credentials	193
1.48.8	telegram.DataCredentials	193
1.48.9	telegram.SecureData	193
1.48.10	telegram.FileCredentials	194
1.48.11	telegram.IdDocumentData	194
1.48.12	telegram.PersonalDetails	195
1.48.13	telegram.ResidentialAddress	196
1.48.14	telegram.PassportData	196
1.48.15	telegram.PassportFile	197
1.48.16	telegram.EncryptedPassportElement	198
1.48.17	telegram.EncryptedCredentials	200
1.49	Module contents	201
2	Changelog	345
2.1	Changes	345
3	Version 12.0.0b1	347
3.1	Context based callbacks (#1100)	347
3.2	Persistence (#1017)	348
3.3	Handler overhaul (#1114)	348
3.4	Filter overhaul (#1221)	348
3.5	ConversationHandler	348
3.6	Bug fixes & improvements	348
3.7	Internal improvements	349
4	Pre-2019 (up and including to version 11.1.0)	351
5	Indices and tables	365
	Python Module Index	367
	Index	369

Below you can find the documentation for the python-telegram-bot library. except for the .ext package most of the objects in the package reflect the types as defined by the [telegram bot api](#).

1.1 telegram.ext package

1.1.1 telegram.ext.Updater

```
class telegram.ext.Updater (token=None, base_url=None, workers=4, bot=None,  
                             private_key=None, private_key_password=None,  
                             user_sig_handler=None, request_kwargs=None, persis-  
                             tence=None, use_context=False)
```

Bases: object

This class, which employs the `telegram.ext.Dispatcher`, provides a frontend to `telegram.Bot` to the programmer, so they can focus on coding the bot. Its purpose is to receive the updates from Telegram and to deliver them to said dispatcher. It also runs in a separate thread, so the user can interact with the bot, for example on the command line. The dispatcher supports handlers for different kinds of data: Updates from Telegram, basic text commands and even arbitrary types. The updater can be started as a polling service or, for production, use a webhook to receive updates. This is achieved using the `WebhookServer` and `WebhookHandler` classes.

bot

The bot used with this Updater.

Type `telegram.Bot`

user_sig_handler

signals the updater will respond to.

Type signal

update_queue

Queue for the updates.

Type Queue

job_queue

Jobqueue for the updater.

Type `telegram.ext.JobQueue`

dispatcher

Dispatcher that handles the updates and dispatches them to the handlers.

Type `telegram.ext.Dispatcher`

running

Indicates if the updater is running.

Type `bool`

persistence

Optional. The persistence class to store data that should be persistent over restarts.

Type `telegram.ext.BasePersistence`

use_context

True if using context based callbacks.

Type `bool`, optional

Parameters

- **token** (`str`, optional) – The bot’s token given by the @BotFather.
- **base_url** (`str`, optional) – Base_url for the bot.
- **workers** (`int`, optional) – Amount of threads in the thread pool for functions decorated with `@run_async`.
- **bot** (`telegram.Bot`, optional) – A pre-initialized bot instance. If a pre-initialized bot is used, it is the user’s responsibility to create it using a *Request* instance with a large enough connection pool.
- **private_key** (`bytes`, optional) – Private key for decryption of telegram passport data.
- **private_key_password** (`bytes`, optional) – Password for above private key.
- **user_sig_handler** (`function`, optional) – Takes `signum`, `frame` as positional arguments. This will be called when a signal is received, defaults are (`SIGINT`, `SIGTERM`, `SIGABRT`) settable with *idle*.
- **request_kwargs** (`dict`, optional) – Keyword args to control the creation of a *telegram.utils.request.Request* object (ignored if *bot* argument is used). The *request_kwargs* are very useful for the advanced users who would like to control the default timeouts and/or control the proxy used for http communication.
- **use_context** (`bool`, optional) – If set to `True` Use the context based callback API. During the deprecation period of the old API the default is `False`. **New users:** set this to `True`.
- **persistence** (`telegram.ext.BasePersistence`, optional) – The persistence class to store data that should be persistent over restarts.

Note: You must supply either a *bot* or a *token* argument.

Raises `ValueError` – If both *token* and *bot* are passed or none of them.

idle (*stop_signals*=(<*Signals.SIGINT*: 2>, <*Signals.SIGTERM*: 15>, <*Signals.SIGABRT*: 6>))

Blocks until one of the signals are received and stops the updater.

Parameters *stop_signals* (`iterable`) – Iterable containing signals from the signal module that should be subscribed to. `Updater.stop()` will be called on receiving one of those signals. Defaults to (`SIGINT`, `SIGTERM`, `SIGABRT`).

start_polling (*poll_interval*=0.0, *timeout*=10, *clean*=False, *bootstrap_retries*=-1, *read_latency*=2.0, *allowed_updates*=None)

Starts polling updates from Telegram.

Parameters

- **poll_interval** (float, optional) – Time to wait between polling updates from Telegram in seconds. Default is 0.0.
- **timeout** (float, optional) – Passed to `telegram.Bot.get_updates`.
- **clean** (bool, optional) – Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is False.
- **bootstrap_retries** (int, optional) – Whether the bootstrapping phase of the *Updater* will retry on failures on the Telegram server.
 - < 0 - retry indefinitely (default)
 - 0 - no retries
 - > 0 - retry up to X times
- **allowed_updates** (List[str], optional) – Passed to `telegram.Bot.get_updates`.
- **read_latency** (float | int, optional) – Grace time in seconds for receiving the reply from server. Will be added to the *timeout* value and used as the read timeout from server (Default: 2).

Returns The update queue that can be filled from the main thread.

Return type Queue

start_webhook (*listen*='127.0.0.1', *port*=80, *url_path*="", *cert*=None, *key*=None, *clean*=False, *bootstrap_retries*=0, *webhook_url*=None, *allowed_updates*=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

Parameters

- **listen** (str, optional) – IP-Address to listen on. Default 127.0.0.1.
- **port** (int, optional) – Port the bot should be listening on. Default 80.
- **url_path** (str, optional) – Path inside url.
- **cert** (str, optional) – Path to the SSL certificate file.
- **key** (str, optional) – Path to the SSL key file.
- **clean** (bool, optional) – Whether to clean any pending updates on Telegram servers before actually starting the webhook. Default is False.
- **bootstrap_retries** (int, optional) – Whether the bootstrapping phase of the *Updater* will retry on failures on the Telegram server.
 - < 0 - retry indefinitely (default)
 - 0 - no retries
 - > 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*.
- **allowed_updates** (List[str], optional) – Passed to `telegram.Bot.set_webhook`.

Returns The update queue that can be filled from the main thread.

Return type Queue

stop ()

Stops the polling/webhook thread, the dispatcher and the job queue.

1.1.2 telegram.ext.Dispatcher

class telegram.ext.Dispatcher(*bot*, *update_queue*, *workers=4*, *exception_event=None*,
 job_queue=None, *persistence=None*, *use_context=False*)

Bases: object

This class dispatches all kinds of updates to its registered handlers.

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 *Queue*

job_queue

Optional. The *telegram.ext.JobQueue* instance to pass onto handler callbacks.

Type *telegram.ext.JobQueue*

workers

Number of maximum concurrent worker threads for the @run_async decorator.

Type *int*

user_data

A dictionary handlers can use to store data for the user.

Type *defaultdict*

chat_data

A dictionary handlers can use to store data for the chat.

Type *defaultdict*

persistence

Optional. The persistence class to store data that should be persistent over restarts

Type *telegram.ext.BasePersistence*

Parameters

- **bot** (*telegram.Bot*) – The bot object that should be passed to the handlers.
- **update_queue** (*Queue*) – The synchronized queue that will contain the updates.
- **job_queue** (*telegram.ext.JobQueue*, optional) – The *telegram.ext.JobQueue* instance to pass onto handler callbacks.
- **workers** (*int*, optional) – Number of maximum concurrent worker threads for the @run_async decorator. defaults to 4.
- **persistence** (*telegram.ext.BasePersistence*, optional) – The persistence class to store data that should be persistent over restarts
- **use_context** (*bool*, optional) – If set to *True* Use the context based callback API. During the deprecation period of the old API the default is *False*. **New users:** set this to *True*.

add_error_handler (*callback*)

Registers an error handler in the Dispatcher.

Parameters **callback** (*callable*) – The callback function for this error handler. Will be called when an error is raised Callback signature for context based API:

```
def callback(update: Update, context: CallbackContext)
```

The error that happened will be present in `context.error`.

Note: See <https://git.io/fxJuV> for more info about switching to context based API.

add_handler (*handler*, *group=0*)

Register a handler.

TL;DR: Order and priority counts. 0 or 1 handlers per group will be used.

A handler must be an instance of a subclass of `telegram.ext.Handler`. 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.DispatcherHandlerStop` 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 should handle an update (see `telegram.ext.Handler.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.

Parameters

- **handler** (`telegram.ext.Handler`) – A Handler instance.
- **group** (`int`, optional) – The group identifier. Default is 0.

dispatch_error (*update*, *error*)

Dispatches an error.

Parameters

- **update** (`str` | `telegram.Update` | `None`) – The update that caused the error
- **error** (`telegram.TelegramError`) – The Telegram error that was raised.

error_handlers = None

A list of errorHandlers.

Type `List[callable]`

classmethod get_instance()

Get the singleton instance of this class.

Returns `telegram.ext.Dispatcher`

Raises `RuntimeError`

groups = None

A list with all groups.

Type `List[int]`

handlers = None

Holds the handlers per group.

Type `Dict[int, List[telegram.ext.Handler]]`

process_update (*update*)

Processes a single update.

Parameters **update** (`str` | `telegram.Update` | `telegram.TelegramError`) – The update to process.

remove_error_handler (*callback*)

Removes an error handler.

Parameters `callback` (callable) – The error handler to remove.

remove_handler (*handler*, *group=0*)
Remove a handler from the specified group.

Parameters

- **handler** (*telegram.ext.Handler*) – A Handler instance.
- **group** (object, optional) – The group identifier. Default is 0.

run_async (*func*, **args*, ***kwargs*)
Queue a function (with given args/kwars) to be run asynchronously.

Warning: If you're using `@run_async` you cannot rely on adding custom attributes to `:class:'telegram.ext.CallbackContext's`. See its docs for more info.

Parameters

- **func** (callable) – The function to run in the thread.
- ***args** (tuple, optional) – Arguments to *func*.
- ****kwargs** (dict, optional) – Keyword arguments to *func*.

Returns Promise

running = None
Indicates if this dispatcher is running.

Type bool

start (*ready=None*)
Thread target of thread 'dispatcher'.
Runs in background and processes the update queue.

Parameters **ready** (*threading.Event*, optional) – If specified, the event will be set once the dispatcher is ready.

stop ()
Stops the thread.

update_persistence ()
Update *user_data* and *chat_data* in *persistence*.

user_data = None
A dictionary handlers can use to store data for the user.

Type dict

1.1.3 telegram.ext.filters Module

This module contains the Filters for use with the MessageHandler class.

class telegram.ext.filters.BaseFilter

Bases: object

Base class for all Message Filters.

Subclassing from this class filters to be combined using bitwise operators:

And:

```
>>> (Filters.text & Filters.entity(MENTION))
```

Or:

```
>>> (Filters.audio | Filters.video)
```

Not:

```
>>> ~ Filters.command
```

Also works with more than two filters:

```
>>> (Filters.text & (Filters.entity(URL) | Filters.entity(TEXT_LINK)))
>>> Filters.text & (~ Filters.forwarded)
```

If you want to create your own filters create a class inheriting from this class 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.

name

Name for this filter. Defaults to the type of filter.

Type str

update_filter

Whether this filter should work on update. If False it will run the filter on update. `effective_message``. Default is False.

Type bool

data_filter

Whether this filter is a data filter. A data filter should return a dict with lists. The dict will be merged with `telegram.ext.CallbackContext`'s internal dict in most cases (depends on the handler).

Type bool

filter (*update*)

This method must be overwritten.

Note: If `update_filter` is false then the first argument is *message* and of type `telegram.Message`.

Parameters `update` (`telegram.Update`) – The update that is tested.

Returns dict or bool

class telegram.ext.filters.Filters

Bases: object

Predefined filters for use as the *filter* argument of `telegram.ext.MessageHandler`.

Examples

Use `MessageHandler(Filters.video, callback_method)` to filter all video messages. Use `MessageHandler(Filters.contact, callback_method)` for all contacts. etc.

all = **Filters.all**

All Messages.

Type Filter

animation = **Filters.animation**

Messages that contain `telegram.Animation`.

Type Filter

audio = **Filters.audio**

Messages that contain *telegram.Audio*.

Type Filter

class caption_entity(*entity_type*)

Bases: *telegram.ext.filters.BaseFilter*

Filters media messages to only allow those which have a *telegram.MessageEntity* where their *type* matches *entity_type*.

Examples

Example `MessageHandler(Filters.caption_entity("hashtag"), callback_method)`

Parameters **entity_type** – Caption Entity type to check for. All types can be found as constants in *telegram.MessageEntity*.

class chat(*chat_id=None, username=None*)

Bases: *telegram.ext.filters.BaseFilter*

Filters messages to allow only those which are from specified chat ID.

Examples

`MessageHandler(Filters.chat(-1234), callback_method)`

Parameters

- **chat_id** (*int* | *List[int]*, optional) – Which chat ID(s) to allow through.
- **username** (*str* | *List[str]*, optional) – Which username(s) to allow through. If username start swith '@' symbol, it will be ignored.

Raises *ValueError* – If *chat_id* and *username* are both present, or neither is.

command = **Filters.command**

Messages starting with /.

Type Filter

contact = **Filters.contact**

Messages that contain *telegram.Contact*.

Type Filter

document = **Filters.document**

Messages that contain *telegram.Document*.

Type Filter

class entity(*entity_type*)

Bases: *telegram.ext.filters.BaseFilter*

Filters messages to only allow those which have a *telegram.MessageEntity* where their *type* matches *entity_type*.

Examples

Example `MessageHandler(Filters.entity("hashtag"), callback_method)`

Parameters **entity_type** – Entity type to check for. All types can be found as constants in *telegram.MessageEntity*.

forwarded = Filters.forwarded

Messages that are forwarded.

Type Filter

game = Filters.game

Messages that contain *telegram.Game*.

Type Filter

group = Filters.group

Messages sent in a group chat.

Type Filter

invoice = Filters.invoice

Messages that contain *telegram.Invoice*.

Type Filter

class language (lang)

Bases: *telegram.ext.filters.BaseFilter*

Filters messages to only allow those which are from users with a certain language code.

Note: According to telegrams documentation, every single user does not have the *language_code* attribute.

Examples

```
MessageHandler(Filters.language("en"), callback_method)
```

Parameters lang (str | List[str]) – Which language code(s) to allow through. This will be matched using *.startswith* meaning that ‘en’ will match both ‘en_US’ and ‘en_GB’.

location = Filters.location

Messages that contain *telegram.Location*.

Type Filter

passport_data = Filters.passport_data

Messages that contain a *telegram.PassportData*

Type Filter

photo = Filters.photo

Messages that contain *telegram.PhotoSize*.

Type Filter

private = Filters.private

Messages sent in a private chat.

Type Filter

class regex (pattern)

Bases: *telegram.ext.filters.BaseFilter*

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.

Parameters `pattern` (`str` | `Pattern`) – The regex pattern.

`reply = Filters.reply`

Messages that are a reply to another message.

Type `Filter`

`status_update = Filters.status_update`

Subset for messages containing a status update.

Examples

Use these filters like: `Filters.status_update.new_chat_members` etc. Or use just `Filters.status_update` for all status update messages.

`chat_created`

Messages that contain `telegram.Message.group_chat_created`, `telegram.Message.supergroup_chat_created` or `telegram.Message.channel_chat_created`.

Type `Filter`

`delete_chat_photo`

Messages that contain `telegram.Message.delete_chat_photo`.

Type `Filter`

`left_chat_member`

Messages that contain `telegram.Message.left_chat_member`.

Type `Filter`

`migrate`

Messages that contain `telegram.Message.migrate_from_chat_id` or :attr: `telegram.Message.migrate_from_chat_id`.

Type `Filter`

`new_chat_members`

Messages that contain `telegram.Message.new_chat_members`.

Type `Filter`

`new_chat_photo`

Messages that contain `telegram.Message.new_chat_photo`.

Type `Filter`

`new_chat_title`

Messages that contain `telegram.Message.new_chat_title`.

Type `Filter`

`pinned_message`

Messages that contain `telegram.Message.pinned_message`.

Type `Filter`

`sticker = Filters.sticker`

Messages that contain `telegram.Sticker`.

Type `Filter`

`successful_payment = Filters.successful_payment`

Messages that confirm a `telegram.SuccessfulPayment`.

Type Filter

text = **Filters.text**
Text Messages.

Type Filter

update = **_UpdateType**
Subset for filtering the type of update.

Examples

Use these filters like: `Filters.update.message` or `Filters.update.channel_posts` etc. Or use just `Filters.update` for all types.

message

Updates with `telegram.Update.message`
Type Filter

edited_message

Updates with `telegram.Update.edited_message`
Type Filter

messages

Updates with either `telegram.Update.message` or `telegram.Update.edited_message`
Type Filter

channel_post

Updates with `telegram.Update.channel_post`
Type Filter

edited_channel_post

Updates with `telegram.Update.edited_channel_post`
Type Filter

channel_posts

Updates with either `telegram.Update.channel_post` or `telegram.Update.edited_channel_post`
Type Filter

class **user** (*user_id=None, username=None*)
Bases: `telegram.ext.filters.BaseFilter`

Filters messages to allow only those which are from specified user ID.

Examples

```
MessageHandler(Filters.user(1234), callback_method)
```

Parameters

- **user_id** (`int` | `List[int]`, optional) – Which user ID(s) to allow through.
- **username** (`str` | `List[str]`, optional) – Which username(s) to allow through. If username starts with '@' symbol, it will be ignored.

Raises `ValueError` – If `chat_id` and `username` are both present, or neither is.

venue = **Filters.venue**
Messages that contain `telegram.Venue`.

Type Filter

video = **Filters.video**
Messages that contain `telegram.Video`.

Type Filter

video_note = **Filters.video_note**
Messages that contain *telegram.VideoNote*.

Type Filter

voice = **Filters.voice**
Messages that contain *telegram.Voice*.

Type Filter

class telegram.ext.filters.**InvertedFilter** (*f*)

Bases: *telegram.ext.filters.BaseFilter*

Represents a filter that has been inverted.

Parameters **f** – The filter to invert.

filter (*update*)
This method must be overwritten.

Note: If *update_filter* is false then the first argument is *message* and of type *telegram.Message*.

Parameters **update** (*telegram.Update*) – The update that is tested.

Returns dict or bool

class telegram.ext.filters.**MergedFilter** (*base_filter, and_filter=None, or_filter=None*)

Bases: *telegram.ext.filters.BaseFilter*

Represents a filter consisting of two other filters.

Parameters

- **base_filter** – Filter 1 of the merged filter
- **and_filter** – Optional filter to “and” with *base_filter*. Mutually exclusive with *or_filter*.
- **or_filter** – Optional filter to “or” with *base_filter*. Mutually exclusive with *and_filter*.

filter (*update*)
This method must be overwritten.

Note: If *update_filter* is false then the first argument is *message* and of type *telegram.Message*.

Parameters **update** (*telegram.Update*) – The update that is tested.

Returns dict or bool

1.1.4 telegram.ext.Job

class telegram.ext.**Job** (*callback, interval=None, repeat=True, context=None, days=(0, 1, 2, 3, 4, 5, 6), name=None, job_queue=None*)

Bases: object

This class encapsulates a Job.

callback

The callback function that should be executed by the new job.

Type `callable`

context

Optional. Additional data needed for the callback function.

Type `object`

name

Optional. The name of the new job.

Type `str`

Parameters

- **callback** (`callable`) – The callback function that should be executed by the new job. It should take `bot`, `job` as parameters, where `job` is the `telegram.ext.Job` instance. It can be used to access it's `context` or change it to a repeating job.
- **interval** (`int` | `float` | `datetime.timedelta`, optional) – The interval in which the job will run. If it is an `int` or a `float`, it will be interpreted as seconds. If you don't set this value, you must set `repeat` to `False` and specify `next_t` when you put the job into the job queue.
- **repeat** (`bool`, optional) – If this job should be periodically execute its callback function (`True`) or only once (`False`). Defaults to `True`.
- **context** (`object`, optional) – Additional data needed for the callback function. Can be accessed through `job.context` in the callback. Defaults to `None`.
- **name** (`str`, optional) – The name of the new job. Defaults to `callback.__name__`.
- **days** (`Tuple[int]`, optional) – Defines on which days of the week the job should run. Defaults to `Days.EVERY_DAY`
- **job_queue** (`telegram.ext.JobQueue`, optional) – The `JobQueue` this job belongs to. Only optional for backward compatibility with `JobQueue.put()`.

days

Optional. Defines on which days of the week the job should run.

Type `Tuple[int]`

enabled

Whether this job is enabled.

Type `bool`

interval

Optional. The interval in which the job will run.

Type `int` | `float` | `datetime.timedelta`

interval_seconds

The interval for this job in seconds.

Type `int`

job_queue

Optional. The `JobQueue` this job belongs to.

Type `telegram.ext.JobQueue`

removed

Whether this job is due to be removed.

Type `bool`

repeat

Optional. If this job should periodically execute its callback function.

Type `bool`

run (*dispatcher*)

Executes the callback function.

schedule_removal ()

Schedules this job for removal from the `JobQueue`. It will be removed without executing its callback function again.

1.1.5 telegram.ext.JobQueue

class `telegram.ext.JobQueue` (*bot=None*)

Bases: `object`

This class allows you to periodically perform tasks with the bot.

_queue

The queue that holds the Jobs.

Type `PriorityQueue`

bot

The bot instance that should be passed to the jobs. DEPRECATED: Use `set_dispatcher` instead.

Type `telegram.Bot`

get_jobs_by_name (*name*)

Returns a tuple of jobs with the given name that are currently in the `JobQueue`

jobs ()

Returns a tuple of all jobs that are currently in the `JobQueue`.

run_daily (*callback, time, days=(0, 1, 2, 3, 4, 5, 6), context=None, name=None*)

Creates a new `Job` that runs on a daily basis and adds it to the queue.

Parameters

- **callback** (*callable*) – The callback function that should be executed by the new job. It should take *bot*, *job* as parameters, where *job* is the `telegram.ext.Job` instance. It can be used to access its `Job.context` or change it to a repeating job.
- **time** (*datetime.time*) – Time of day at which the job should run.
- **days** (*Tuple[int], optional*) – Defines on which days of the week the job should run. Defaults to `EVERY_DAY`
- **context** (*object, optional*) – Additional data needed for the callback function. Can be accessed through `job.context` in the callback. Defaults to `None`.
- **name** (*str, optional*) – The name of the new job. Defaults to `callback.__name__`.

Returns The new `Job` instance that has been added to the job queue.

Return type `telegram.ext.Job`

run_once (*callback, when, context=None, name=None*)

Creates a new `Job` that runs once and adds it to the queue.

Parameters

- **callback** (callable) – The callback function that should be executed by the new job. It should take `bot`, `job` as parameters, where `job` is the `telegram.ext.Job` instance. It can be used to access its `job.context` or change it to a repeating job.
- **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.
 - 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.
- **context** (object, optional) – Additional data needed for the callback function. Can be accessed through `job.context` in the callback. Defaults to None.
- **name** (str, optional) – The name of the new job. Defaults to `callback.__name__`.

Returns The new `Job` instance that has been added to the job queue.

Return type `telegram.ext.Job`

run_repeating (callback, interval, first=None, context=None, name=None)

Creates a new `Job` that runs at specified intervals and adds it to the queue.

Parameters

- **callback** (callable) – The callback function that should be executed by the new job. It should take `bot`, `job` as parameters, where `job` is the `telegram.ext.Job` instance. It can be used to access its `Job.context` or change it to a repeating job.
 - **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.
 - 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.
- Defaults to `interval`
- **context** (object, optional) – Additional data needed for the callback function. Can be accessed through `job.context` in the callback. Defaults to None.
 - **name** (str, optional) – The name of the new job. Defaults to `callback.__name__`.

Returns The new `Job` instance that has been added to the job queue.

Return type `telegram.ext.Job`

start()

Starts the `job_queue` thread.

stop()

Stops the thread.

tick()

Run all jobs that are due and re-enqueue them with their interval.

1.1.6 telegram.ext.MessageQueue

```
class telegram.ext.MessageQueue (all_burst_limit=30,          all_time_limit_ms=1000,
                                group_burst_limit=20,         group_time_limit_ms=60000,
                                exc_route=None, autostart=True)
```

Bases: `object`

Implements callback processing with proper delays to avoid hitting Telegram's message limits. Contains two `DelayQueue`, for group and for all messages, interconnected in delay chain. Callables are processed through `group` `DelayQueue`, then through `all` `DelayQueue` for group-type messages. For non-group messages, only the `all` `DelayQueue` is used.

Parameters

- **all_burst_limit** (`int`, optional) – Number of maximum *all-type* callbacks to process per time-window defined by `all_time_limit_ms`. Defaults to 30.
- **all_time_limit_ms** (`int`, optional) – Defines width of *all-type* time-window used when each processing limit is calculated. Defaults to 1000 ms.
- **group_burst_limit** (`int`, optional) – Number of maximum *group-type* callbacks to process per time-window defined by `group_time_limit_ms`. Defaults to 20.
- **group_time_limit_ms** (`int`, optional) – Defines width of *group-type* time-window used when each processing limit is calculated. Defaults to 60000 ms.
- **exc_route** (`callable`, optional) – A callable, accepting one positional argument; used to route exceptions from processor threads to main thread; is called on `Exception` subclass exceptions. If not provided, exceptions are routed through dummy handler, which re-raises them.
- **autostart** (`bool`, optional) – If `True`, processors are started immediately after object's creation; if `False`, should be started manually by `start` method. Defaults to `True`.

__call__ (`promise`, `is_group_msg=False`)

Processes callables in throughput-limiting queues to avoid hitting limits (specified with `burst_limit` and `time_limit`).

Parameters

- **promise** (`callable`) – Mainly the `telegram.utils.promise.Promise` (see Notes for other callables), that is processed in delay queues.
- **is_group_msg** (`bool`, optional) – Defines whether `promise` would be processed in `group*+*all* DelayQueue`s` (if set to `True`), or only through `all DelayQueue` (if set to `False`), resulting in needed delays to avoid hitting specified limits. Defaults to `False`.

Notes

Method is designed to accept `telegram.utils.promise.Promise` as `promise` argument, but other callables could be used too. For example, lambdas or simple functions could be used to wrap

original func to be called with needed args. In that case, be sure that either wrapper func does not raise outside exceptions or the proper `exc_route` handler is provided.

Returns Used as `promise` argument.

Return type `callable`

__init__ (*all_burst_limit=30, all_time_limit_ms=1000, group_burst_limit=20, group_time_limit_ms=60000, exc_route=None, autostart=True*)

Initialize self. See `help(type(self))` for accurate signature.

__weakref__

list of weak references to the object (if defined)

start ()

Method is used to manually start the `MessageQueue` processing.

stop (*timeout=None*)

Used to gently stop processor and shutdown its thread.

Parameters **timeout** (`float`) – Indicates maximum time to wait for processor to stop and its thread to exit. If timeout exceeds and processor has not stopped, method silently returns. `is_alive` could be used afterwards to check the actual status. `timeout` set to `None`, blocks until processor is shut down. Defaults to `None`.

1.1.7 telegram.ext.DelayQueue

class `telegram.ext.DelayQueue` (*queue=None, burst_limit=30, time_limit_ms=1000, exc_route=None, autostart=True, name=None*)

Bases: `threading.Thread`

Processes callbacks from queue with specified throughput limits. Creates a separate thread to process callbacks with delays.

burst_limit

Number of maximum callbacks to process per time-window.

Type `int`

time_limit

Defines width of time-window used when each processing limit is calculated.

Type `int`

exc_route

A callable, accepting 1 positional argument; used to route exceptions from processor thread to main thread;

Type `callable`

name

Thread's name.

Type `str`

Parameters

- **queue** (`Queue`, optional) – Used to pass callbacks to thread. Creates `Queue` implicitly if not provided.
- **burst_limit** (`int`, optional) – Number of maximum callbacks to process per time-window defined by `time_limit_ms`. Defaults to 30.
- **time_limit_ms** (`int`, optional) – Defines width of time-window used when each processing limit is calculated. Defaults to 1000.

- **exc_route** (callable, optional) – A callable, accepting 1 positional argument; used to route exceptions from processor thread to main thread; is called on *Exception* subclass exceptions. If not provided, exceptions are routed through dummy handler, which re-raises them.
- **autostart** (bool, optional) – If True, processor is started immediately after object's creation; if False, should be started manually by *start* method. Defaults to True.
- **name** (str, optional) – Thread's name. Defaults to 'DelayQueue-N', where N is sequential number of object created.

__call__ (*func*, **args*, ***kwargs*)

Used to process callbacks in throughput-limiting thread through queue.

Parameters

- **func** (callable) – The actual function (or any callable) that is processed through queue.
- ***args** (list) – Variable-length *func* arguments.
- ****kwargs** (dict) – Arbitrary keyword-arguments to *func*.

__init__ (*queue=None*, *burst_limit=30*, *time_limit_ms=1000*, *exc_route=None*, *autostart=True*, *name=None*)

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form “Thread-N” where N is a small decimal number.

args is the argument tuple for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

run ()

Do not use the method except for unthreaded testing purposes, the method normally is automatically called by autostart argument.

stop (*timeout=None*)

Used to gently stop processor and shutdown its thread.

Parameters **timeout** (float) – Indicates maximum time to wait for processor to stop and its thread to exit. If timeout exceeds and processor has not stopped, method silently returns. *is_alive* could be used afterwards to check the actual status. *timeout* set to None, blocks until processor is shut down. Defaults to None.

1.1.8 telegram.ext.CallbackContext

class telegram.ext.CallbackContext (*dispatcher*)

This is a context object passed to the callback called by *telegram.ext.Handler* or by the *telegram.ext.Dispatcher* in an error handler added by *telegram.ext.Dispatcher.add_error_handler* or to the callback of a *telegram.ext.Job*.

Note: *telegram.ext.Dispatcher* 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 get passed 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 and `@run_async`. Due to how `@run_async` works, 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.

chat_data

A dict that can be used to keep any data in. For each update from the same chat it will be the same dict.

Type dict, optional

user_data

A dict that can be used to keep any data in. For each update from the same user it will be the same dict.

Type dict, optional

matches

If the associated update originated from a regex-supported handler or had a `Filters.regex`, this will contain a list of match objects for every pattern where `re.search(pattern, string)` returned a match.

Type List[re match object], optional

args

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], optional

error

The Telegram error that was raised. Only present when passed to a error handler registered with `telegram.ext.Dispatcher.add_error_handler`.

Type telegram.TelegramError, optional

job

The job that that originated this callback. Only present when passed to the callback of `telegram.ext.Job`.

Type telegram.ext.Job

bot

The bot associated with this context.

Type telegram.Bot

job_queue

The `JobQueue` used by the `telegram.ext.Dispatcher` and (usually) the `telegram.ext.Updater` associated with this context.

Type telegram.ext.JobQueue

match

The first match from `matches`. Useful if you are only filtering using a single regex filter. Returns `None` if `matches` is empty.

Type Regex match type

update_queue

The `Queue` instance used by the `telegram.ext.Dispatcher` and (usually) the `telegram.ext.Updater` associated with this context.

Type `queue.Queue`

1.1.9 Handlers

`telegram.ext.Handler`

class `telegram.ext.Handler`(*callback*, *pass_update_queue=False*, *pass_job_queue=False*,
pass_user_data=False, *pass_chat_data=False*)

Bases: `object`

The base class for all update handlers. Create custom handlers by inheriting from it.

callback

The callback function for this handler.

Type `callable`

pass_update_queue

Determines whether `update_queue` will be passed to the callback function.

Type `bool`

pass_job_queue

Determines whether `job_queue` will be passed to the callback function.

Type `bool`

pass_user_data

Determines whether `user_data` will be passed to the callback function.

Type `bool`

pass_chat_data

Determines whether `chat_data` will be passed to the callback function.

Type `bool`

Note: `pass_user_data` and `pass_chat_data` determine whether a `dict` you can use to keep any data in will be sent to the `callback` function. Related to either the user or the chat that the update was sent in. For each update from the same user or in the same chat, it will be the same `dict`.

Note that this is DEPRECATED, and you should use context based callbacks. See <https://git.io/fxJuV> for more info.

Parameters

- **callback** (`callable`) – 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 for context based API:

```
def callback(update: Update, context: CallbackContext)
```

The return value of the callback is usually ignored except for the special case of `telegram.ext.ConversationHandler`.

- **pass_update_queue** (`bool`, optional) – If set to `True`, a keyword argument called `update_queue` will be passed to the callback function. It will be the `Queue` instance used by the `telegram.ext.Updater` and `telegram.ext.Dispatcher` that contains new updates which can be used to insert updates. Default is `False`. DEPRECATED: Please switch to context based callbacks.

- **pass_job_queue** (bool, optional) – If set to True, a keyword argument called `job_queue` will be passed to the callback function. It will be a `telegram.ext.JobQueue` instance created by the `telegram.ext.Updater` which can be used to schedule new jobs. Default is False. DEPRECATED: Please switch to context based callbacks.
- **pass_user_data** (bool, optional) – If set to True, a keyword argument called `user_data` will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.
- **pass_chat_data** (bool, optional) – If set to True, a keyword argument called `chat_data` will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.

check_update (*update*)

This method is called to determine if an update should be handled by this handler instance. It should always be overridden.

Parameters **update** (str | `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, dispatcher, 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.
- **dispatcher** (`telegram.ext.Dispatcher`) – The calling dispatcher.
- **check_result** – The result (return value) from `check_update`.

collect_optional_args (*dispatcher, update=None, check_result=None*)

Prepares the optional arguments. If the handler has additional optional args, it should subclass this method, but remember to call this super method.

DEPRECATED: This method is being replaced by new context based callbacks. Please see <https://git.io/fxJuV> for more info.

Parameters

- **dispatcher** (`telegram.ext.Dispatcher`) – The dispatcher.
- **update** (`telegram.Update`) – The update to gather chat/user id from.
- **check_result** – The result from `check_update`

handle_update (*update, dispatcher, check_result, context=None*)

This method is called if it was determined that an update should indeed be handled by this instance. Calls `self.callback` along with its respectful arguments. To work with the `telegram.ext.ConversationHandler`, this method returns the value returned from `self.callback`. Note that it can be overridden if needed by the subclassing handler.

Parameters

- **update** (str | `telegram.Update`) – The update to be handled.
- **dispatcher** (`telegram.ext.Dispatcher`) – The calling dispatcher.
- **check_result** – The result from `check_update`.

telegram.ext.CallbackQueryHandler

```
class telegram.ext.CallbackQueryHandler(callback, pass_update_queue=False,  
                                         pass_job_queue=False, pattern=None, pass_groups=False,  
                                         pass_groupdict=False, pass_user_data=False,  
                                         pass_chat_data=False)
```

Bases: telegram.ext.handler.Handler

Handler class to handle Telegram callback queries. Optionally based on a regex.

Read the documentation of the `re` module for more information.

callback

The callback function for this handler.

Type callable

pass_update_queue

Determines whether `update_queue` will be passed to the callback function.

Type bool

pass_job_queue

Determines whether `job_queue` will be passed to the callback function.

Type bool

pattern

Optional. Regex pattern to test `telegram.CallbackQuery.data` against.

Type str | Pattern

pass_groups

Determines whether `groups` will be passed to the callback function.

Type bool

pass_groupdict

Determines whether `groupdict` will be passed to the callback function.

Type bool

pass_user_data

Determines whether `user_data` will be passed to the callback function.

Type bool

pass_chat_data

Determines whether `chat_data` will be passed to the callback function.

Type bool

Note: `pass_user_data` and `pass_chat_data` determine whether a dict you can use to keep any data in will be sent to the `callback` function. Related to either the user or the chat that the update was sent in. For each update from the same user or in the same chat, it will be the same dict.

Note that this is DEPRECATED, and you should use context based callbacks. See <https://git.io/fxJuV> for more info.

Parameters

- **callback** (callable) – 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 for context based API:

```
def callback(update: Update, context:
    CallbackContext)
```

The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.

- **pass_update_queue** (bool, optional) – If set to True, a keyword argument called `update_queue` will be passed to the callback function. It will be the *Queue* instance used by the *telegram.ext.Updater* and *telegram.ext.Dispatcher* that contains new updates which can be used to insert updates. Default is False. DEPRECATED: Please switch to context based callbacks.
- **pass_job_queue** (bool, optional) – If set to True, a keyword argument called `job_queue` will be passed to the callback function. It will be a *telegram.ext.JobQueue* instance created by the *telegram.ext.Updater* which can be used to schedule new jobs. Default is False. DEPRECATED: Please switch to context based callbacks.
- **pattern** (str | *Pattern*, optional) – Regex pattern. If not None, `re.match` is used on *telegram.CallbackQuery.data* to determine if an update should be handled by this handler.
- **pass_groups** (bool, optional) – If the callback should be passed the result of `re.match(pattern, data).groups()` as a keyword argument called `groups`. Default is False DEPRECATED: Please switch to context based callbacks.
- **pass_groupdict** (bool, optional) – If the callback should be passed the result of `re.match(pattern, data).groupdict()` as a keyword argument called `groupdict`. Default is False DEPRECATED: Please switch to context based callbacks.
- **pass_user_data** (bool, optional) – If set to True, a keyword argument called `user_data` will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.
- **pass_chat_data** (bool, optional) – If set to True, a keyword argument called `chat_data` will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.

check_update (*update*)

Determines whether an update should be passed to this handlers *callback*.

Parameters **update** (*telegram.Update*) – Incoming telegram update.

Returns bool

collect_additional_context (*context, update, dispatcher, 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.
- **dispatcher** (*telegram.ext.Dispatcher*) – The calling dispatcher.
- **check_result** – The result (return value) from *check_update*.

collect_optional_args (*dispatcher, update=None, check_result=None*)

Prepares the optional arguments. If the handler has additional optional args, it should subclass this method, but remember to call this super method.

DEPRECATED: This method is being replaced by new context based callbacks. Please see <https://git.io/fxJuV> for more info.

Parameters

- **dispatcher** (*telegram.ext.Dispatcher*) – The dispatcher.
- **update** (*telegram.Update*) – The update to gather chat/user id from.
- **check_result** – The result from `check_update`

telegram.ext.ChosenInlineResultHandler

```
class telegram.ext.ChosenInlineResultHandler(callback, pass_update_queue=False,  
                                             pass_job_queue=False,  
                                             pass_user_data=False,  
                                             pass_chat_data=False)
```

Bases: `telegram.ext.handler.Handler`

Handler class to handle Telegram updates that contain a chosen inline result.

callback

The callback function for this handler.

Type callable

pass_update_queue

Determines whether `update_queue` will be passed to the callback function.

Type bool

pass_job_queue

Determines whether `job_queue` will be passed to the callback function.

Type bool

pass_user_data

Determines whether `user_data` will be passed to the callback function.

Type bool

pass_chat_data

Determines whether `chat_data` will be passed to the callback function.

Type bool

Note: `pass_user_data` and `pass_chat_data` determine whether a dict you can use to keep any data in will be sent to the `callback` function. Related to either the user or the chat that the update was sent in. For each update from the same user or in the same chat, it will be the same dict.

Note that this is DEPRECATED, and you should use context based callbacks. See <https://git.io/fxJuV> for more info.

Parameters

- **callback** (callable) – 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 for context based API:

```
def callback(update: Update, context: CallbackContext)
```

The return value of the callback is usually ignored except for the special case of `telegram.ext.ConversationHandler`.

- **pass_update_queue** (bool, optional) – If set to `True`, a keyword argument called `update_queue` will be passed to the callback function. It will be the `Queue` instance used by the `telegram.ext.Updater` and `telegram.ext.Dispatcher` that contains new updates which can be used to insert updates. Default is `False`. DEPRECATED: Please switch to context based callbacks.

- **pass_job_queue** (bool, optional) – If set to True, a keyword argument called `job_queue` will be passed to the callback function. It will be a `telegram.ext.JobQueue` instance created by the `telegram.ext.Updater` which can be used to schedule new jobs. Default is False. DEPRECATED: Please switch to context based callbacks.
- **pass_user_data** (bool, optional) – If set to True, a keyword argument called `user_data` will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.
- **pass_chat_data** (bool, optional) – If set to True, a keyword argument called `chat_data` will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.

check_update (*update*)

Determines whether an update should be passed to this handlers *callback*.

Parameters **update** (*telegram.Update*) – Incoming telegram update.

Returns bool

telegram.ext.ConversationHandler

```
class telegram.ext.ConversationHandler(entry_points, states, fallbacks, allow_reentry=False, per_chat=True, per_user=True, per_message=False, conversation_timeout=None, name=None, persistent=False)
```

Bases: `telegram.ext.handler.Handler`

A handler to hold a conversation with a single user by managing four collections of other handlers. Note that neither posts in Telegram Channels, nor group interactions with multiple users are managed by instances of this class.

The first collection, a list named *entry_points*, is used to initiate the conversation, for example with a `telegram.ext.CommandHandler` or `telegram.ext.RegexHandler`.

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 `@run_async` decorated 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.

The fourth, optional collection of handlers, a list named *timed_out_behavior* is used if the wait for `run_async` takes longer than defined in `run_async_timeout`. For example, you can let the user know that they should wait for a bit before they can continue.

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`.

entry_points

A list of `Handler` objects that can trigger the start of the conversation.

Type List[*telegram.ext.Handler*]

states

A dict that defines the different states of conversation a user can be in and one or more associated Handler objects that should be used in that state.

Type Dict[object, List[*telegram.ext.Handler*]]

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.Handler*]

allow_reentry

Determines if a user can restart a conversation with an entry point.

Type bool

per_chat

If the conversationkey should contain the Chat's ID.

Type bool

per_user

If the conversationkey should contain the User's ID.

Type bool

per_message

If the conversationkey should contain the Message's ID.

Type bool

conversation_timeout

Optional. When this handler is inactive more than this timeout (in seconds), it will be automatically ended. If this value is 0 (default), there will be no timeout. when it's triggered. The last received update will be handled by ALL the handler's who's *check_update* method returns True that are in the state *ConversationHandler.TIMEOUT*.

Type float`|:obj:`datetime.timedelta

name

Optional. The name for this conversationhandler. Required for persistence

Type str

persistent

Optional. If the conversations dict for this handler should be saved. Name is required and persistence has to be set in *telegram.ext.Updater*

Type bool

Parameters

- **entry_points** (List[*telegram.ext.Handler*]) – A list of Handler objects that can trigger the start of the conversation. The first handler which *check_update* method returns True will be used. If all return False, the update is not handled.
- **states** (Dict[object, List[*telegram.ext.Handler*]]) – A dict that defines the different states of conversation a user can be in and one or more associated Handler objects that should be used in that state. The first handler which *check_update* method returns True will be used.
- **fallbacks** (List[*telegram.ext.Handler*]) – A list of handlers that might be used if the user is in a conversation, but every handler for their current state returned False on *check_update*. The first handler which *check_update* method returns True will be used. If all return False, the update is not handled.
- **allow_reentry** (bool, optional) – If set to True, a user that is currently in a conversation can restart the conversation by triggering one of the entry points.

- **per_chat** (bool, optional) – If the conversationkey should contain the Chat’s ID. Default is True.
- **per_user** (bool, optional) – If the conversationkey should contain the User’s ID. Default is True.
- **per_message** (bool, optional) – If the conversationkey 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 will be handled by ALL the handler’s who’s *check_update* method returns True that are in the state *ConversationHandler.TIMEOUT*.
- **name** (str, optional) – The name for this conversationhandler. Required for persistence
- **persistent** (bool, optional) – If the conversations dict for this handler should be saved. Name is required and persistence has to be set in *telegram.ext.Updater*

Raises ValueError

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.

Type int

WAITING = -3

Used as a constant to handle state when a conversation is still waiting on the previous @run_sync decorated running handler to finish.

Type int

check_update (update)

Determines whether an update should be handled by this conversationhandler, and if so in which state the conversation currently is.

Parameters *update* (*telegram.Update*) – Incoming telegram update.

Returns bool

handle_update (update, dispatcher, check_result, context=None)

Send the update to the callback for the current state and Handler

Parameters

- **check_result** – The result from check_update. For this handler it’s a tuple of key, handler, and the handler’s check result.
- **update** (*telegram.Update*) – Incoming telegram update.
- **dispatcher** (*telegram.ext.Dispatcher*) – Dispatcher that originated the Update.

persistence = None

The persistence used to store conversations. Set by dispatcher

Type telegram.ext.BasePersistence

telegram.ext.CommandHandler

```
class telegram.ext.CommandHandler(command, callback, filters=None, allow_edited=None,  
                                pass_args=False, pass_update_queue=False,  
                                pass_job_queue=False, pass_user_data=False,  
                                pass_chat_data=False)
```

Bases: telegram.ext.handler.Handler

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.update.edited_message` in the filter argument.

command

The command or list of commands this handler should listen for. Limitations are the same as described here <https://core.telegram.org/bots#commands>

Type `str | List[str]`

callback

The callback function for this handler.

Type `callable`

filters

Optional. Only allow updates with these Filters.

Type `telegram.ext.BaseFilter`

allow_edited

Determines Whether the handler should also accept edited messages.

Type `bool`

pass_args

Determines whether the handler should be passed `args`.

Type `bool`

pass_update_queue

Determines whether `update_queue` will be passed to the callback function.

Type `bool`

pass_job_queue

Determines whether `job_queue` will be passed to the callback function.

Type `bool`

pass_user_data

Determines whether `user_data` will be passed to the callback function.

Type `bool`

pass_chat_data

Determines whether `chat_data` will be passed to the callback function.

Type `bool`

Note: `pass_user_data` and `pass_chat_data` determine whether a dict you can use to keep any data in will be sent to the `callback` function. Related to either the user or the chat that the update was sent in. For each update from the same user or in the same chat, it will be the same dict.

Note that this is DEPRECATED, and you should use context based callbacks. See <https://git.io/fxJuV> for more info.

Parameters

- **command** (`str` | `List[str]`) – The command or list of commands this handler should listen for. Limitations are the same as described here <https://core.telegram.org/bots#commands>
- **callback** (`callable`) – 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 for context based API:

```
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.BaseFilter`, optional) – A filter inheriting from `telegram.ext.filters.BaseFilter`. Standard filters can be found in `telegram.ext.filters.Filters`. Filters can be combined using bitwise operators (& for and, | for or, ~ for not).
- **allow_edited** (`bool`, optional) – Determines whether the handler should also accept edited messages. Default is `False`. DEPRECATED: Edited is allowed by default. To change this behavior use `~Filters.update.edited_message`.
- **pass_args** (`bool`, optional) – Determines whether the handler should be passed the arguments passed to the command as a keyword argument called `args`. It will contain a list of strings, which is the text following the command split on single or consecutive whitespace characters. Default is `False`. DEPRECATED: Please switch to context based callbacks.
- **pass_update_queue** (`bool`, optional) – If set to `True`, a keyword argument called `update_queue` will be passed to the callback function. It will be the `Queue` instance used by the `telegram.ext.Updater` and `telegram.ext.Dispatcher` that contains new updates which can be used to insert updates. Default is `False`. DEPRECATED: Please switch to context based callbacks.
- **pass_job_queue** (`bool`, optional) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `telegram.ext.JobQueue` instance created by the `telegram.ext.Updater` which can be used to schedule new jobs. Default is `False`. DEPRECATED: Please switch to context based callbacks.
- **pass_user_data** (`bool`, optional) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. Default is `False`. DEPRECATED: Please switch to context based callbacks.
- **pass_chat_data** (`bool`, optional) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. Default is `False`. DEPRECATED: Please switch to context based callbacks.

Raises `ValueError` - when command is too long or has illegal chars.

`check_update(update)`

Determines whether an update should be passed to this handlers `callback`.

Parameters `update` (`telegram.Update`) – Incoming telegram update.

Returns The list of args for the handler

Return type `list`

collect_additional_context (*context, update, dispatcher, 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.
- **dispatcher** (*telegram.ext.Dispatcher*) – The calling dispatcher.
- **check_result** – The result (return value) from *check_update*.

collect_optional_args (*dispatcher, update=None, check_result=None*)

Prepares the optional arguments. If the handler has additional optional args, it should subclass this method, but remember to call this super method.

DEPRECATED: This method is being replaced by new context based callbacks. Please see <https://git.io/fxJuV> for more info.

Parameters

- **dispatcher** (*telegram.ext.Dispatcher*) – The dispatcher.
- **update** (*telegram.Update*) – The update to gather chat/user id from.
- **check_result** – The result from *check_update*

telegram.ext.InlineQueryHandler

```
class telegram.ext.InlineQueryHandler (callback, pass_update_queue=False,
                                       pass_job_queue=False, pattern=None,
                                       pass_groups=False, pass_groupdict=False,
                                       pass_user_data=False, pass_chat_data=False)
```

Bases: *telegram.ext.handler.Handler*

Handler class to handle Telegram inline queries. Optionally based on a regex. Read the documentation of the *re* module for more information.

callback

The callback function for this handler.

Type callable

pass_update_queue

Determines whether *update_queue* will be passed to the callback function.

Type bool

pass_job_queue

Determines whether *job_queue* will be passed to the callback function.

Type bool

pattern

Optional. Regex pattern to test *telegram.InlineQuery.query* against.

Type str|Pattern

pass_groups

Determines whether *groups* will be passed to the callback function.

Type bool

pass_groupdict

Determines whether *groupdict*. will be passed to the callback function.

Type bool

pass_user_data

Determines whether `user_data` will be passed to the callback function.

Type `bool`

pass_chat_data

Determines whether `chat_data` will be passed to the callback function.

Type `bool`

Note: `pass_user_data` and `pass_chat_data` determine whether a `dict` you can use to keep any data in will be sent to the `callback` function. Related to either the user or the chat that the update was sent in. For each update from the same user or in the same chat, it will be the same `dict`.

Note that this is DEPRECATED, and you should use context based callbacks. See <https://git.io/fxJuV> for more info.

Parameters

- **callback** (callable) – 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 for context based API:

```
def callback(update: Update, context: CallbackContext)
```

The return value of the callback is usually ignored except for the special case of `telegram.ext.ConversationHandler`.

- **pass_update_queue** (bool, optional) – If set to `True`, a keyword argument called `update_queue` will be passed to the callback function. It will be the `Queue` instance used by the `telegram.ext.Updater` and `telegram.ext.Dispatcher` that contains new updates which can be used to insert updates. Default is `False`. DEPRECATED: Please switch to context based callbacks.
- **pass_job_queue** (bool, optional) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `telegram.ext.JobQueue` instance created by the `telegram.ext.Updater` which can be used to schedule new jobs. Default is `False`. DEPRECATED: Please switch to context based callbacks.
- **pattern** (str | 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.
- **pass_groups** (bool, optional) – If the callback should be passed the result of `re.match(pattern, data).groups()` as a keyword argument called `groups`. Default is `False` DEPRECATED: Please switch to context based callbacks.
- **pass_groupdict** (bool, optional) – If the callback should be passed the result of `re.match(pattern, data).groupdict()` as a keyword argument called `groupdict`. Default is `False` DEPRECATED: Please switch to context based callbacks.
- **pass_user_data** (bool, optional) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. Default is `False`. DEPRECATED: Please switch to context based callbacks.
- **pass_chat_data** (bool, optional) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. Default is `False`. DEPRECATED: Please switch to context based callbacks.

check_update (*update*)

Determines whether an update should be passed to this handlers `callback`.

Parameters `update` (*telegram.Update*) – Incoming telegram update.

Returns `bool`

collect_additional_context (*context, update, dispatcher, 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.
- **dispatcher** (*telegram.ext.Dispatcher*) – The calling dispatcher.
- **check_result** – The result (return value) from *check_update*.

collect_optional_args (*dispatcher, update=None, check_result=None*)

Prepares the optional arguments. If the handler has additional optional args, it should subclass this method, but remember to call this super method.

DEPRECATED: This method is being replaced by new context based callbacks. Please see <https://git.io/fxJuV> for more info.

Parameters

- **dispatcher** (*telegram.ext.Dispatcher*) – The dispatcher.
- **update** (*telegram.Update*) – The update to gather chat/user id from.
- **check_result** – The result from *check_update*

telegram.ext.MessageHandler

```
class telegram.ext.MessageHandler (filters,      callback,      pass_update_queue=False,
                                   pass_job_queue=False,      pass_user_data=False,
                                   pass_chat_data=False,      message_updates=None,
                                   channel_post_updates=None, edited_updates=None)
```

Bases: *telegram.ext.handler.Handler*

Handler class to handle telegram messages. They might contain text, media or status updates.

filters

Only allow updates with these Filters. See *telegram.ext.filters* for a full list of all available filters.

Type `Filter`

callback

The callback function for this handler.

Type `callable`

pass_update_queue

Determines whether *update_queue* will be passed to the callback function.

Type `bool`

pass_job_queue

Determines whether *job_queue* will be passed to the callback function.

Type `bool`

pass_user_data

Determines whether *user_data* will be passed to the callback function.

Type `bool`

pass_chat_data

Determines whether *chat_data* will be passed to the callback function.

Type bool

message_updates

Should “normal” message updates be handled? Default is None.

Type bool

channel_post_updates

Should channel posts updates be handled? Default is None.

Type bool

edited_updates

Should “edited” message updates be handled? Default is None.

Type bool

Note: `pass_user_data` and `pass_chat_data` determine whether a dict you can use to keep any data in will be sent to the `callback` function. Related to either the user or the chat that the update was sent in. For each update from the same user or in the same chat, it will be the same dict.

Note that this is DEPRECATED, and you should use context based callbacks. See <https://git.io/fxJuV> for more info.

Parameters

- **filters** (`telegram.ext.BaseFilter`, optional) – A filter inheriting from `telegram.ext.filters.BaseFilter`. Standard filters can be found in `telegram.ext.filters.Filters`. Filters can be combined using bitwise operators (& for and, | for or, ~ for not). Default is `telegram.ext.filters.Filters.update`. This defaults to all message_type updates being: message, edited_message, channel_post and edited_channel_post. If you don’t want or need any of those pass `~Filters.update.*` in the filter argument.
- **callback** (callable) – 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 for context based API:

```
def callback(update: Update, context: CallbackContext)
```

The return value of the callback is usually ignored except for the special case of `telegram.ext.ConversationHandler`.

- **pass_update_queue** (bool, optional) – If set to True, a keyword argument called `update_queue` will be passed to the callback function. It will be the `Queue` instance used by the `telegram.ext.Updater` and `telegram.ext.Dispatcher` that contains new updates which can be used to insert updates. Default is False. DEPRECATED: Please switch to context based callbacks.
- **pass_job_queue** (bool, optional) – If set to True, a keyword argument called `job_queue` will be passed to the callback function. It will be a `telegram.ext.JobQueue` instance created by the `telegram.ext.Updater` which can be used to schedule new jobs. Default is False. DEPRECATED: Please switch to context based callbacks.
- **pass_user_data** (bool, optional) – If set to True, a keyword argument called `user_data` will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.
- **pass_chat_data** (bool, optional) – If set to True, a keyword argument called `chat_data` will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.
- **message_updates** (bool, optional) – Should “normal” message updates be handled? Default is None. DEPRECATED: Please switch to filters for update filtering.

- **channel_post_updates** (*bool*, optional) – Should channel posts updates be handled? Default is *None*. DEPRECATED: Please switch to filters for update filtering.
- **edited_updates** (*bool*, optional) – Should “edited” message updates be handled? Default is *None*. DEPRECATED: Please switch to filters for update filtering.

Raises *ValueError*

check_update (*update*)

Determines whether an update should be passed to this handlers *callback*.

Parameters **update** (*telegram.Update*) – Incoming telegram update.

Returns *bool*

collect_additional_context (*context, update, dispatcher, 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.
- **dispatcher** (*telegram.ext.Dispatcher*) – The calling dispatcher.
- **check_result** – The result (return value) from *check_update*.

telegram.ext.PreCheckoutQueryHandler

```
class telegram.ext.PreCheckoutQueryHandler (callback, pass_update_queue=False,  
                                             pass_job_queue=False,  
                                             pass_user_data=False,  
                                             pass_chat_data=False)
```

Bases: *telegram.ext.handler.Handler*

Handler class to handle Telegram PreCheckout callback queries.

callback

The callback function for this handler.

Type *callable*

pass_update_queue

Determines whether *update_queue* will be passed to the callback function.

Type *bool*

pass_job_queue

Determines whether *job_queue* will be passed to the callback function.

Type *bool*

pass_user_data

Determines whether *user_data* will be passed to the callback function.

Type *bool*

pass_chat_data

Determines whether *chat_data* will be passed to the callback function.

Type *bool*

Note: *pass_user_data* and *pass_chat_data* determine whether a dict you can use to keep any data in will be sent to the *callback* function. Related to either the user or the chat that the update was sent in. For each update from the same user or in the same chat, it will be the same dict.

Note that this is DEPRECATED, and you should use context based callbacks. See <https://git.io/fxJuV> for more info.

Parameters

- **callback** (callable) – 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 for context based API:

```
def callback(update: Update, context: CallbackContext)
```

The return value of the callback is usually ignored except for the special case of `telegram.ext.ConversationHandler`.

- **pass_update_queue** (bool, optional) – If set to True, a keyword argument called `update_queue` will be passed to the callback function. It will be the `Queue` DEPRECATED: Please switch to context based callbacks. instance used by the `telegram.ext.Updater` and `telegram.ext.Dispatcher` that contains new updates which can be used to insert updates. Default is False.
- **pass_job_queue** (bool, optional) – If set to True, a keyword argument called `job_queue` will be passed to the callback function. It will be a `telegram.ext.JobQueue` instance created by the `telegram.ext.Updater` which can be used to schedule new jobs. Default is False. DEPRECATED: Please switch to context based callbacks.
- **pass_user_data** (bool, optional) – If set to True, a keyword argument called `user_data` will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.
- **pass_chat_data** (bool, optional) – If set to True, a keyword argument called `chat_data` will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.

`check_update(update)`

Determines whether an update should be passed to this handlers `callback`.

Parameters `update` (`telegram.Update`) – Incoming telegram update.

Returns bool

telegram.ext.PrefixHandler

```
class telegram.ext.PrefixHandler(prefix, command, callback, filters=None,
                                pass_args=False, pass_update_queue=False,
                                pass_job_queue=False, pass_user_data=False,
                                pass_chat_data=False)
```

Bases: `telegram.ext.commandhandler.CommandHandler`

Handler class to handle custom prefix commands

This is a 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`. It will contain 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'.
```

(continues on next page)

(continued from previous page)

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.update.edited_message`”.

prefix

The prefix(es) that will precede *command*.

Type `str | List[str]`

command

The command or list of commands this handler should listen for.

Type `str | List[str]`

callback

The callback function for this handler.

Type `callable`

filters

Optional. Only allow updates with these Filters.

Type `telegram.ext.BaseFilter`

pass_args

Determines whether the handler should be passed `args`.

Type `bool`

pass_update_queue

Determines whether `update_queue` will be passed to the callback function.

Type `bool`

pass_job_queue

Determines whether `job_queue` will be passed to the callback function.

Type `bool`

pass_user_data

Determines whether `user_data` will be passed to the callback function.

Type `bool`

pass_chat_data

Determines whether `chat_data` will be passed to the callback function.

Type `bool`

Note: `pass_user_data` and `pass_chat_data` determine whether a dict you can use to keep any data in will be sent to the *callback* function. Related to either the user or the chat that the update was sent in. For each update from the same user or in the same chat, it will be the same dict.

Note that this is DEPRECATED, and you should use context based callbacks. See <https://git.io/fxJuV> for more info.

Parameters

- **prefix** (`str` | `List[str]`) – The prefix(es) that will precede `command`.
- **command** (`str` | `List[str]`) – The command or list of commands this handler should listen for.
- **callback** (callable) – 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 for context based API:

```
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.BaseFilter`, optional) – A filter inheriting from `telegram.ext.filters.BaseFilter`. Standard filters can be found in `telegram.ext.filters.Filters`. Filters can be combined using bitwise operators (& for and, | for or, ~ for not).
- **pass_args** (`bool`, optional) – Determines whether the handler should be passed the arguments passed to the command as a keyword argument called `args`. It will contain a list of strings, which is the text following the command split on single or consecutive whitespace characters. Default is `False` DEPRECATED: Please switch to context based callbacks.
- **pass_update_queue** (`bool`, optional) – If set to `True`, a keyword argument called `update_queue` will be passed to the callback function. It will be the `Queue` instance used by the `telegram.ext.Updater` and `telegram.ext.Dispatcher` that contains new updates which can be used to insert updates. Default is `False`. DEPRECATED: Please switch to context based callbacks.
- **pass_job_queue** (`bool`, optional) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `telegram.ext.JobQueue` instance created by the `telegram.ext.Updater` which can be used to schedule new jobs. Default is `False`. DEPRECATED: Please switch to context based callbacks.
- **pass_user_data** (`bool`, optional) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. Default is `False`. DEPRECATED: Please switch to context based callbacks.
- **pass_chat_data** (`bool`, optional) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. Default is `False`. DEPRECATED: Please switch to context based callbacks.

`check_update` (`update`)

Determines whether an update should be passed to this handlers `callback`.

Parameters `update` (`telegram.Update`) – Incoming telegram update.

Returns The list of args for the handler

Return type `list`

`collect_additional_context` (`context`, `update`, `dispatcher`, `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.
- **dispatcher** (`telegram.ext.Dispatcher`) – The calling dispatcher.
- **check_result** – The result (return value) from `check_update`.

telegram.ext.RegexHandler

```
class telegram.ext.RegexHandler (pattern,          callback,          pass_groups=False,
                                pass_groupdict=False, pass_update_queue=False,
                                pass_job_queue=False, pass_user_data=False,
                                pass_chat_data=False, allow_edited=False, mes-
                                sage_updates=True,    channel_post_updates=False,
                                edited_updates=False)
```

Bases: telegram.ext.messagehandler.MessageHandler

Handler class to handle Telegram updates based on a regex.

It uses a regular expression to check text messages. 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.

pattern

The regex pattern.

Type `str|Pattern`

callback

The callback function for this handler.

Type `callable`

pass_groups

Determines whether groups will be passed to the callback function.

Type `bool`

pass_groupdict

Determines whether groupdict. will be passed to the callback function.

Type `bool`

pass_update_queue

Determines whether update_queue will be passed to the callback function.

Type `bool`

pass_job_queue

Determines whether job_queue will be passed to the callback function.

Type `bool`

pass_user_data

Determines whether user_data will be passed to the callback function.

Type `bool`

pass_chat_data

Determines whether chat_data will be passed to the callback function.

Type `bool`

Note: This handler is being deprecated. For the same usecase use: `MessageHandler(Filters.regex(r'pattern'), callback)`

Parameters

- **pattern** (`str|Pattern`) – The regex pattern.
- **callback** (`callable`) – 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 for context based API:

```
def callback(update: Update, context: CallbackContext)
```

The return value of the callback is usually ignored except for the special case of `telegram.ext.ConversationHandler`.

- **pass_groups** (bool, optional) – If the callback should be passed the result of `re.match(pattern, data).groups()` as a keyword argument called `groups`. Default is `False`
- **pass_groupdict** (bool, optional) – If the callback should be passed the result of `re.match(pattern, data).groupdict()` as a keyword argument called `groupdict`. Default is `False`
- **pass_update_queue** (bool, optional) – If set to `True`, a keyword argument called `update_queue` will be passed to the callback function. It will be the `Queue` instance used by the `telegram.ext.Updater` and `telegram.ext.Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.
- **pass_job_queue** (bool, optional) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `telegram.ext.JobQueue` instance created by the `telegram.ext.Updater` which can be used to schedule new jobs. Default is `False`.
- **pass_user_data** (bool, optional) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. Default is `False`.
- **pass_chat_data** (bool, optional) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. Default is `False`.
- **message_updates** (bool, optional) – Should “normal” message updates be handled? Default is `True`.
- **channel_post_updates** (bool, optional) – Should channel posts updates be handled? Default is `True`.
- **edited_updates** (bool, optional) – Should “edited” message updates be handled? Default is `False`.

Raises `ValueError`

collect_optional_args (*dispatcher, update=None, check_result=None*)

Prepares the optional arguments. If the handler has additional optional args, it should subclass this method, but remember to call this super method.

DEPRECATED: This method is being replaced by new context based callbacks. Please see <https://git.io/fxJuV> for more info.

Parameters

- **dispatcher** (`telegram.ext.Dispatcher`) – The dispatcher.
- **update** (`telegram.Update`) – The update to gather chat/user id from.
- **check_result** – The result from `check_update`

telegram.ext.ShippingQueryHandler

```
class telegram.ext.ShippingQueryHandler(callback, pass_update_queue=False,
                                         pass_job_queue=False,
                                         pass_user_data=False,
                                         pass_chat_data=False)
```

Bases: `telegram.ext.handler.Handler`

Handler class to handle Telegram shipping callback queries.

callback

The callback function for this handler.

Type callable

pass_update_queue

Determines whether `update_queue` will be passed to the callback function.

Type `bool`

pass_job_queue

Determines whether `job_queue` will be passed to the callback function.

Type `bool`

pass_user_data

Determines whether `user_data` will be passed to the callback function.

Type `bool`

pass_chat_data

Determines whether `chat_data` will be passed to the callback function.

Type `bool`

Note: `pass_user_data` and `pass_chat_data` determine whether a `dict` you can use to keep any data in will be sent to the `callback` function. Related to either the user or the chat that the update was sent in. For each update from the same user or in the same chat, it will be the same `dict`.

Note that this is DEPRECATED, and you should use context based callbacks. See <https://git.io/fxJuV> for more info.

Parameters

- **callback** (callable) – 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 for context based API:

```
def callback(update: Update, context: CallbackContext)
```

The return value of the callback is usually ignored except for the special case of `telegram.ext.ConversationHandler`.

- **pass_update_queue** (bool, optional) – If set to `True`, a keyword argument called `update_queue` will be passed to the callback function. It will be the `Queue` instance used by the `telegram.ext.Updater` and `telegram.ext.Dispatcher` that contains new updates which can be used to insert updates. Default is `False`. DEPRECATED: Please switch to context based callbacks.
- **pass_job_queue** (bool, optional) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `telegram.ext.JobQueue` instance created by the `telegram.ext.Updater` which can be used to schedule new jobs. Default is `False`. DEPRECATED: Please switch to context based callbacks.
- **pass_user_data** (bool, optional) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. Default is `False`. DEPRECATED: Please switch to context based callbacks.
- **pass_chat_data** (bool, optional) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. Default is `False`. DEPRECATED: Please switch to context based callbacks.

check_update (*update*)

Determines whether an update should be passed to this handlers `callback`.

Parameters **update** (`telegram.Update`) – Incoming telegram update.

Returns `bool`

telegram.ext.StringCommandHandler

```
class telegram.ext.StringCommandHandler(command, callback, pass_args=False,
                                       pass_update_queue=False,
                                       pass_job_queue=False)
```

Bases: telegram.ext.handler.Handler

Handler class to handle string commands. Commands are string updates that start with /.

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.

command

The command this handler should listen for.

Type `str`

callback

The callback function for this handler.

Type `callable`

pass_args

Determines whether the handler should be passed args.

Type `bool`

pass_update_queue

Determines whether `update_queue` will be passed to the callback function.

Type `bool`

pass_job_queue

Determines whether `job_queue` will be passed to the callback function.

Type `bool`

Parameters

- **callback** (`callable`) – 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 for context based API:

```
def callback(update: Update, context: CallbackContext)
```

The return value of the callback is usually ignored except for the special case of *telegram.ext.ConversationHandler*.

- **pass_args** (`bool`, optional) – Determines whether the handler should be passed the arguments passed to the command as a keyword argument called `args`. It will contain a list of strings, which is the text following the command split on single or consecutive whitespace characters. Default is `False` DEPRECATED: Please switch to context based callbacks.
- **pass_update_queue** (`bool`, optional) – If set to `True`, a keyword argument called `update_queue` will be passed to the callback function. It will be the `Queue` instance used by the *telegram.ext.Updater* and *telegram.ext.Dispatcher* that contains new updates which can be used to insert updates. Default is `False`. DEPRECATED: Please switch to context based callbacks.
- **pass_job_queue** (`bool`, optional) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a class:*telegram.ext.JobQueue* instance created by the *telegram.ext.Updater* which can be used to schedule new jobs. Default is `False`. DEPRECATED: Please switch to context based callbacks.

check_update (*update*)

Determines whether an update should be passed to this handlers *callback*.

Parameters **update** (*str*) – An incoming command.

Returns *bool*

collect_additional_context (*context, update, dispatcher, 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.
- **dispatcher** (*telegram.ext.Dispatcher*) – The calling dispatcher.
- **check_result** – The result (return value) from *check_update*.

collect_optional_args (*dispatcher, update=None, check_result=None*)

Prepares the optional arguments. If the handler has additional optional args, it should subclass this method, but remember to call this super method.

DEPRECATED: This method is being replaced by new context based callbacks. Please see <https://git.io/fxJuV> for more info.

Parameters

- **dispatcher** (*telegram.ext.Dispatcher*) – The dispatcher.
- **update** (*telegram.Update*) – The update to gather chat/user id from.
- **check_result** – The result from *check_update*

telegram.ext.StringRegexHandler

```
class telegram.ext.StringRegexHandler (pattern, callback, pass_groups=False,  
                                       pass_groupdict=False,  
                                       pass_update_queue=False,  
                                       pass_job_queue=False)
```

Bases: *telegram.ext.handler.Handler*

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.

pattern

The regex pattern.

Type *str|Pattern*

callback

The callback function for this handler.

Type *callable*

pass_groups

Determines whether groups will be passed to the callback function.

Type *bool*

pass_groupdict

Determines whether groupdict. will be passed to the callback function.

Type `bool`

pass_update_queue

Determines whether `update_queue` will be passed to the callback function.

Type `bool`

pass_job_queue

Determines whether `job_queue` will be passed to the callback function.

Type `bool`

Parameters

- **pattern** (`str` | `Pattern`) – The regex pattern.
- **callback** (callable) – 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 for context based API:

```
def callback(update: Update, context: CallbackContext)
```

The return value of the callback is usually ignored except for the special case of `telegram.ext.ConversationHandler`.

- **pass_groups** (`bool`, optional) – If the callback should be passed the result of `re.match(pattern, data).groups()` as a keyword argument called `groups`. Default is `False` DEPRECATED: Please switch to context based callbacks.
- **pass_groupdict** (`bool`, optional) – If the callback should be passed the result of `re.match(pattern, data).groupdict()` as a keyword argument called `groupdict`. Default is `False` DEPRECATED: Please switch to context based callbacks.
- **pass_update_queue** (`bool`, optional) – If set to `True`, a keyword argument called `update_queue` will be passed to the callback function. It will be the `Queue` instance used by the `telegram.ext.Updater` and `telegram.ext.Dispatcher` that contains new updates which can be used to insert updates. Default is `False`. DEPRECATED: Please switch to context based callbacks.
- **pass_job_queue** (`bool`, optional) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `telegram.ext.JobQueue` instance created by the `telegram.ext.Updater` which can be used to schedule new jobs. Default is `False`. DEPRECATED: Please switch to context based callbacks.

check_update (`update`)

Determines whether an update should be passed to this handlers `callback`.

Parameters `update` (`str`) – An incoming command.

Returns `bool`

collect_additional_context (`context`, `update`, `dispatcher`, `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.
- **dispatcher** (`telegram.ext.Dispatcher`) – The calling dispatcher.
- **check_result** – The result (return value) from `check_update`.

collect_optional_args (`dispatcher`, `update=None`, `check_result=None`)

Prepares the optional arguments. If the handler has additional optional args, it should subclass this method, but remember to call this super method.

DEPRECATED: This method is being replaced by new context based callbacks. Please see <https://git.io/fxJuV> for more info.

Parameters

- **dispatcher** (*telegram.ext.Dispatcher*) – The dispatcher.
- **update** (*telegram.Update*) – The update to gather chat/user id from.
- **check_result** – The result from check_update

telegram.ext.TypeHandler

```
class telegram.ext.TypeHandler(type, callback, strict=False, pass_update_queue=False,  
                               pass_job_queue=False)
```

Bases: telegram.ext.handler.Handler

Handler class to handle updates of custom types.

type

The type of updates this handler should process.

Type *type*

callback

The callback function for this handler.

Type callable

strict

Use type instead of isinstance. Default is False.

Type bool

pass_update_queue

Determines whether update_queue will be passed to the callback function.

Type bool

pass_job_queue

Determines whether job_queue will be passed to the callback function.

Type bool

Parameters

- **type** (*type*) – The type of updates this handler should process, as determined by isinstance
- **callback** (callable) – 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 for context based API:

```
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
- **pass_update_queue** (bool, optional) – If set to True, a keyword argument called update_queue will be passed to the callback function. It will be the Queue instance used by the *telegram.ext.Updater* and *telegram.ext.Dispatcher* that contains new updates which can be used to insert updates. Default is False. DEPRECATED: Please switch to context based callbacks.

- **pass_job_queue** (bool, optional) – If set to True, a keyword argument called `job_queue` will be passed to the callback function. It will be a `telegram.ext.JobQueue` instance created by the `telegram.ext.Updater` which can be used to schedule new jobs. Default is False. DEPRECATED: Please switch to context based callbacks.

check_update (*update*)

Determines whether an update should be passed to this handlers *callback*.

Parameters **update** (*telegram.Update*) – Incoming telegram update.

Returns bool

1.1.10 Persistence

telegram.ext.BasePersistence

class telegram.ext.**BasePersistence** (*store_user_data=True, store_chat_data=True*)

Bases: object

Interface class for adding persistence to your bot. Subclass this object for different implementations of a persistent bot.

All relevant methods must be overwritten. This means:

- If `store_chat_data` is True you must overwrite `get_chat_data()` and `update_chat_data()`.
- If `store_user_data` is True you must overwrite `get_user_data()` and `update_user_data()`.
- If you want to store conversation data with `telegram.ext.ConversationHandler`, you must overwrite `get_conversations()` and `update_conversation()`.
- `flush()` will be called when the bot is shutdown.

store_user_data

Optional, Whether user_data should be saved by this persistence class.

Type bool

store_chat_data

Optional. Whether chat_data should be saved by this persistence class.

Type bool

Parameters

- **store_user_data** (bool, optional) – Whether user_data should be saved by this persistence class. Default is True.
- **store_chat_data** (bool, optional) – Whether chat_data should be saved by this persistence class. Default is True .

flush ()

Will be called by `telegram.ext.Updater` upon receiving a stop signal. Gives the persistence a chance to finish up saving or close a database connection gracefully. If this is not of any importance just pass will be sufficient.

get_chat_data ()

“Will be called by `telegram.ext.Dispatcher` upon creation with a persistence object. It should return the chat_data if stored, or an empty defaultdict (dict).

Returns The restored chat data.

Return type defaultdict

get_conversations (*name*)

“Will be called by `telegram.ext.Dispatcher` 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`

get_user_data ()

“Will be called by `telegram.ext.Dispatcher` upon creation with a persistence object. It should return the user_data if stored, or an empty `defaultdict` (`dict`).

Returns The restored user data.

Return type `defaultdict`

update_chat_data (*chat_id*, *data*)

Will be called by the `telegram.ext.Dispatcher` after a handler has handled an update.

Parameters

- **chat_id** (`int`) – The chat the data might have been changed for.
- **data** (`dict`) – The :attr:‘`telegram.ext.dispatcher.chat_data`‘[*user_id*].

update_conversation (*name*, *key*, *new_state*)

Will be called when a `telegram.ext.ConversationHandler.update_state` is called. this allows the storage of the new state in the persistence.

Parameters

- **name** (`str`) – The handlers name.
- **key** (`tuple`) – The key the state is changed for.
- **new_state** (`tuple` | `any`) – The new state for the given key.

update_user_data (*user_id*, *data*)

Will be called by the `telegram.ext.Dispatcher` after a handler has handled an update.

Parameters

- **user_id** (`int`) – The user the data might have been changed for.
- **data** (`dict`) – The :attr:‘`telegram.ext.dispatcher.user_data`‘[*user_id*].

telegram.ext.PicklePersistence

```
class telegram.ext.PicklePersistence (filename,                               store_user_data=True,
                                     store_chat_data=True,                 single_file=True,
                                     on_flush=False)
```

Bases: `telegram.ext.basepersistence.BasePersistence`

Using python’s builtin pickle for making you bot persistent.

filename

The filename for storing the pickle files. When `single_file` is false this will be used as a prefix.

Type `str`

store_user_data

Optional. Whether user_data should be saved by this persistence class.

Type `bool`

store_chat_data

Optional. Whether user_data should be saved by this persistence class.

Type bool

single_file

Optional. When False will store 3 sperate files of *filename_user_data*, *filename_chat_data* and *filename_conversations*. Default is True.

Type bool

on_flush

When True will only save to file when *flush()* is called and keep data in memory until that happens. When False will store data on any transaction *and* on call fo *flush()*. Default is False.

Type bool, optional

Parameters

- **filename** (str) – The filename for storing the pickle files. When *single_file* is false this will be used as a prefix.
- **store_user_data** (bool, optional) – Whether user_data should be saved by this persistence class. Default is True.
- **store_chat_data** (bool, optional) – Whether user_data should be saved by this persistence class. Default is True.
- **single_file** (bool, optional) – When False will store 3 sperate files of *filename_user_data*, *filename_chat_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 fo *flush()*. Default is False.

flush()

Will save all data in memory to pickle file(s).

get_chat_data()

Returns the chat_data from the pickle file if it exists or an empty defaultdict.

Returns The restored chat data.

Return type defaultdict

get_conversations(name)

Returns the conversations from the pickle file if it exists or an empty defaultdict.

Parameters **name** (str) – The handlers name.

Returns The restored conversations for the handler.

Return type dict

get_user_data()

Returns the user_data from the pickle file if it exists or an empty defaultdict.

Returns The restored user data.

Return type defaultdict

update_chat_data(chat_id, data)

Will update the chat_data (if changed) 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 :attr:'telegram.ext.dispatcher.chat_data'[chat_id].

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 handlers name.
- **key** (*tuple*) – The key the state is changed for.
- **new_state** (*tuple | any*) – The new state for the given key.

update_user_data (*user_id, data*)

Will update the user_data (if changed) 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 :attr:telegram.ext.dispatcher.user_data'[user_id].

telegram.ext.DictPersistence

```
class telegram.ext.DictPersistence (store_user_data=True,          store_chat_data=True,
                                   user_data_json="",  chat_data_json="",  conversa-
                                   tions_json="")
```

Bases: telegram.ext.basepersistence.BasePersistence

Using python's dicts and json for making you bot persistent.

store_user_data

Whether user_data should be saved by this persistence class.

Type bool

store_chat_data

Whether chat_data should be saved by this persistence class.

Type bool

Parameters

- **store_user_data** (bool, optional) – Whether user_data should be saved by this persistence class. Default is True.
- **store_chat_data** (bool, optional) – Whether user_data should be saved by this persistence class. Default is True.
- **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 "".
- **conversations_json** (str, optional) – Json string that will be used to reconstruct conversation on creating this persistence. Default is "".

chat_data

The chat_data as a dict

Type dict

chat_data_json

The chat_data serialized as a JSON-string.

Type str

conversations

The conversations as a dict

Type dict

conversations_json

The conversations serialized as a JSON-string.

Type str

get_chat_data()

Returns the chat_data created from the chat_data_json or an empty defaultdict.

Returns The restored user data.

Return type defaultdict

get_conversations(name)

Returns the conversations created from the conversations_json or an empty defaultdict.

Returns The restored user data.

Return type defaultdict

get_user_data()

Returns the user_data created from the user_data_json or an empty defaultdict.

Returns The restored user data.

Return type defaultdict

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 :attr:‘telegram.ext.dispatcher.chat_data’[chat_id].

update_conversation(name, key, new_state)

Will update the conversations for the given handler.

Parameters

- **name** (str) – The handlers name.
- **key** (tuple) – The key the state is changed for.
- **new_state** (tuple | any) – The new state for the given key.

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 :attr:‘telegram.ext.dispatcher.user_data’[user_id].

user_data

The user_data as a dict

Type dict

user_data_json

The user_data serialized as a JSON-string.

Type str

1.2 telegram.utils package

1.2.1 telegram.utils.helpers Module

This module contains helper functions.

`telegram.utils.helpers.decode_conversations_from_json(json_string)`

Helper method to decode a conversations dict (that uses tuples as keys) from a JSON-string created with `_encode_conversations_to_json`.

Parameters `json_string` (`str`) – The conversations dict as JSON string.

Returns The conversations dict after decoding

Return type `dict`

`telegram.utils.helpers.decode_user_chat_data_from_json(data)`

Helper method to decode chat or user data (that uses ints as keys) from a JSON-string.

Parameters `data` (`str`) – The user/chat_data dict as JSON string.

Returns The user/chat_data defaultdict after decoding

Return type `dict`

`telegram.utils.helpers.effective_message_type(entity)`

Extracts the type of message as a string identifier from a `telegram.Message` or a `telegram.Update`.

Parameters `entity` (`Update | Message`) –

Returns One of `Message.MESSAGE_TYPES`

Return type `str`

`telegram.utils.helpers.encode_conversations_to_json(conversations)`

Helper method to encode a conversations dict (that uses tuples as keys) to a JSON-serializable way. Use `_decode_conversations_from_json` to decode.

Parameters `conversations` (`dict`) – The conversations dict to transform to JSON.

Returns The JSON-serialized conversations dict

Return type `str`

`telegram.utils.helpers.escape_markdown(text)`

Helper function to escape telegram markup symbols.

`telegram.utils.helpers.from_timestamp(unixtime)`

Parameters `unixtime` (`int`) –

Returns

Return type `datetime.datetime`

`telegram.utils.helpers.get_signal_name(signum)`

Returns the signal name of the given signal number.

`telegram.utils.helpers.mention_html(user_id, name)`

Parameters

- `user_id` (`int`) –
- `name` (`str`) –

Returns The inline mention for the user as html.

Return type `str`

`telegram.utils.helpers.mention_markdown(user_id, name)`

Parameters

- **user_id** (int) –
- **name** (str) –

Returns The inline mention for the user as markdown.

Return type str

`telegram.utils.helpers.to_timestamp(dt_obj)`

Parameters `dt_obj` (datetime.datetime) –

Returns

Return type int

1.2.2 telegram.utils.promise.Promise

class `telegram.utils.promise.Promise` (*pooled_function, args, kwargs*)

Bases: object

A simple Promise implementation for use with the `run_async` decorator, `DelayQueue` etc.

Parameters

- **pooled_function** (callable) – The callable that will be called concurrently.
- **args** (list | tuple) – Positional arguments for *pooled_function*.
- **kwargs** (dict) – Keyword arguments for *pooled_function*.

pooled_function

The callable that will be called concurrently.

Type callable

args

Positional arguments for *pooled_function*.

Type list | tuple

kwargs

Keyword arguments for *pooled_function*.

Type dict

done

Is set when the result is available.

Type `threading.Event`

exception

The exception raised by *pooled_function* or None if no exception has been raised (yet).

result (*timeout=None*)

Return the result of the Promise.

Parameters **timeout** (float, optional) – Maximum time in seconds to wait for the result to be calculated. None means indefinite. Default is None.

Returns Returns the return value of *pooled_function* or None if the timeout expires.

Raises Any exception raised by *pooled_function*.

run()

Calls the *pooled_function* callable.

1.2.3 telegram.utils.request.Request

```
class telegram.utils.request.Request (con_pool_size=1, proxy_url=None, url-  
lib3_proxy_kwargs=None, connect_timeout=5.0,  
read_timeout=5.0)
```

Bases: object

Helper class for python-telegram-bot which provides methods to perform POST & GET towards telegram servers.

Parameters

- **con_pool_size** (*int*) – Number of connections to keep in the connection pool.
- **proxy_url** (*str*) – The URL to the proxy server. For example: *http://127.0.0.1:3128*.
- **urllib3_proxy_kwargs** (*dict*) – Arbitrary arguments passed as-is to *urllib3.ProxyManager*. This value will be ignored if *proxy_url* is not set.
- **connect_timeout** (*int | float*) – The maximum amount of time (in seconds) to wait for a connection attempt to a server to succeed. None will set an infinite timeout for connection attempts. (default: 5.)
- **read_timeout** (*int | float*) – The maximum amount of time (in seconds) to wait between consecutive read operations for a response from the server. None will set an infinite timeout. This value is usually overridden by the various `telegram.Bot` methods. (default: 5.)

con_pool_size

The size of the connection pool used.

download (*url, filename, timeout=None*)

Download a file by its URL.

Parameters

- **url** (*str*) – The web location we want to retrieve.
- **timeout** – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

get (*url, timeout=None*)

Request an URL.

Parameters

- **url** (*str*) – The web location we want to retrieve.
- **timeout** (*int | float*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

Returns A JSON object.

post (*url, data, timeout=None*)

Request an URL.

Parameters

- **url** (*str*) – The web location we want to retrieve.
- **data** (*dict[str, str | int]*) – A dict of key/value pairs. Note: On py2.7 value is unicode.
- **timeout** (*int | float*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

Returns A JSON object.

retrieve (*url*, *timeout=None*)

Retrieve the contents of a file by its URL.

Parameters

- **url** (*str*) – The web location we want to retrieve.
- **timeout** (*int* | *float*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

1.3 telegram.Animation

class telegram.**Animation** (*file_id*, *width*, *height*, *duration*, *thumb=None*, *file_name=None*, *mime_type=None*, *file_size=None*, ***kwargs*)

Bases: telegram.base.TelegramObject

This object represents an animation file to be displayed in the message containing a game.

file_id

Unique file identifier.

Type *str*

width

Video width as defined by sender.

Type *int*

height

Video height as defined by sender.

Type *int*

duration

Duration of the video in seconds as defined by sender.

Type *int*

thumb

Optional. Animation thumbnail as defined by sender.

Type *telegram.PhotoSize*

file_name

Optional. Original animation filename as defined by sender.

Type *str*

mime_type

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

Type *str*

file_size

Optional. File size.

Type *int*

Parameters

- **file_id** (*str*) – Unique file identifier.
- **width** (*int*) – Video width as defined by sender.
- **height** (*int*) – Video height as defined by sender.
- **duration** (*int*) – Duration of the video in seconds as defined by sender.

- **thumb** (*telegram.PhotoSize*, optional) – Animation thumbnail as defined by sender.
- **file_name** (str, optional) – Original animation filename as defined by sender.
- **mime_type** (str, optional) – MIME type of the file as defined by sender.
- **file_size** (int, optional) – File size.

1.4 telegram.Audio

```
class telegram.Audio(file_id, duration, performer=None, title=None, mime_type=None,  
                    file_size=None, thumb=None, bot=None, **kwargs)
```

Bases: telegram.base.TelegramObject

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

file_id

Unique identifier for this file.

Type str

duration

Duration of the audio in seconds.

Type int

performer

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

Type str

title

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

Type str

mime_type

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

Type str

file_size

Optional. File size.

Type int

thumb

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

Type *telegram.PhotoSize*

bot

Optional. The Bot to use for instance methods.

Type *telegram.Bot*

Parameters

- **file_id** (str) – Unique identifier for this 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.
- **mime_type** (str, optional) – MIME type of the file as defined by sender.

- **file_size** (int, optional) – File size.
- **thumb** (*telegram.PhotoSize*, optional) – Thumbnail of the album cover to which the music file belongs
- **bot** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- ****kwargs** (dict) – Arbitrary keyword arguments.

get_file (*timeout=None*, ***kwargs*)

Convenience wrapper over *telegram.Bot.get_file*

Parameters

- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns *telegram.File*

Raises *telegram.TelegramError*

1.5 telegram.Bot

```
class telegram.Bot (token, base_url=None, base_file_url=None, request=None, private_key=None, private_key_password=None)
```

Bases: *telegram.base.TelegramObject*

This object represents a Telegram Bot.

Parameters

- **token** (str) – Bot's unique authentication.
- **base_url** (str, optional) – Telegram Bot API service URL.
- **base_file_url** (str, optional) – Telegram Bot API file URL.
- **request** (*telegram.utils.request.Request*, optional) – Pre initialized *telegram.utils.request.Request*.
- **private_key** (bytes, optional) – Private key for decryption of telegram passport data.
- **private_key_password** (bytes, optional) – Password for above private key.

```
addStickerToSet (user_id, name, png_sticker, emojis, mask_position=None, timeout=20, **kwargs)
```

Alias for *add_sticker_to_set*

```
add_sticker_to_set (user_id, name, png_sticker, emojis, mask_position=None, timeout=20, **kwargs)
```

Use this method to add a new sticker to a set created by the bot.

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

Parameters

- **user_id** (int) – User identifier of created sticker set owner.
- **name** (str) – Sticker set name.

- **png_sticker** (*str* | *filelike object*) – Png image with the sticker, must be up to 512 kilobytes in size, dimensions must not exceed 512px, and either width or height must be exactly 512px. Pass a *file_id* as a *String* to send a file that already exists on the Telegram servers, pass an HTTP URL as a *String* for Telegram to get a file from the Internet, or upload a new one using multipart/form-data.
- **emojis** (*str*) – One or more emoji corresponding to the sticker.
- **mask_position** (*telegram.MaskPosition*, optional) – Position where the mask should be placed on faces.
- **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns On success, *True* is returned.

Return type *bool*

Raises *telegram.TelegramError*

answerCallbackQuery (*callback_query_id*, *text=None*, *show_alert=False*, *url=None*,
cache_time=None, *timeout=None*, ***kwargs*)

Alias for *answer_callback_query*

answerInlineQuery (*inline_query_id*, *results*, *cache_time=300*, *is_personal=None*,
next_offset=None, *switch_pm_text=None*, *switch_pm_parameter=None*,
timeout=None, ***kwargs*)

Alias for *answer_inline_query*

answerPreCheckoutQuery (*pre_checkout_query_id*, *ok*, *error_message=None*, *timeout=None*,
***kwargs*)

Alias for *answer_pre_checkout_query*

answerShippingQuery (*shipping_query_id*, *ok*, *shipping_options=None*, *error_message=None*,
timeout=None, ***kwargs*)

Alias for *answer_shipping_query*

answer_callback_query (*callback_query_id*, *text=None*, *show_alert=False*, *url=None*,
cache_time=None, *timeout=None*, ***kwargs*)

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.

- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns bool On success, True is returned.

Raises telegram.TelegramError

answer_inline_query (inline_query_id, results, cache_time=300, is_personal=None, next_offset=None, switch_pm_text=None, switch_pm_parameter=None, timeout=None, **kwargs)

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

Parameters

- **inline_query_id** (str) – Unique identifier for the answered query.
- **results** (List[telegram.InlineQueryResult]) – A list of results for the inline query.
- **cache_time** (int, optional) – The maximum amount of time in seconds that the result of the inline query may be cached on the server. Defaults to 300.
- **is_personal** (bool, optional) – Pass True, if results may be cached on the server side only for the user that sent the query. By default, results may be returned to any user who sends the same query.
- **next_offset** (str, optional) – Pass the offset that a client should send in the next query with the same text to receive more results. Pass an empty string if there are no more results or if you don't support pagination. Offset length can't exceed 64 bytes.
- **switch_pm_text** (str, optional) – If passed, clients will display a button with specified text that switches the user to a private chat with the bot and sends the bot a start message with the parameter switch_pm_parameter.
- **switch_pm_parameter** (str, optional) – Deep-linking parameter for the /start message sent to the bot when user presses the switch button. 1-64 characters, only A-Z, a-z, 0-9, _ and - are allowed.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Example

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

Returns bool On success, True is returned.

Raises telegram.TelegramError

answer_pre_checkout_query (pre_checkout_query_id, ok, error_message=None, timeout=None, **kwargs)

Once the user has confirmed their payment and shipping details, the Bot API sends the final confirmation in the form of an Update with the field 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.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns On success, `True` is returned.

Return type `bool`

Raises `telegram.TelegramError`

answer_shipping_query (*shipping_query_id*, *ok*, *shipping_options=None*, *error_message=None*, *timeout=None*, ***kwargs*)

If you sent an invoice requesting a shipping address and the parameter `is_flexible` was specified, the Bot API will send an Update with a `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** (`List[telegram.ShippingOption]`) – Required if `ok` is True. A JSON-serialized array of available shipping options.
- **error_message** (`str`, optional) – Required if `ok` is False. Error message in human readable form that explains why it is impossible to complete the order (e.g. “Sorry, delivery to your desired address is unavailable”). Telegram will display this message to the user.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

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

Raises `telegram.TelegramError`

createNewStickerSet (*user_id*, *name*, *title*, *png_sticker*, *emojis*, *contains_masks=None*, *mask_position=None*, *timeout=20*, ***kwargs*)

Alias for `create_new_sticker_set`

create_new_sticker_set (*user_id, name, title, png_sticker, emojis, contains_masks=None, mask_position=None, timeout=20, **kwargs*)

Use this method to create new sticker set owned by a user.

The bot will be able to edit the created sticker set.

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

Parameters

- **user_id** (`int`) – User identifier of created sticker set owner.
- **name** (`str`) – Short name of sticker set, to be used in `t.me/addstickers/` URLs (e.g., `animals`). Can contain only english letters, digits and underscores. Must begin with a letter, can't contain consecutive underscores and must end in “`_by_<bot username>`”. `<bot_username>` is case insensitive. 1-64 characters.
- **title** (`str`) – Sticker set title, 1-64 characters.
- **png_sticker** (`str` | *filelike object*) – Png image with the sticker, must be up to 512 kilobytes in size, dimensions must not exceed 512px, and either width or height must be exactly 512px. Pass a `file_id` as a String to send a file that already exists on the Telegram servers, pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data.
- **emojis** (`str`) – One or more emoji corresponding to the sticker.
- **contains_masks** (`bool`, optional) – Pass True, if a set of mask stickers should be created.
- **mask_position** (`telegram.MaskPosition`, optional) – Position where the mask should be placed on faces.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns On success, `True` is returned.

Return type `bool`

Raises `telegram.TelegramError`

deleteChatPhoto (*chat_id, timeout=None, **kwargs*)

Alias for `delete_chat_photo`

deleteChatStickerSet (*chat_id, timeout=None, **kwargs*)

Alias for `delete_chat_sticker_set`

deleteMessage (*chat_id, message_id, timeout=None, **kwargs*)

Alias for `delete_message`

deleteStickerFromSet (*sticker, timeout=None, **kwargs*)

Alias for `delete_sticker_from_set`

deleteWebhook (*timeout=None, **kwargs*)

Alias for `delete_webhook`

delete_chat_photo (*chat_id, timeout=None, **kwargs*)

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`).
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments

Note: In regular groups (non-supergroups), this method will only work if the ‘All Members Are Admins’ setting is off in the target group.

Returns Returns `True` on success.

Return type `bool`

Raises `telegram.TelegramError`

delete_chat_sticker_set (`chat_id`, `timeout=None`, ****kwargs**)

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`).
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns `True` on success.

Return type `bool`

delete_message (`chat_id`, `message_id`, `timeout=None`, ****kwargs**)

Use this method to delete a message. A message can only be deleted if it was sent less than 48 hours ago. Any such recently sent outgoing message may be deleted. Additionally, if the bot is an administrator in a group chat, it can delete any message. If the bot is an administrator in a supergroup, it can delete messages from any other user and service messages about people joining or leaving the group (other types of service messages may only be removed by the group creator). In channels, bots can only remove their own messages.

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.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as
- **read timeout** (`the`) – from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns On success, `True` is returned.

Return type `bool`

Raises `telegram.TelegramError`

delete_sticker_from_set (*sticker*, *timeout=None*, ***kwargs*)

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

Parameters

- **sticker** (*str*) – File identifier of the sticker.
- **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns On success, `True` is returned.

Return type `bool`

Raises `telegram.TelegramError`

delete_webhook (*timeout=None*, ***kwargs*)

Use this method to remove webhook integration if you decide to switch back to `getUpdates`. Requires no parameters.

Parameters

- **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

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

Raises `telegram.TelegramError`

editMessageCaption (*chat_id=None*, *message_id=None*, *inline_message_id=None*, *caption=None*, *reply_markup=None*, *timeout=None*, *parse_mode=None*, ***kwargs*)

Alias for `edit_message_caption`

editMessageLiveLocation (*chat_id=None*, *message_id=None*, *inline_message_id=None*, *latitude=None*, *longitude=None*, *location=None*, *reply_markup=None*, *timeout=None*, ***kwargs*)

Alias for `edit_message_live_location`

editMessageMedia (*chat_id=None*, *message_id=None*, *inline_message_id=None*, *media=None*, *reply_markup=None*, *timeout=None*, ***kwargs*)

Alias for `edit_message_media`

editMessageReplyMarkup (*chat_id=None*, *message_id=None*, *inline_message_id=None*, *reply_markup=None*, *timeout=None*, ***kwargs*)

Alias for `edit_message_reply_markup`

editMessageText (*text*, *chat_id=None*, *message_id=None*, *inline_message_id=None*, *parse_mode=None*, *disable_web_page_preview=None*, *reply_markup=None*, *timeout=None*, ***kwargs*)

Alias for `edit_message_text`

edit_message_caption (*chat_id=None*, *message_id=None*, *inline_message_id=None*, *caption=None*, *reply_markup=None*, *timeout=None*, *parse_mode=None*, ***kwargs*)

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

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) – 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.
- **caption** (str, optional) – New caption of the message.
- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

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 `telegram.TelegramError`

```
edit_message_live_location(chat_id=None,          message_id=None,          in-  
                           line_message_id=None, latitude=None, longitude=None,  
                           location=None,      reply_markup=None,  timeout=None,  
                           **kwargs)
```

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 `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) – 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.
- **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.
- **location** (`telegram.Location`, optional) – The location to send.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

Returns On success the edited message.

Return type `telegram.Message`

edit_message_media (*chat_id=None, message_id=None, inline_message_id=None, media=None, reply_markup=None, timeout=None, **kwargs*)

Use this method to edit audio, document, photo, or video messages. If a message is a part of a message album, then it can be edited only to a photo or a video. Otherwise, message type can be changed arbitrarily. When inline message is edited, new file can't be uploaded. Use previously uploaded file via its `file_id` or specify a URL. On success, if the edited message was sent by the bot, the edited Message is returned, otherwise True is returned.

Parameters

- **chat_id** (*int | str*, optional) – 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.
- **inline_message_id** (*str*, optional) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.
- **media** (*telegram.InputMedia*) – An object for a new media content of the message.
- **reply_markup** (*telegram.ReplyMarkup*, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (*int | float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

edit_message_reply_markup (*chat_id=None, message_id=None, inline_message_id=None, reply_markup=None, timeout=None, **kwargs*)

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

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) – 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.
- **reply_markup** (*telegram.ReplyMarkup*, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (*int | float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

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

Return type *telegram.Message*

Raises *telegram.TelegramError*

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, timeout=None, **kwargs*)

Use this method to edit text and game messages sent by the bot or via the bot (for inline bots).

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) – 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.
- **text** (`str`) – New text of the message.
- **parse_mode** (`str`) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message. See the constants in `telegram.ParseMode` for the available modes.
- **disable_web_page_preview** (`bool`, optional) – Disables link previews for links in this message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

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 `telegram.TelegramError`

exportChatInviteLink (`chat_id`, `timeout=None`, ****kwargs**)

Alias for `export_chat_invite_link`

export_chat_invite_link (`chat_id`, `timeout=None`, ****kwargs**)

Use this method to export an invite link to 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`).
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments

Returns Exported invite link on success.

Return type `str`

Raises `telegram.TelegramError`

first_name

Bot's first name.

Type `str`

forwardMessage (`chat_id`, `from_chat_id`, `message_id`, `disable_notification=False`, `timeout=None`, ****kwargs**)

Alias for `forward_message`

forward_message (*chat_id, from_chat_id, message_id, disable_notification=False, timeout=None, **kwargs*)

Use this method to forward messages of any kind.

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).
- **disable_notification** (*bool*, optional) – Sends the message silently. Users will receive a notification with no sound.
- **message_id** (*int*) – Message identifier in the chat specified in *from_chat_id*.
- **timeout** (*int | float*, optional) – If this value is specified, use it as
- **read timeout** (*the*) – from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type *telegram.Message*

Raises *telegram.TelegramError*

getChat (*chat_id, timeout=None, **kwargs*)

Alias for *get_chat*

getChatAdministrators (*chat_id, timeout=None, **kwargs*)

Alias for *get_chat_administrators*

getChatMember (*chat_id, user_id, timeout=None, **kwargs*)

Alias for *get_chat_member*

getChatMembersCount (*chat_id, timeout=None, **kwargs*)

Alias for *get_chat_members_count*

getFile (*file_id, timeout=None, **kwargs*)

Alias for *get_file*

getGameHighScores (*user_id, chat_id=None, message_id=None, inline_message_id=None, timeout=None, **kwargs*)

Alias for *get_game_high_scores*

getMe (*timeout=None, **kwargs*)

Alias for *get_me*

getStickerSet (*name, timeout=None, **kwargs*)

Alias for *get_sticker_set*

getUpdates (*offset=None, limit=100, timeout=0, read_latency=2.0, allowed_updates=None, **kwargs*)

Alias for *get_updates*

getUserProfilePhotos (*user_id, offset=None, limit=100, timeout=None, **kwargs*)

Alias for *get_user_profile_photos*

getWebhookInfo (*timeout=None, **kwargs*)

Alias for *get_webhook_info*

get_chat (*chat_id, timeout=None, **kwargs*)

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).
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns `telegram.Chat`

Raises `telegram.TelegramError`

get_chat_administrators (*chat_id*, *timeout=None*, ***kwargs*)

Use this method to get a list of administrators in a chat. On success, returns an Array 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.

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns List[`telegram.ChatMember`]

Raises `telegram.TelegramError`

get_chat_member (*chat_id*, *user_id*, *timeout=None*, ***kwargs*)

Use this method to get information about a member of a 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.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns `telegram.ChatMember`

Raises `telegram.TelegramError`

get_chat_members_count (*chat_id*, *timeout=None*, ***kwargs*)

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

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns Number of members in the chat.

Return type int

Raises `telegram.TelegramError`

get_file (*file_id*, *timeout=None*, ***kwargs*)

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 20MB in size. The file can then be downloaded with `telegram.File.download`. It is guaranteed that the link will be valid for at least 1 hour. When the link expires, a new one can be requested by calling `get_file` again.

Parameters

- **file_id** (`str` | `telegram.Audio` | `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.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns `telegram.File`

Raises `telegram.TelegramError`

get_game_high_scores (*user_id*, *chat_id=None*, *message_id=None*, *inline_message_id=None*, *timeout=None*, ***kwargs*)

Use this method to get data for high score tables. Will return the score of the specified user and several of his neighbors in a game

Parameters

- **user_id** (`int`) – User identifier.
- **chat_id** (`int` | `str`, optional) – Required if `inline_message_id` is not specified. Unique identifier for the target chat.
- **message_id** (`int`, optional) – Required if `inline_message_id` is not specified. Identifier of the sent message.
- **inline_message_id** (`str`, optional) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns `List[telegram.GameHighScore]`

Raises `telegram.TelegramError`

get_me (*timeout=None*, ***kwargs*)

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

Parameters **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

Return type `telegram.User`

Raises `telegram.TelegramError`

get_sticker_set (*name*, *timeout=None*, ***kwargs*)

Use this method to get a sticker set.

Parameters

- **name** (`str`) – Short name of the sticker set that is used in `t.me/addstickers/` URLs (e.g., `animals`)
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns `telegram.StickerSet`

Raises `telegram.TelegramError`

get_updates (`offset=None`, `limit=100`, `timeout=0`, `read_latency=2.0`, `allowed_updates=None`, ****kwargs**)

Use this method to receive incoming updates using long polling.

Parameters

- **offset** (`int`, optional) – Identifier of the first update to be returned. Must be greater by one than the highest among the identifiers of previously received updates. By default, updates starting with the earliest unconfirmed update are returned. An update is considered confirmed as soon as `getUpdates` is called with an offset higher than its `update_id`. The negative offset can be specified to retrieve updates starting from -offset update from the end of the updates queue. All previous updates will forgotten.
- **limit** (`int`, optional) – Limits the number of updates to be retrieved. Values between 1-100 are accepted. Defaults to 100.
- **timeout** (`int`, optional) – Timeout in seconds for long polling. Defaults to 0, i.e. usual short polling. Should be positive, short polling should be used for testing purposes only.
- **allowed_updates** (`List[str]`, optional) – List the types of updates you want your bot to receive. For example, specify `["message", "edited_channel_post", "callback_query"]` to only receive updates of these types. See `telegram.Update` for a complete list of available update types. Specify an empty list to receive all updates regardless of type (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.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Notes

1. This method will not work if an outgoing webhook is set up.
2. In order to avoid getting duplicate updates, recalculate offset after each server response.
3. To take full advantage of this library take a look at `telegram.ext.Updater`

Returns `List[telegram.Update]`

Raises `telegram.TelegramError`

get_user_profile_photos (`user_id`, `offset=None`, `limit=100`, `timeout=None`, ****kwargs**)

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.

- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns `telegram.UserProfilePhotos`

Raises `telegram.TelegramError`

get_webhook_info (timeout=None, **kwargs)

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

If the bot is using getUpdates, will return an object with the url field empty.

Parameters

- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns `telegram.WebhookInfo`

id

Unique identifier for this bot.

Type int

kickChatMember (chat_id, user_id, timeout=None, until_date=None, **kwargs)

Alias for `kick_chat_member`

kick_chat_member (chat_id, user_id, timeout=None, until_date=None, **kwargs)

Use this method to kick a user from a group or a supergroup. In the case of supergroups, 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 group for this to work.

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.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **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.
- ****kwargs** (dict) – Arbitrary keyword arguments.

Note: In regular groups (non-supergroups), this method will only work if the ‘All Members Are Admins’ setting is off in the target group. Otherwise members may only be removed by the group’s creator or by the member that added them.

Returns bool On success, True is returned.

Raises `telegram.TelegramError`

last_name

Optional. Bot’s last name.

Type str

leaveChat (*chat_id*, *timeout=None*, ***kwargs*)

Alias for `leave_chat`

leave_chat (*chat_id*, *timeout=None*, ***kwargs*)

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).
- **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns *bool* On success, *True* is returned.

Raises `telegram.TelegramError`

name

Bot's @username.

Type *str*

pinChatMessage (*chat_id*, *message_id*, *disable_notification=None*, *timeout=None*, ***kwargs*)

Alias for `pin_chat_message`

pin_chat_message (*chat_id*, *message_id*, *disable_notification=None*, *timeout=None*, ***kwargs*)

Use this method to pin a message in a supergroup. 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).
- **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 group members about the new pinned message.
- **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments

Returns Returns *True* on success.

Return type *bool*

Raises `telegram.TelegramError`

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*, *timeout=None*, ***kwargs*)

Alias for `promote_chat_member`

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*, *timeout=None*, ***kwargs*)

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

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.
- **can_change_info** (`bool`, optional) – Pass True, if the administrator can change chat title, photo and other settings.
- **can_post_messages** (`bool`, optional) – Pass True, if the administrator can create channel posts, channels only.
- **can_edit_messages** (`bool`, optional) – Pass True, if the administrator can edit messages of other users, channels only.
- **can_delete_messages** (`bool`, optional) – Pass True, if the administrator can delete messages of other users.
- **can_invite_users** (`bool`, optional) – Pass True, if the administrator can invite new users to the chat.
- **can_restrict_members** (`bool`, optional) – Pass True, if the administrator can restrict, ban or unban chat members.
- **can_pin_messages** (`bool`, optional) – Pass True, if the administrator can pin messages, supergroups only.
- **can_promote_members** (`bool`, optional) – Pass True, if the administrator can add new administrators with a subset of his own privileges or demote administrators that he has promoted, directly or indirectly (promoted by administrators that were appointed by him).
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments

Returns Returns True on success.

Return type `bool`

Raises `telegram.TelegramError`

restrictChatMember (*chat_id*, *user_id*, *until_date=None*, *can_send_messages=None*,
can_send_media_messages=None, *can_send_other_messages=None*,
can_add_web_page_previews=None, *timeout=None*, ***kwargs*)

Alias for `restrict_chat_member`

restrict_chat_member (*chat_id*, *user_id*, *until_date=None*, *can_send_messages=None*,
can_send_media_messages=None, *can_send_other_messages=None*,
can_add_web_page_previews=None, *timeout=None*, ***kwargs*)

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.

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.
- **can_send_messages** (bool, optional) – Pass True, if the user can send text messages, contacts, locations and venues.
- **can_send_media_messages** (bool, optional) – Pass True, if the user can send audios, documents, photos, videos, video notes and voice notes, implies can_send_messages.
- **can_send_other_messages** (bool, optional) – Pass True, if the user can send animations, games, stickers and use inline bots, implies can_send_media_messages.
- **can_add_web_page_previews** (bool, optional) – Pass True, if the user may add web page previews to their messages, implies can_send_media_messages.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments

Returns Returns True on success.

Return type bool

Raises telegram.TelegramError

sendAnimation (chat_id, animation, duration=None, width=None, height=None, thumb=None, caption=None, parse_mode=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, **kwargs)

Alias for [send_animation](#)

sendAudio (chat_id, audio, duration=None, performer=None, title=None, caption=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, thumb=None, **kwargs)

Alias for [send_audio](#)

sendChatAction (chat_id, action, timeout=None, **kwargs)

Alias for [send_chat_action](#)

sendContact (chat_id, phone_number=None, first_name=None, last_name=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=None, contact=None, vcard=None, **kwargs)

Alias for [send_contact](#)

sendDocument (chat_id, document, filename=None, caption=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, thumb=None, **kwargs)

Alias for [send_document](#)

sendGame (chat_id, game_short_name, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=None, **kwargs)

Alias for [send_game](#)

sendInvoice (chat_id, title, description, payload, provider_token, start_parameter, currency, prices, 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=False, reply_to_message_id=None, reply_markup=None, provider_data=None, send_phone_number_to_provider=None, send_email_to_provider=None, timeout=None, **kwargs)

Alias for [send_invoice](#)

sendLocation (*chat_id*, *latitude=None*, *longitude=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=None*, *location=None*, *live_period=None*, ***kwargs*)

Alias for `send_location`

sendMediaGroup (*chat_id*, *media*, *disable_notification=None*, *reply_to_message_id=None*, *timeout=20*, ***kwargs*)

Alias for `send_media_group`

sendMessage (*chat_id*, *text*, *parse_mode=None*, *disable_web_page_preview=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=None*, ***kwargs*)

Alias for `send_message`

sendPhoto (*chat_id*, *photo*, *caption=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, *parse_mode=None*, ***kwargs*)

Alias for `send_photo`

sendSticker (*chat_id*, *sticker*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, ***kwargs*)

Alias for `send_sticker`

sendVenue (*chat_id*, *latitude=None*, *longitude=None*, *title=None*, *address=None*, *foursquare_id=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=None*, *venue=None*, *foursquare_type=None*, ***kwargs*)

Alias for `send_venue`

sendVideo (*chat_id*, *video*, *duration=None*, *caption=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, *width=None*, *height=None*, *parse_mode=None*, *supports_streaming=None*, *thumb=None*, ***kwargs*)

Alias for `send_video`

sendVideoNote (*chat_id*, *video_note*, *duration=None*, *length=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, *thumb=None*, ***kwargs*)

Alias for `send_video_note`

sendVoice (*chat_id*, *voice*, *duration=None*, *caption=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, *parse_mode=None*, ***kwargs*)

Alias for `send_voice`

send_animation (*chat_id*, *animation*, *duration=None*, *width=None*, *height=None*, *thumb=None*, *caption=None*, *parse_mode=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, ***kwargs*)

Use this method to send animation files (GIF or H.264/MPEG-4 AVC video without sound).

Parameters

- **chat_id** (*int* | *str*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **animation** (*str* | *filelike object* | *telegram.Animation*) – Animation to send. Pass a *file_id* as *String* to send an animation that exists on the Telegram servers (recommended), pass an *HTTP URL* as a *String* for Telegram to get an animation from the Internet, or upload a new animation using *multipart/form-data*. Lastly you can pass an existing *telegram.Animation* object to send.
- **duration** (*int*, optional) – Duration of sent animation in seconds.
- **width** (*int*, optional) – Animation width.
- **height** (*int*, optional) – Animation height.

- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not passed as a string or file_id.
- **caption** (*str*, optional) – Animation caption (may also be used when resending animations by file_id), 0-1024 characters.
- **parse_mode** (*str*, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **disable_notification** (*bool*, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (*int*, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (*int* | *float*, optional) – Send file timeout (default: 20 seconds).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

send_audio (*chat_id*, *audio*, *duration=None*, *performer=None*, *title=None*, *caption=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, *parse_mode=None*, *thumb=None*, ***kwargs*)

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 format. On success, the sent Message is returned. Bots can currently send audio files of up to 50 MB in size, this limit may be changed in the future.

For sending voice messages, use the `sendVoice` method instead.

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

Parameters

- **chat_id** (*int* | *str*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **audio** (*str* | *filelike object* | `telegram.Audio`) – Audio file to send. Pass a file_id as String to send an audio file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an audio file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing `telegram.Audio` object to send.
- **caption** (*str*, optional) – Audio caption, 0-1024 characters.
- **parse_mode** (*str*, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **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.
- **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (*telegram.ReplyMarkup*, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.
- **timeout** (int | float, optional) – Send file timeout (default: 20 seconds).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type *telegram.Message*

Raises *telegram.TelegramError*

send_chat_action (*chat_id*, *action*, *timeout=None*, ***kwargs*)

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).

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **action** (*telegram.ChatAction* | 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.ChatAction*
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns True on success.

Return type bool

Raises *telegram.TelegramError*

send_contact (*chat_id*, *phone_number=None*, *first_name=None*, *last_name=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=None*, *contact=None*, *vcard=None*, ***kwargs*)

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.
- **contact** (*telegram.Contact*, optional) – The contact to send.
- **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (*telegram.ReplyMarkup*, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type *telegram.Message*

Raises *telegram.TelegramError*

send_document (*chat_id*, *document*, *filename=None*, *caption=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, *parse_mode=None*, *thumb=None*, ***kwargs*)

Use this method to send general files.

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

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **document** (str | filelike object | *telegram.Document*) – File to send. Pass a file_id as String to send a file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing *telegram.Document* object to send.
- **filename** (str, optional) – File name that shows in telegram message (it is useful when you send file generated by temp module, for example). Undocumented.
- **caption** (str, optional) – Document caption (may also be used when resending documents by file_id), 0-1024 characters.
- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.
- **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.

- **reply_markup** (*telegram.ReplyMarkup*, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.
- **timeout** (int | float, optional) – Send file timeout (default: 20 seconds).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type *telegram.Message*

Raises *telegram.TelegramError*

send_game (*chat_id, game_short_name, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=None, **kwargs*)

Use this method to send a game.

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **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.
- **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (*telegram.ReplyMarkup*, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type *telegram.Message*

Raises *telegram.TelegramError*

send_invoice (*chat_id, title, description, payload, provider_token, start_parameter, currency, prices, 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=False, reply_to_message_id=None, reply_markup=None, provider_data=None, send_phone_number_to_provider=None, send_email_to_provider=None, timeout=None, **kwargs*)

Use this method to send invoices.

Parameters

- **chat_id** (int | str) – Unique identifier for the target private chat.
- **title** (str) – Product name.
- **description** (str) – Product description.

- **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.
- **start_parameter** (`str`) – Unique deep-linking parameter that can be used to generate this invoice when used as a start parameter.
- **currency** (`str`) – Three-letter ISO 4217 currency code.
- **prices** (`List[telegram.LabeledPrice]`) – Price breakdown, a list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.).
- **provider_data** (`str | object`, optional) – JSON-encoded 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.
- **reply_to_message_id** (`int`, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. An inlinekeyboard. If empty, one 'Pay total price' button will be shown. If not empty, the first button must be a Pay button.
- **timeout** (`int | float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

send_location(*chat_id*, *latitude=None*, *longitude=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=None*, *location=None*, *live_period=None*, ***kwargs*)

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.
- **location**(`telegram.Location`, optional) – The location to send.
- **live_period**(`int`, optional) – Period in seconds for which the location will be updated, should be between 60 and 86400.
- **disable_notification**(`bool`, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id**(`int`, optional) – If the message is a reply, ID of the original message.
- **reply_markup**(`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout**(`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs**(`dict`) – Arbitrary keyword arguments.

Returns On success, the sent `Message` is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

send_media_group(*chat_id*, *media*, *disable_notification=None*, *reply_to_message_id=None*, *timeout=20*, ***kwargs*)

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

Parameters

- **chat_id**(`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **media**(`List[telegram.InputMedia]`) – An array describing photos and videos to be sent, must include 2–10 items.
- **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.
- **timeout**(`int` | `float`, optional) – Send file timeout (default: 20 seconds).
- ****kwargs**(`dict`) – Arbitrary keyword arguments.

Returns An array of the sent `Messages`.

Return type `List[telegram.Message]`

Raises telegram.TelegramError

send_message (*chat_id*, *text*, *parse_mode=None*, *disable_web_page_preview=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=None*, ***kwargs*)

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. Also found as `telegram.constants.MAX_MESSAGE_LENGTH`.
- **parse_mode** (*str*) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message. See the constants in `telegram.ParseMode` for the available modes.
- **disable_web_page_preview** (*bool*, optional) – Disables link previews for links in this message.
- **disable_notification** (*bool*, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (*int*, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns On success, the sent message is returned.

Return type `telegram.Message`

Raises telegram.TelegramError

send_photo (*chat_id*, *photo*, *caption=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, *parse_mode=None*, ***kwargs*)

Use this method to send photos.

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

Parameters

- **chat_id** (*int* | *str*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **photo** (*str* | *filelike object* | `telegram.PhotoSize`) – Photo to send. Pass a `file_id` as String to send a photo that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a photo from the Internet, or upload a new photo using multipart/form-data. Lastly you can pass an existing `telegram.PhotoSize` object to send.
- **caption** (*str*, optional) – Photo caption (may also be used when resending photos by `file_id`), 0-1024 characters.

- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – Send file timeout (default: 20 seconds).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

send_sticker (*chat_id*, *sticker*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, ***kwargs*)

Use this method to send .webp stickers.

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

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **sticker** (str | filelike object `telegram.Sticker`) – Sticker to send. Pass a file_id as String to send a file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a .webp file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing `telegram.Sticker` object to send.
- **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – Send file timeout (default: 20 seconds).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

```
send_venue (chat_id, latitude=None, longitude=None, title=None, address=None,
            foursquare_id=None, disable_notification=False, reply_to_message_id=None,
            reply_markup=None, timeout=None, venue=None, foursquare_type=None,
            **kwargs)
```

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 optionally foursquare_type.

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”).
- **venue** ([telegram.Venue](#), optional) – The venue to send.
- **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.
- **reply_markup** ([telegram.ReplyMarkup](#), optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type [telegram.Message](#)

Raises [telegram.TelegramError](#)

```
send_video (chat_id, video, duration=None, caption=None, disable_notification=False,
            reply_to_message_id=None, reply_markup=None, timeout=20, width=None,
            height=None, parse_mode=None, supports_streaming=None, thumb=None,
            **kwargs)
```

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

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

Parameters

- **chat_id** (`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **video** (`str` | *filelike object* | `telegram.Video`) – Video file to send. Pass a `file_id` as `String` to send an video file that exists on the Telegram servers (recommended), pass an HTTP URL as a `String` for Telegram to get an video file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing `telegram.Video` object to send.
- **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.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **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.
- **reply_to_message_id** (`int`, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or `file_id`.
- **timeout** (`int` | `float`, optional) – Send file timeout (default: 20 seconds).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns On success, the sent `Message` is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

```
send_video_note(chat_id, video_note, duration=None, length=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, thumb=None, **kwargs)
```

Use this method to send video messages.

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

Parameters

- **chat_id** (`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **video_note** (`str` | *filelike object* | `telegram.VideoNote`) – Video note to send. Pass a `file_id` as `String` to send a video note that exists on the Telegram servers (recommended) or upload a new video using multipart/form-data. Or you can pass an

existing `telegram.VideoNote` object to send. Sending video notes by a URL is currently unsupported.

- **duration** (`int`, optional) – Duration of sent video in seconds.
- **length** (`int`, optional) – Video width and height
- **disable_notification** (`bool`, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (`int`, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or `file_id`.
- **timeout** (`int` | `float`, optional) – Send file timeout (default: 20 seconds).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns On success, the sent `Message` is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

send_voice (`chat_id`, `voice`, `duration=None`, `caption=None`, `disable_notification=False`, `reply_to_message_id=None`, `reply_markup=None`, `timeout=20`, `parse_mode=None`, ****kwargs**)

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).

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

Parameters

- **chat_id** (`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **voice** (`str` | *filelike object* | `telegram.Voice`) – Voice file to send. Pass a `file_id` as String to send an voice file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an voice file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing `telegram.Voice` object to send.
- **caption** (`str`, optional) – Voice message caption, 0-1024 characters.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **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.
- **reply_to_message_id** (`int`, optional) – If the message is a reply, ID of the original message.

- **reply_markup** (*telegram.ReplyMarkup*, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (*int* | *float*, optional) – Send file timeout (default: 20 seconds).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type *telegram.Message*

Raises *telegram.TelegramError*

setChatDescription (*chat_id*, *description*, *timeout=None*, ***kwargs*)

Alias for *set_chat_description*

setChatPhoto (*chat_id*, *photo*, *timeout=20*, ***kwargs*)

Alias for *set_chat_photo*

setChatStickerSet (*chat_id*, *sticker_set_name*, *timeout=None*, ***kwargs*)

Alias for *set_chat_sticker_set*

setChatTitle (*chat_id*, *title*, *timeout=None*, ***kwargs*)

Alias for *set_chat_title*

setGameScore (*user_id*, *score*, *chat_id=None*, *message_id=None*, *inline_message_id=None*, *force=None*, *disable_edit_message=None*, *timeout=None*, ***kwargs*)

Alias for *set_game_score*

setPassportDataErrors (*user_id*, *errors*, *timeout=None*, ***kwargs*)

Alias for *set_passport_data_errors*

setStickerPositionInSet (*sticker*, *position*, *timeout=None*, ***kwargs*)

Alias for *set_sticker_position_in_set*

setWebhook (*url=None*, *certificate=None*, *timeout=None*, *max_connections=40*, *allowed_updates=None*, ***kwargs*)

Alias for *set_webhook*

set_chat_description (*chat_id*, *description*, *timeout=None*, ***kwargs*)

Use this method to change the description of 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*) – New chat description, 1-255 characters.
- **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments

Returns Returns True on success.

Return type *bool*

Raises *telegram.TelegramError*

set_chat_photo (*chat_id*, *photo*, *timeout=20*, ***kwargs*)

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** (*filelike object*) – New chat photo.
- **timeout** (`int | float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments

Note: In regular groups (non-supergroups), this method will only work if the ‘All Members Are Admins’ setting is off in the target group.

Returns Returns True on success.

Return type `bool`

Raises `telegram.TelegramError`

set_chat_sticker_set (`chat_id, sticker_set_name, timeout=None, **kwargs`)

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.
- **timeout** (`int | float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns True on success.

Return type `bool`

set_chat_title (`chat_id, title, timeout=None, **kwargs`)

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, 1-255 characters.
- **timeout** (`int | float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments

Note: In regular groups (non-supergroups), this method will only work if the ‘All Members Are Admins’ setting is off in the target group.

Returns Returns `True` on success.

Return type `bool`

Raises `telegram.TelegramError`

set_game_score (*user_id*, *score*, *chat_id=None*, *message_id=None*, *inline_message_id=None*, *force=None*, *disable_edit_message=None*, *timeout=None*, ***kwargs*)

Use this method to set the score of the specified user in a game. On success, if the message was sent by the bot, returns the edited `Message`, otherwise returns `True`. Returns an error, if the new score is not greater than the user's current score in the chat and *force* is `False`.

Parameters

- **user_id** (`int`) – User identifier.
- **score** (`int`) – New score, must be non-negative.
- **force** (`bool`, optional) – Pass `True`, if the high score is allowed to decrease. This can be useful when fixing mistakes or banning cheaters
- **disable_edit_message** (`bool`, optional) – Pass `True`, if the game message should not be automatically edited to include the current scoreboard.
- **chat_id** (`int`/`str`, optional) – Required if *inline_message_id* is not specified. Unique identifier for the target chat.
- **message_id** (`int`, optional) – Required if *inline_message_id* is not specified. Identifier of the sent message.
- **inline_message_id** (`str`, optional) – Required if *chat_id* and *message_id* are not specified. Identifier of the inline message.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns The edited message, or if the message wasn't sent by the bot, `True`.

Return type `telegram.Message`

Raises

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

set_passport_data_errors (*user_id*, *errors*, *timeout=None*, ***kwargs*)

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

Use this if the data submitted by the user doesn't satisfy the standards your service requires for any reason. For example, if a birthday date seems invalid, a submitted document is blurry, a scan shows evidence of tampering, etc. Supply some details in the error message to make sure the user knows how to correct the issues.

Parameters

- **user_id** (`int`) – User identifier
- **errors** (`List`[`PassportElementError`]) – A JSON-serialized array describing the errors.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

Returns On success, True is returned.

Return type bool

Raises telegram.TelegramError

set_sticker_position_in_set (sticker, position, timeout=None, **kwargs)

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.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, True is returned.

Return type bool

Raises telegram.TelegramError

set_webhook (url=None, certificate=None, timeout=None, max_connections=40, allowed_updates=None, **kwargs)

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

If you'd like to make sure that the Webhook request comes from Telegram, we recommend using a secret path in the URL, e.g. <https://www.example.com/<token>>. Since nobody else knows your bot's token, you can be pretty sure it's us.

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

Parameters

- **url** (str) – HTTPS url to send updates to. Use an empty string to remove webhook integration.
- **certificate** (filelike) – Upload your public key certificate so that the root certificate in use can be checked. See our self-signed guide for details. (<https://goo.gl/rw7w6Y>)
- **max_connections** (int, optional) – Maximum allowed number of simultaneous HTTPS connections to the webhook for update delivery, 1-100. Defaults to 40. Use lower values to limit the load on your bot's server, and higher values to increase your bot's throughput.
- **allowed_updates** (List[str], optional) – List the types of updates you want your bot to receive. For example, specify ["message", "edited_channel_post", "callback_query"] to only receive updates of these types. See [telegram.Update](#) for a complete list of available update types. Specify an empty list to receive all updates regardless of type (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.

- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Note:

1. You will not be able to receive updates using `get_updates` for as long as an outgoing webhook is set up.
 2. To use a self-signed certificate, you need to upload your public key certificate using `certificate` parameter. Please upload as `InputFile`, sending a `String` will not work.
 3. Ports currently supported for Webhooks: 443, 80, 88, 8443.
-

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

Raises `telegram.TelegramError`

stopMessageLiveLocation (`chat_id=None`, `message_id=None`, `inline_message_id=None`, `reply_markup=None`, `timeout=None`, ****kwargs**)

Alias for `stop_message_live_location`

stop_message_live_location (`chat_id=None`, `message_id=None`, `inline_message_id=None`, `reply_markup=None`, `timeout=None`, ****kwargs**)

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`) – 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.
- **inline_message_id** (`str`, optional) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

Returns On success the edited message.

Return type `telegram.Message`

unbanChatMember (`chat_id`, `user_id`, `timeout=None`, ****kwargs**)

Alias for `unban_chat_member`

unban_chat_member (`chat_id`, `user_id`, `timeout=None`, ****kwargs**)

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

The user will not return to the group automatically, but will be able to join via link, etc. The bot must be an administrator in the group for this to work.

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.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns bool On success, True is returned.

Raises telegram.TelegramError

unpinChatMessage (chat_id, timeout=None, **kwargs)

Alias for [unpin_chat_message](#)

unpin_chat_message (chat_id, timeout=None, **kwargs)

Use this method to unpin a message in a supergroup. 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).
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments

Returns Returns True on success.

Return type bool

Raises telegram.TelegramError

uploadStickerFile (user_id, png_sticker, timeout=20, **kwargs)

Alias for [upload_sticker_file](#)

upload_sticker_file (user_id, png_sticker, timeout=20, **kwargs)

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

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

Parameters

- **user_id** (int) – User identifier of sticker file owner.
- **png_sticker** (str | filelike object) – Png image with the sticker, must be up to 512 kilobytes in size, dimensions must not exceed 512px, and either width or height must be exactly 512px.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns The uploaded File

Return type [telegram.File](#)

Raises telegram.TelegramError

username

Bot's username.

Type `str`

1.6 telegram.CallbackQuery

```
class telegram.CallbackQuery(id, from_user, chat_instance, message=None, data=None, in-  
                             line_message_id=None, game_short_name=None, bot=None,  
                             **kwargs)  
Bases: telegram.base.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.

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.
-

id
Unique identifier for this query.

Type `str`

from_user
Sender.

Type `telegram.User`

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

Type `telegram.Message`

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

Type `str`

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

Type `str`

data
Optional. Data associated with the callback button.

Type `str`

game_short_name
Optional. Short name of a Game to be returned.

Type `str`

Parameters

- **id** (`str`) – Unique identifier for this query.
- **from_user** (`telegram.User`) – Sender.
- **message** (`telegram.Message`, optional) – Message with the callback button that originated the query. Note that message content and message date will not be available if the message is too old.

- **inline_message_id** (`str`, optional) – Identifier of the message sent via the bot in inline mode, that originated the query.
- **chat_instance** (`str`, optional) – Global identifier, uniquely corresponding to the chat to which the message with the callback button was sent. Useful for high scores in games.
- **data** (`str`, optional) – Data associated with the callback button. Be aware that a bad client can send arbitrary data in this field.
- **game_short_name** (`str`, optional) – Short name of a Game to be returned, serves as the unique identifier for the game

Note: 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).

answer (**args*, ***kwargs*)

Shortcut for:

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

Returns On success, `True` is returned.

Return type `bool`

edit_message_caption (*caption*, **args*, ***kwargs*)

Shortcut for either:

```
bot.edit_message_caption(caption=caption,
                        chat_id=update.callback_query.message.chat_id,
                        message_id=update.callback_query.message.message_id,
                        *args, **kwargs)
```

or:

```
bot.edit_message_caption(caption=caption
                        inline_message_id=update.callback_query.inline_
↪message_id,
                        *args, **kwargs)
```

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

Return type `telegram.Message`

edit_message_reply_markup (*reply_markup*, **args*, ***kwargs*)

Shortcut for either:

```
bot.edit_message_replyMarkup(chat_id=update.callback_query.message.chat_id,
                             message_id=update.callback_query.message.
↪message_id,
                             reply_markup=reply_markup,
                             *args, **kwargs)
```

or:

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

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

Return type `telegram.Message`

edit_message_text (*text*, *args, **kwargs)

Shortcut for either:

```
bot.edit_message_text(text, chat_id=update.callback_query.message.chat_id,
                      message_id=update.callback_query.message.message_id,
                      *args, **kwargs)
```

or:

```
bot.edit_message_text(text, inline_message_id=update.callback_query.inline_
↪message_id,
                      *args, **kwargs)
```

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

Return type `telegram.Message`

1.7 telegram.Chat

class telegram.**Chat** (*id*, *type*, *title=None*, *username=None*, *first_name=None*, *last_name=None*, *all_members_are_administrators=None*, *bot=None*, *photo=None*, *description=None*, *invite_link=None*, *pinned_message=None*, *sticker_set_name=None*, *can_set_sticker_set=None*, **kwargs)

Bases: telegram.base.TelegramObject

This object represents a chat.

id

Unique identifier for this chat.

Type int

type

Type of chat.

Type str

title

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

Type str

username

Optional. Username.

Type str

first_name

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

Type str

last_name

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

Type `str`

all_members_are_administrators

Optional.

Type `bool`

photo

Optional. Chat photo.

Type `telegram.ChatPhoto`

description

Optional. Description, for supergroups and channel chats.

Type `str`

invite_link

Optional. Chat invite link, for supergroups and channel chats.

Type `str`

pinned_message

Optional. Pinned message, for supergroups. Returned only in `get_chat`.

Type `telegram.Message`

sticker_set_name

Optional. For supergroups, name of Group sticker set.

Type `str`

can_set_sticker_set

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

Type `bool`

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.
- **all_members_are_administrators** (`bool`, optional) – True if a group has *All Members Are Admins* enabled.
- **photo** (`telegram.ChatPhoto`, optional) – Chat photo. Returned only in `getChat`.
- **description** (`str`, optional) – Description, for supergroups and channel chats. Returned only in `get_chat`.
- **invite_link** (`str`, optional) – Chat invite link, for supergroups and channel chats. Returned only in `get_chat`.
- **pinned_message** (`telegram.Message`, optional) – Pinned message, for supergroups. Returned only in `get_chat`.

- **bot** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- **sticker_set_name** (str, optional) – For supergroups, name of Group sticker set. Returned only in `get_chat`.
- **can_set_sticker_set** (bool, optional) – True, if the bot can change group the sticker set. Returned only in `get_chat`.
- ****kwargs** (dict) – Arbitrary keyword arguments.

CHANNEL = 'channel'
 'channel'

 Type str

GROUP = 'group'
 'group'

 Type str

PRIVATE = 'private'
 'private'

 Type str

SUPERGROUP = 'supergroup'
 'supergroup'

 Type str

get_administrators (*args, **kwargs)
 Shortcut for:

```
bot.get_chat_administrators(update.message.chat.id, *args, **kwargs)
```

Returns A list of administrators in a chat. An Array of *telegram.ChatMember* objects that contains information about all chat administrators except other bots. If the chat is a group or a supergroup and no administrators were appointed, only the creator will be returned

Return type List[*telegram.ChatMember*]

get_member (*args, **kwargs)
 Shortcut for:

```
bot.get_chat_member(update.message.chat.id, *args, **kwargs)
```

Returns *telegram.ChatMember*

get_members_count (*args, **kwargs)
 Shortcut for:

```
bot.get_chat_members_count(update.message.chat.id, *args, **kwargs)
```

Returns int

kick_member (*args, **kwargs)
 Shortcut for:

```
bot.kick_chat_member(update.message.chat.id, *args, **kwargs)
```

Returns If the action was sent succesfully.

Return type bool

Note: This method will only work if the *All Members Are Admins* setting is off in the target group. Otherwise members may only be removed by the group's creator or by the member that added them.

leave (*args, **kwargs)

Shortcut for:

```
bot.leave_chat(update.message.chat.id, *args, **kwargs)
```

Returns bool If the action was sent successfully.

link

Convenience property. If the chat has a *username*, returns a t.me link of the chat.

Type str

send_action (*args, **kwargs)

Shortcut for:

```
bot.send_chat_action(update.message.chat.id, *args, **kwargs)
```

Returns If the action was sent successfully.

Return type bool

send_animation (*args, **kwargs)

Shortcut for:

```
bot.send_animation(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_audio (*args, **kwargs)

Shortcut for:

```
bot.send_audio(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_document (*args, **kwargs)

Shortcut for:

```
bot.send_document(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_message (*args, **kwargs)

Shortcut for:

```
bot.send_message(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_photo (*args, **kwargs)

Shortcut for:

```
bot.send_photo(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_sticker (*args, **kwargs)

Shortcut for:

```
bot.send_sticker(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_video (*args, **kwargs)

Shortcut for:

```
bot.send_video(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_video_note (*args, **kwargs)

Shortcut for:

```
bot.send_video_note(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_voice (*args, **kwargs)

Shortcut for:

```
bot.send_voice(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

unban_member (*args, **kwargs)

Shortcut for:

```
bot.unban_chat_member(update.message.chat.id, *args, **kwargs)
```

Returns If the action was sent successfully.

Return type bool

1.8 telegram.ChatAction

```
class telegram.ChatAction
    Bases: object

    Helper class to provide constants for different chatactions.

    FIND_LOCATION = 'find_location'
        'find_location'

        Type str

    RECORD_AUDIO = 'record_audio'
        'record_audio'

        Type str

    RECORD_VIDEO = 'record_video'
        'record_video'

        Type str

    RECORD_VIDEO_NOTE = 'record_video_note'
        'record_video_note'

        Type str

    TYPING = 'typing'
        'typing'

        Type str

    UPLOAD_AUDIO = 'upload_audio'
        'upload_audio'

        Type str

    UPLOAD_DOCUMENT = 'upload_document'
        'upload_document'

        Type str

    UPLOAD_PHOTO = 'upload_photo'
        'upload_photo'

        Type str

    UPLOAD_VIDEO = 'upload_video'
        'upload_video'

        Type str

    UPLOAD_VIDEO_NOTE = 'upload_video_note'
        'upload_video_note'

        Type str
```

1.9 telegram.ChatMember

```
class telegram.ChatMember(user, status, until_date=None, can_be_edited=None,  
                           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,  
                           can_send_messages=None, can_send_media_messages=None,  
                           can_send_other_messages=None, can_add_web_page_previews=None,  
                           **kwargs)
```

Bases: telegram.base.TelegramObject

This object contains information about one member of the chat.

user

Information about the user.

Type `telegram.User`

status

The member's status in the chat.

Type `str`

until_date

Optional. Date when restrictions will be lifted for this user.

Type `datetime.datetime`

can_be_edited

Optional. If the bot is allowed to edit administrator privileges of that user.

Type `bool`

can_change_info

Optional. If the administrator can change the chat title, photo and other settings.

Type `bool`

can_post_messages

Optional. If the administrator can post in the channel.

Type `bool`

can_edit_messages

Optional. If the administrator can edit messages of other users.

Type `bool`

can_delete_messages

Optional. If the administrator can delete messages of other users.

Type `bool`

can_invite_users

Optional. If the administrator can invite new users to the chat.

Type `bool`

can_restrict_members

Optional. If the administrator can restrict, ban or unban chat members.

Type `bool`

can_pin_messages

Optional. If the administrator can pin messages.

Type `bool`

can_promote_members

Optional. If the administrator can add new administrators.

Type `bool`

can_send_messages

Optional. If the user can send text messages, contacts, locations and venues.

Type `bool`

can_send_media_messages

Optional. If the user can send media messages, implies `can_send_messages`.

Type `bool`

can_send_other_messages

Optional. If the user can send animations, games, stickers and use inline bots, implies `can_send_media_messages`.

Type `bool`

can_add_web_page_previews

Optional. If user may add web page previews to his messages, implies `can_send_media_messages`

Type `bool`

Parameters

- **user** (`telegram.User`) – Information about the user.
- **status** (`str`) – The member's status in the chat. Can be 'creator', 'administrator', 'member', 'restricted', 'left' or 'kicked'.
- **until_date** (`datetime.datetime`, optional) – Restricted and kicked only. Date when restrictions will be lifted for this user.
- **can_be_edited** (`bool`, optional) – Administrators only. True, if the bot is allowed to edit administrator privileges of that user.
- **can_change_info** (`bool`, optional) – Administrators only. True, if the administrator can change the chat title, photo and other settings.
- **can_post_messages** (`bool`, optional) – Administrators only. True, if the administrator can post in the channel, channels only.
- **can_edit_messages** (`bool`, optional) – Administrators only. True, if the administrator can edit messages of other users, channels only.
- **can_delete_messages** (`bool`, optional) – Administrators only. True, if the administrator can delete messages of other user.
- **can_invite_users** (`bool`, optional) – Administrators only. True, if the administrator can invite new users to the chat.
- **can_restrict_members** (`bool`, optional) – Administrators only. True, if the administrator can restrict, ban or unban chat members.
- **can_pin_messages** (`bool`, optional) – Administrators only. True, if the administrator can pin messages, supergroups only.
- **can_promote_members** (`bool`, optional) – Administrators only. 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_send_messages** (`bool`, optional) – Restricted only. True, if the user can send text messages, contacts, locations and venues.

- **can_send_media_messages** (`bool`, optional) – Restricted only. True, if the user can send audios, documents, photos, videos, video notes and voice notes, implies `can_send_messages`.
- **can_send_other_messages** (`bool`, optional) – Restricted only. True, if the user can send animations, games, stickers and use inline bots, implies `can_send_media_messages`.
- **can_add_web_page_previews** (`bool`, optional) – Restricted only. True, if user may add web page previews to his messages, implies `can_send_media_messages`.

ADMINISTRATOR = 'administrator'
 'administrator'

 Type `str`

CREATOR = 'creator'
 'creator'

 Type `str`

KICKED = 'kicked'
 'kicked'

 Type `str`

LEFT = 'left'
 'left'

 Type `str`

MEMBER = 'member'
 'member'

 Type `str`

RESTRICTED = 'restricted'
 'restricted'

 Type `str`

1.10 telegram.ChatPhoto

class `telegram.ChatPhoto` (*small_file_id, big_file_id, bot=None, **kwargs*)
 Bases: `telegram.base.TelegramObject`

This object represents a chat photo.

small_file_id
 Unique file identifier of small (160x160) chat photo.

 Type `str`

big_file_id
 Unique file identifier of big (640x640) chat photo.

 Type `str`

Parameters

- **small_file_id** (`str`) – Unique file identifier of small (160x160) chat photo. This `file_id` can be used only for photo download.
- **big_file_id** (`str`) – Unique file identifier of big (640x640) chat photo. This `file_id` can be used only for photo download.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods

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

1.11 telegram.constants Module

Constants in the Telegram network.

The following constants were extracted from the [Telegram Bots FAQ](#).

```
telegram.constants.MAX_MESSAGE_LENGTH
4096
```

Type int

```
telegram.constants.MAX_CAPTION_LENGTH
1024
```

Type int

```
telegram.constants.SUPPORTED_WEBHOOK_PORTS
[443, 80, 88, 8443]
```

Type List[int]

```
telegram.constants.MAX_FILESIZE_DOWNLOAD
In bytes (20MB)
```

Type int

```
telegram.constants.MAX_FILESIZE_UPLOAD
In bytes (50MB)
```

Type int

```
telegram.constants.MAX_MESSAGES_PER_SECOND_PER_CHAT
1. Telegram may allow short bursts that go over this limit, but eventually you'll begin receiving 429 errors.
```

Type int

```
telegram.constants.MAX_MESSAGES_PER_SECOND
30
```

Type int

```
telegram.constants.MAX_MESSAGES_PER_MINUTE_PER_GROUP
20
```

Type int

```
telegram.constants.MAX_INLINE_QUERY_RESULTS
50
```

Type int

The following constant have been found by experimentation:

```
telegram.constants.MAX_MESSAGE_ENTITIES
100 (Beyond this cap telegram will simply ignore further formatting styles)
```

Type int

1.12 telegram.Contact

```
class telegram.Contact (phone_number, first_name, last_name=None, user_id=None,
                        vcard=None, **kwargs)
    Bases: telegram.base.TelegramObject
```

This object represents a phone contact.

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`

Parameters

- **phone_number** (`str`) – Contact's phone number.
- **first_name** (`str`) – Contact's first name.
- **last_name** (`str`, optional) – Contact's last name.
- **user_id** (`int`, optional) – Contact's user identifier in Telegram.
- **vcard** (`str`, optional) – Additional data about the contact in the form of a vCard.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.13 telegram.Document

```
class telegram.Document (file_id, thumb=None, file_name=None, mime_type=None,  
                          file_size=None, bot=None, **kwargs)
```

Bases: `telegram.base.TelegramObject`

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

file_id

Unique file identifier.

Type `str`

thumb

Optional. Document thumbnail.

Type `telegram.PhotoSize`

file_name

Original filename.

Type `str`

mime_type

Optional. MIME type of the file.

Type `str`

file_size

Optional. File size.

Type `int`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

Parameters

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

get_file (`timeout=None`, ****kwargs**)

Convenience wrapper over `telegram.Bot.get_file`

Parameters

- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns `telegram.File`

Raises `telegram.TelegramError`

1.14 telegram.error module

This module contains an object that represents Telegram errors.

exception `telegram.error.BadRequest` (`message`)

Bases: `telegram.error.NetworkError`

exception `telegram.error.ChatMigrated` (`new_chat_id`)

Bases: `telegram.error.TelegramError`

Parameters `new_chat_id` (`int`) –

exception `telegram.error.Conflict` (`msg`)

Bases: `telegram.error.TelegramError`

Raised when a long poll or webhook conflicts with another one.

Parameters `msg` (`str`) – The message from telegrams server.

exception `telegram.error.InvalidToken`

Bases: `telegram.error.TelegramError`

exception `telegram.error.NetworkError` (`message`)

Bases: `telegram.error.TelegramError`

exception `telegram.error.RetryAfter` (`retry_after`)

Bases: `telegram.error.TelegramError`

Parameters `retry_after` (`int`) –

exception telegram.error.TelegramError(*message*)

Bases: Exception

exception telegram.error.TimedOut

Bases: telegram.error.NetworkError

exception telegram.error.Unauthorized(*message*)

Bases: telegram.error.TelegramError

1.15 telegram.File

class telegram.File(*file_id*, *bot=None*, *file_size=None*, *file_path=None*, ***kwargs*)

Bases: telegram.base.TelegramObject

This object represents a file ready to be downloaded. The file can be downloaded with `download`. It is guaranteed that the link will be valid for at least 1 hour. When the link expires, a new one can be requested by calling `getFile`.

Note: Maximum file size to download is 20 MB

file_id

Unique identifier for this file.

Type str

file_size

Optional. File size.

Type str

file_path

Optional. File path. Use `download` to get the file.

Type str

Parameters

- **file_id** (str) – Unique identifier for this file.
- **file_size** (int, optional) – Optional. File size, if known.
- **file_path** (str, optional) – File path. Use `download` to get the file.
- **bot** (telegram.Bot, optional) – Bot to use with shortcut method.
- ****kwargs** (dict) – Arbitrary keyword arguments.

Note: If you obtain an instance of this class from `telegram.PassportFile.get_file`, then it will automatically be decrypted as it downloads when you call `download()`.

download (*custom_path=None*, *out=None*, *timeout=None*)

Download this file. By default, the file is saved in the current working directory with its original filename as reported by Telegram. If a `custom_path` is supplied, it will be saved to that path instead. If `out` is defined, the file contents will be saved to that object using the `out.write` method.

Note: `custom_path` and `out` are mutually exclusive.

Parameters

- **custom_path** (str, optional) – Custom path.
- **out** (io.BufferedWriter, optional) – A file-like object. Must be opened for writing in binary mode, if applicable.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

Returns The same object as `out` if specified. Otherwise, returns the filename downloaded to.

Return type `str | io.BufferedWriter`

Raises `ValueError` – If both `custom_path` and `out` are passed.

download_as_bytearray (*buf=None*)

Download this file and return it as a bytearray.

Parameters **buf** (bytearray, optional) – Extend the given bytearray with the downloaded data.

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

Return type `bytearray`

1.16 telegram.ForceReply

class `telegram.ForceReply` (*force_reply=True, selective=False, **kwargs*)

Bases: `telegram.replymarkup.ReplyMarkup`

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.

force_reply

Shows reply interface to the user.

Type `True`

selective

Optional. Force reply from specific users only.

Type `bool`

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 Message object
 - 2) if the bot's message is a reply (has `reply_to_message_id`), sender of the original message.
- ****kwargs** (dict) – Arbitrary keyword arguments.

1.17 telegram.InlineKeyboardButton

```
class telegram.InlineKeyboardButton (text,          url=None,          callback_data=None,
                                     switch_inline_query=None,
                                     switch_inline_query_current_chat=None,      call-
                                     back_game=None, pay=None, **kwargs)
```

Bases: telegram.base.TelegramObject

This object represents one button of an inline keyboard.

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.

text

Label text on the button.

Type `str`

url

Optional. HTTP url to be opened when button is pressed.

Type `str`

callback_data

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

Type `str`

switch_inline_query

Optional. Will prompt the user to select one of their chats, open that chat and insert the bot's username and the specified inline query in the input field.

Type `str`

switch_inline_query_current_chat

Optional. Will insert the bot's username and the specified inline query in the current chat's input field.

Type `str`

callback_game

Optional. Description of the game that will be launched when the user presses the button.

Type `telegram.CallbackGame`

pay

Optional. Specify True, to send a Pay button.

Type `bool`

Parameters

- **text** (`str`) – Label text on the button.
- **url** (`str`) – HTTP url to be opened when button is pressed.
- **callback_data** (`str`, optional) – Data to be sent in a callback query to the bot when button is pressed, 1-64 bytes.
- **switch_inline_query** (`str`, optional) – If set, pressing the button will prompt the user to select one of their chats, open that chat and insert the bot's username and the specified inline query in the input field. Can be empty, in which case just the bot's username will be inserted. This offers an easy way for users to start using your bot in inline mode when they are currently in a private chat with it. Especially useful when combined with `switch_pm*` actions - in this case the user will be automatically returned to the chat they switched from, skipping the chat selection screen.

- **switch_inline_query_current_chat** (`str`, optional) – If set, pressing the button will insert the bot’s username and the specified inline query in the current chat’s input field. Can be empty, in which case only the bot’s username will be inserted. This offers a quick way for the user to open your bot in inline mode in the same chat - good for selecting something from multiple options.
- **callback_game** (`telegram.CallbackGame`, optional) – Description of the game that will be launched when the user presses the button. This type of button must always be the `first` button in the first row.
- **pay** (`bool`, optional) – Specify `True`, to send a Pay button. This type of button must always be the `first` button in the first row.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.18 telegram.InlineKeyboardMarkup

class telegram.InlineKeyboardMarkup (*inline_keyboard*, ***kwargs*)

Bases: telegram.replymarkup.ReplyMarkup

This object represents an inline keyboard that appears right next to the message it belongs to.

inline_keyboard

Array of button rows, each represented by an Array of InlineKeyboardButton objects.

Type List[List[*telegram.InlineKeyboardButton*]]

Parameters

- **inline_keyboard** (List[List[*telegram.InlineKeyboardButton*]]) – Array of button rows, each represented by an Array of InlineKeyboardButton objects.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

classmethod from_button (*button*, ***kwargs*)

Shortcut for:

```
InlineKeyboardMarkup([[button]], **kwargs)
```

Return an InlineKeyboardMarkup from a single InlineKeyboardButton

Parameters

- **button** (*telegram.InlineKeyboardButton*) – The button to use in the markup
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

classmethod from_column (*button_column*, ***kwargs*)

Shortcut for:

```
InlineKeyboardMarkup([[button] for button in button_column], **kwargs)
```

Return an InlineKeyboardMarkup from a single column of InlineKeyboardButtons

Parameters

- **button_column** (List[*telegram.InlineKeyboardButton*]) – The button to use in the markup
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

classmethod from_row (*button_row*, ***kwargs*)

Shortcut for:

```
InlineKeyboardMarkup([button_row], **kwargs)
```

Return an `InlineKeyboardMarkup` from a single row of `InlineKeyboardButtons`

Parameters

- **button_row** (List[`telegram.InlineKeyboardButton`]) – The button to use in the markup
- ****kwargs** (dict) – Arbitrary keyword arguments.

1.19 telegram.InputFile

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

Bases: `object`

This object represents a Telegram `InputFile`.

input_file_content

The binary content of the file to send.

Type `bytes`

filename

Optional, Filename for the file to be sent.

Type `str`

attach

Optional, attach id for sending multiple files.

Type `str`

Parameters

- **obj** (`File handler`) – An open file descriptor.
- **filename** (`str`, optional) – Filename for this `InputFile`.
- **attach** (`bool`, optional) – Whether this should be send as one file or is part of a collection of files.

Raises `TelegramError`

static is_image (*stream*)

Check if the content file is an image by analyzing its headers.

Parameters **stream** (`str`) – A `str` representing the content of a file.

Returns The `str` mime-type of an image.

Return type `str`

1.20 telegram.InputMedia

class `telegram.InputMedia`

Bases: `telegram.base.TelegramObject`

Base class for Telegram `InputMedia` Objects.

See `telegram.InputMediaAnimation`, `telegram.InputMediaAudio`, `telegram.InputMediaDocument`, `telegram.InputMediaPhoto` and `telegram.InputMediaVideo` for detailed use.

1.21 telegram.InputMediaAnimation

```
class telegram.InputMediaAnimation(media, thumb=None, caption=None,  
                                   parse_mode=None, width=None, height=None,  
                                   duration=None)
```

Bases: telegram.files.inputmedia.InputMedia

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

type

animation.

Type str

media

File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing *telegram.Animation* object to send.

Type str

thumb

Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

Type filelike object

caption

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

Type str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.

Type str

width

Optional. Animation width.

Type int

height

Optional. Animation height.

Type int

duration

Optional. Animation duration.

Type int

Parameters

- **media** (str) – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing *telegram.Animation* object to send.
- **thumb** (filelike object, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.
- **caption** (str, optional) – Caption of the animation to be sent, 0-1024 characters.

- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in [telegram.ParseMode](#) for the available modes.
- **width** (int, optional) – Animation width.
- **height** (int, optional) – Animation height.
- **duration** (int, optional) – Animation duration.

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.

1.22 telegram.InputMediaAudio

class telegram.**InputMediaAudio** (*media*, *thumb=None*, *caption=None*, *parse_mode=None*,
duration=None, *performer=None*, *title=None*)

Bases: telegram.files.inputmedia.InputMedia

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

type

audio.

Type str

media

File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing [telegram.Audio](#) object to send.

Type str

caption

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

Type str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in [telegram.ParseMode](#) for the available modes.

Type str

duration

Duration of the audio in seconds.

Type int

performer

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

Type str

title

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

Type str

thumb

Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

Type *filelike object*

Parameters

- **media** (*str*) – File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.Document` object to send.
- **caption** (*str*, optional) – Caption of the audio to be sent, 0-1024 characters.
- **parse_mode** (*str*, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **duration** (*int*) – Duration of the audio in seconds as defined by sender.
- **performer** (*str*, optional) – Performer of the audio as defined by sender or by audio tags.
- **title** (*str*, optional) – Title of the audio as defined by sender or by audio tags.
- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not passed as a string or `file_id`.

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.

1.23 telegram.InputMediaDocument

```
class telegram.InputMediaDocument (media, thumb=None, caption=None,
                                   parse_mode=None)
```

Bases: `telegram.files.inputmedia.InputMedia`

Represents a general file to be sent.

type

`document.`

Type `str`

media

File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.Document` object to send.

Type `str`

caption

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

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

Type `str`

thumb

Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB

in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or `file_id`.

Type *filelike object*

Parameters

- **media** (`str`) – File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.Document` object to send.
- **caption** (`str`, optional) – Caption of the document to be sent, 0-1024 characters.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or `file_id`.

1.24 telegram.InputMediaPhoto

class `telegram.InputMediaPhoto` (*media*, *caption=None*, *parse_mode=None*)

Bases: `telegram.files.inputmedia.InputMedia`

Represents a photo to be sent.

type

`photo.`

Type `str`

media

File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.PhotoSize` object to send.

Type `str`

caption

Optional. Caption of the photo to be sent, 0-1024 characters.

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

Type `str`

Parameters

- **media** (`str`) – File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.PhotoSize` object to send.
- **caption** (`str`, optional) – Caption of the photo to be sent, 0-1024 characters.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

1.25 telegram.InputMediaVideo

```
class telegram.InputMediaVideo(media, caption=None, width=None, height=None, duration=None, supports_streaming=None, parse_mode=None,  
                               thumb=None)
```

Bases: telegram.files.inputmedia.InputMedia

Represents a video to be sent.

type

video.

Type str

media

File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing *telegram.Video* object to send.

Type str

caption

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

Type str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.

Type str

width

Optional. Video width.

Type int

height

Optional. Video height.

Type int

duration

Optional. Video duration.

Type int

supports_streaming

Optional. Pass True, if the uploaded video is suitable for streaming.

Type bool

thumb

Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

Type filelike object

Parameters

- **media** (str) – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing *telegram.Video* object to send.
- **caption** (str, optional) – Caption of the video to be sent, 0-1024 characters.

- **parse_mode** (*str*, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **width** (*int*, optional) – Video width.
- **height** (*int*, optional) – Video height.
- **duration** (*int*, optional) – Video duration.
- **supports_streaming** (*bool*, optional) – Pass True, if the uploaded video is suitable for streaming.
- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not passed as a string or `file_id`.

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.

1.26 telegram.KeyboardButton

```
class telegram.KeyboardButton(text,      request_contact=None,      request_location=None,
                               **kwargs)
    Bases: telegram.base.TelegramObject
```

This object represents one button of the reply keyboard. For simple text buttons String can be used instead of this object to specify text of the button.

Note: Optional fields are mutually exclusive.

text

Text of the button.

Type `str`

request_contact

Optional. If the user's phone number will be sent.

Type `bool`

request_location

Optional. If the user's current location will be sent.

Type `bool`

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.

Note: `request_contact` and `request_location` options will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

1.27 telegram.Location

class telegram.Location(*longitude, latitude, **kwargs*)

Bases: telegram.base.TelegramObject

This object represents a point on the map.

longitude

Longitude as defined by sender.

Type float

latitude

Latitude as defined by sender.

Type float

Parameters

- **longitude** (float) – Longitude as defined by sender.
- **latitude** (float) – Latitude as defined by sender.
- ****kwargs** (dict) – Arbitrary keyword arguments.

1.28 telegram.Message

class telegram.Message(*message_id, from_user, date, chat, 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=False, group_chat_created=False, supergroup_chat_created=False, channel_chat_created=False, 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, bot=None, **kwargs*)

Bases: telegram.base.TelegramObject

This object represents a message.

Note:

- In Python *from* is a reserved word, use *from_user* instead.
-

message_id

Unique message identifier inside this chat.

Type int

from_user

Optional. Sender.

Type *telegram.User*

date

Date the message was sent.

Type `datetime.datetime`

chat

Conversation the message belongs to.

Type `telegram.Chat`

forward_from

Optional. Sender of the original message.

Type `telegram.User`

forward_from_chat

Optional. Information about the original channel.

Type `telegram.Chat`

forward_from_message_id

Optional. Identifier of the original message in the channel.

Type `int`

forward_date

Optional. Date the original message was sent.

Type `datetime.datetime`

reply_to_message

Optional. The original message.

Type `telegram.Message`

edit_date

Optional. Date the message was last edited.

Type `datetime.datetime`

media_group_id

Optional. The unique identifier of a media message group this message belongs to.

Type `str`

text

Optional. The actual UTF-8 text of the message.

Type `str`

entities

Optional. Special entities like usernames, URLs, bot commands, etc. that appear in the text. See `Message.parse_entity` and `parse_entities` methods for how to use properly.

Type `List[telegram.MessageEntity]`

caption_entities

Optional. 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.

Type `List[telegram.MessageEntity]`

audio

Optional. Information about the file.

Type `telegram.Audio`

document

Optional. Information about the file.

Type `telegram.Document`

animation

For backward compatibility, when this field is set, the document field will also be set.

Type `telegram.Animation`

game

Optional. Information about the game.

Type `telegram.Game`

photo

Optional. Available sizes of the photo.

Type `List[telegram.PhotoSize]`

sticker

Optional. Information about the sticker.

Type `telegram.Sticker`

video

Optional. Information about the video.

Type `telegram.Video`

voice

Optional. Information about the file.

Type `telegram.Voice`

video_note

Optional. Information about the video message.

Type `telegram.VideoNote`

new_chat_members

Optional. Information about new members to the chat. (the bot itself may be one of these members).

Type `List[telegram.User]`

caption

Optional. Caption for the document, photo or video, 0-1024 characters.

Type `str`

contact

Optional. Information about the contact.

Type `telegram.Contact`

location

Optional. Information about the location.

Type `telegram.Location`

venue

Optional. Information about the venue.

Type `telegram.Venue`

left_chat_member

Optional. Information about the user that left the group. (this member may be the bot itself).

Type `telegram.User`

new_chat_title

Optional. A chat title was changed to this value.

Type `str`

new_chat_photo

Optional. A chat photo was changed to this value.

Type List[*telegram.PhotoSize*]

delete_chat_photo

Optional. The chat photo was deleted.

Type bool

group_chat_created

Optional. The group has been created.

Type bool

supergroup_chat_created

Optional. The supergroup has been created.

Type bool

channel_chat_created

Optional. The channel has been created.

Type bool

migrate_to_chat_id

Optional. The group has been migrated to a supergroup with the specified identifier.

Type int

migrate_from_chat_id

Optional. The supergroup has been migrated from a group with the specified identifier.

Type int

pinned_message

Optional. Specified message was pinned.

Type *telegram.message*

invoice

Optional. Information about the invoice.

Type *telegram.Invoice*

successful_payment

Optional. Information about the payment.

Type *telegram.SuccessfulPayment*

connected_website

Optional. The domain name of the website on which the user has logged in.

Type str

forward_signature

Optional. Signature of the post author for messages forwarded from channels.

Type str

author_signature

Optional. Signature of the post author for messages in channels.

Type str

passport_data

Optional. Telegram Passport data

Type *telegram.PassportData*

bot

Optional. The Bot to use for instance methods.

Type *telegram.Bot*

Parameters

- **message_id** (`int`) – Unique message identifier inside this chat.
- **from_user** (`telegram.User`, optional) – Sender, can be empty for messages sent to channels.
- **date** (`datetime.datetime`) – Date the message was sent in Unix time. Converted to `datetime.datetime`.
- **chat** (`telegram.Chat`) – Conversation the message belongs to.
- **forward_from** (`telegram.User`, optional) – For forwarded messages, sender of the original message.
- **forward_from_chat** (`telegram.Chat`, optional) – For messages forwarded from a channel, information about the original channel.
- **forward_from_message_id** (`int`, optional) – For forwarded channel posts, identifier of the original message in the channel.
- **forward_date** (`datetime.datetime`, optional) – For forwarded messages, date the original message was sent in Unix time. Converted to `datetime.datetime`.
- **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`.
- **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. Also found as `telegram.constants.MAX_MESSAGE_LENGTH`.
- **entities** (`List[telegram.MessageEntity]`, optional) – For text messages, special entities like usernames, URLs, bot commands, etc. that appear in the text. See `attr:parse_entity` and `attr:parse_entities` methods for how to use properly.
- **caption_entities** (`List[telegram.MessageEntity]`) – Optional. For Messages with a Caption. Special entities like usernames, URLs, bot commands, etc. that appear in the caption. See `Message.parse_caption_entity` and `parse_caption_entities` methods for how to use properly.
- **audio** (`telegram.Audio`, optional) – Message is an audio file, information about the file.
- **document** (`telegram.Document`, optional) – Message is a general file, information about the file.
- **animation** (`telegram.Animation`, optional) – Message is an animation, information about the animation. For backward compatibility, when this field is set, the `document` field will also be set.
- **game** (`telegram.Game`, optional) – Message is a game, information about the game.
- **photo** (`List[telegram.PhotoSize]`, optional) – Message is a photo, available sizes of the photo.
- **sticker** (`telegram.Sticker`, optional) – Message is a sticker, information about the sticker.
- **video** (`telegram.Video`, optional) – Message is a video, information about the video.

- **voice** (*telegram.Voice*, optional) – Message is a voice message, information about the file.
- **video_note** (*telegram.VideoNote*, optional) – Message is a video note, information about the video message.
- **new_chat_members** (List[*telegram.User*], optional) – New members that were added to the group or supergroup and information about them (the bot itself may be one of these members).
- **caption** (str, optional) – Caption for the document, photo or video, 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.
- **left_chat_member** (*telegram.User*, optional) – A member was removed from the group, information about them (this member may be the bot itself).
- **new_chat_title** (str, optional) – A chat title was changed to this value.
- **new_chat_photo** (List[*telegram.PhotoSize*], optional) – A chat photo was change to this value.
- **delete_chat_photo** (bool, optional) – Service message: The chat photo was deleted.
- **group_chat_created** (bool, optional) – Service message: The group has been created.
- **supergroup_chat_created** (bool, optional) – Service message: The supergroup has been created. This field can't be received in a message coming through updates, because bot can't be a member of a supergroup when it is created. It can only be found in *reply_to_message* if someone replies to a very first message in a directly created supergroup.
- **channel_chat_created** (bool, optional) – Service message: The channel has been created. This field can't be received in a message coming through updates, because bot can't be a member of a channel when it is created. It can only be found in *attr:reply_to_message* if someone replies to a very first message in a channel.
- **migrate_to_chat_id** (int, optional) – The group has been migrated to a supergroup with the specified identifier. This number may be greater than 32 bits and some programming languages may have difficulty/silent defects in interpreting it. But it is smaller than 52 bits, so a signed 64 bit integer or double-precision float type are safe for storing this identifier.
- **migrate_from_chat_id** (int, optional) – The supergroup has been migrated from a group with the specified identifier. This number may be greater than 32 bits and some programming languages may have difficulty/silent defects in interpreting it. But it is smaller than 52 bits, so a signed 64 bit integer or double-precision float type are safe for storing this identifier.
- **pinned_message** (*telegram.message*, optional) – Specified message was pinned. Note that the Message object in this field will not contain further *attr:reply_to_message* fields even if it is itself a reply.
- **invoice** (*telegram.Invoice*, optional) – Message is an invoice for a payment, information about the invoice.

- **successful_payment** (*telegram.SuccessfulPayment*, optional) – Message is a service message about a successful payment, information about the payment.
- **connected_website** (*str*, optional) – The domain name of the website on which the user has logged in.
- **forward_signature** (*str*, optional) – Signature of the post author for messages forwarded from channels.
- **author_signature** (*str*, optional) – Signature of the post author for messages in channels.
- **passport_data** (*telegram.PassportData*, optional) – Telegram Passport data

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.

Returns Message caption with caption entities formatted as HTML.

Return type *str*

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.

Returns Message caption with caption entities formatted as HTML.

Return type *str*

caption_markdown

Creates an Markdown-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 Markdown in the same way the original message was formatted.

Returns Message caption with caption entities formatted as Markdown.

Return type *str*

caption_markdown_urled

Creates an Markdown-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 Markdown. This also formats *telegram.MessageEntity.URL* as a hyperlink.

Returns Message caption with caption entities formatted as Markdown.

Return type *str*

chat_id

Shortcut for *telegram.Chat.id* for *chat*.

Type *int*

delete (**args, **kwargs*)

Shortcut for:

```
bot.delete_message(chat_id=message.chat_id,
                   message_id=message.message_id,
                   *args,
                   **kwargs)
```

Returns On success, *True* is returned.

Return type `bool`

edit_caption (**args*, ***kwargs*)

Shortcut for:

```
bot.edit_message_caption(chat_id=message.chat_id,
                        message_id=message.message_id,
                        *args,
                        **kwargs)
```

Note: You can only edit messages that the bot sent itself, therefore this method can only be used on the return value of the `bot.send_*` family of methods.

Returns On success, instance representing the edited message.

Return type `telegram.Message`

edit_media (*media*, **args*, ***kwargs*)

Shortcut for:

```
bot.edit_message_media(chat_id=message.chat_id,
                      message_id=message.message_id,
                      *args,
                      **kwargs)
```

Note: You can only edit messages that the bot sent itself, therefore this method can only be used on the return value of the `bot.send_*` family of methods.

Returns On success, instance representing the edited message.

Return type `telegram.Message`

edit_reply_markup (**args*, ***kwargs*)

Shortcut for:

```
bot.edit_message_reply_markup(chat_id=message.chat_id,
                             message_id=message.message_id,
                             *args,
                             **kwargs)
```

Note: You can only edit messages that the bot sent itself, therefore this method can only be used on the return value of the `bot.send_*` family of methods.

Returns On success, instance representing the edited message.

Return type `telegram.Message`

edit_text (**args*, ***kwargs*)

Shortcut for:

```
bot.edit_message_text(chat_id=message.chat_id,
                     message_id=message.message_id,
                     *args,
                     **kwargs)
```

Note: You can only edit messages that the bot sent itself, therefore this method can only be used on the return value of the `bot.send_*` family of methods.

Returns On success, instance representing the edited message.

Return type `telegram.Message`

effective_attachment

`telegram.Audio` or `telegram.Contact` or `telegram.Document` or `telegram.Animation` or `telegram.Game` or `telegram.Invoice` or `telegram.Location` or `List[telegram.PhotoSize]` or `telegram.Sticker` or `telegram.SuccessfulPayment` or `telegram.Venue` or `telegram.Video` or `telegram.VideoNote` or `telegram.Voice`: The attachment that this message was sent with. May be `None` if no attachment was sent.

forward (`chat_id`, `disable_notification=False`)

Shortcut for:

```
bot.forward_message(chat_id=chat_id,
                    from_chat_id=update.message.chat_id,
                    disable_notification=disable_notification,
                    message_id=update.message.message_id)
```

Returns On success, instance representing the message forwarded.

Return type `telegram.Message`

link

Convenience property. If the chat of the message is a supergroup or a channel and has a `Chat.username`, returns a t.me link of the message.

Type `str`

parse_caption_entities (`types=None`)

Returns a dict that maps `telegram.MessageEntity` to `str`. It contains entities from this message's caption filtered by their `telegram.MessageEntity.type` attribute as the key, and the text that each entity belongs to as the value of the dict.

Note: This method should always be used instead of the `caption_entities` attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See `parse_entity` for more info.

Parameters **types** (`List[str]`, optional) – List of `telegram.MessageEntity` types as strings. If the `type` attribute of an entity is contained in this list, it will be returned. Defaults to a list of all types. All types can be found as constants in `telegram.MessageEntity`.

Returns A dictionary of entities mapped to the text that belongs to them, calculated based on UTF-16 codepoints.

Return type `Dict[telegram.MessageEntity, str]`

parse_caption_entity (`entity`)

Returns the text from a given `telegram.MessageEntity`.

Note: This method is present because Telegram calculates the offset and length in UTF-16 codepoint pairs, which some versions of Python don't handle automatically. (That is, you can't just slice

`Message.caption` with the offset and length.)

Parameters

- **entity** (`telegram.MessageEntity`) – The entity to extract the text from. It must
- **an entity that belongs to this message.** (*be*) –

Returns The text of the given entity

Return type `str`

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
- **an entity that belongs to this message.** (*be*) –

Returns The text of the given entity

Return type `str`

reply_animation (**args, **kwargs*)

Shortcut for:

```
bot.send_animation(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (`bool`, optional) – If set to `True`, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: `True` in group chats and `False` in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

reply_audio (*args, **kwargs)

Shortcut for:

```
bot.send_audio(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

reply_contact (*args, **kwargs)

Shortcut for:

```
bot.send_contact(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

reply_document (*args, **kwargs)

Shortcut for:

```
bot.send_document(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

reply_html (*args, **kwargs)

Shortcut for:

```
bot.send_message(update.message.chat_id, parse_mode=ParseMode.HTML, *args,
↳ **kwargs)
```

Sends a message with HTML formatting.

Keyword Arguments **quote** (bool, optional) – If set to True, the message is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

reply_location (*args, **kwargs)

Shortcut for:

```
bot.send_location(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

reply_markdown (*args, **kwargs)

Shortcut for:

```
bot.send_message(update.message.chat_id, parse_mode=ParseMode.MARKDOWN,
↳ *args,
**kwargs)
```

Sends a message with markdown formatting.

Keyword Arguments **quote** (bool, optional) – If set to True, the message is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

reply_media_group (*args, **kwargs)

Shortcut for:

```
bot.reply_media_group(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the media group is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns An array of the sent Messages.

Return type List[`telegram.Message`]

Raises `telegram.TelegramError`

reply_photo (*args, **kwargs)

Shortcut for:

```
bot.send_photo(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

reply_sticker (*args, **kwargs)

Shortcut for:

```
bot.send_sticker(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

reply_text (*args, **kwargs)

Shortcut for:

```
bot.send_message(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the message is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

reply_vnuue (*args, **kwargs)

Shortcut for:

```
bot.send_vnuue(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

reply_video (*args, **kwargs)

Shortcut for:

```
bot.send_video(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

reply_video_note (*args, **kwargs)

Shortcut for:

```
bot.send_video_note(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

reply_voice (*args, **kwargs)

Shortcut for:

```
bot.send_voice(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

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.

Returns Message text with entities formatted as HTML.

Return type `str`

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.

Returns Message text with entities formatted as HTML.

Return type `str`

text_markdown

Creates an Markdown-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 Markdown in the same way the original message was formatted.

Returns Message text with entities formatted as Markdown.

Return type `str`

text_markdown_urled

Creates an Markdown-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 Markdown. This also formats `telegram.MessageEntity.URL` as a hyperlink.

Returns Message text with entities formatted as Markdown.

Return type `str`

1.29 telegram.MessageEntity

class `telegram.MessageEntity` (*type, offset, length, url=None, user=None, **kwargs*)

Bases: `telegram.base.TelegramObject`

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

type

Type of the entity.

Type `str`

offset

Offset in UTF-16 code units to the start of the entity.

Type `int`

length

Length of the entity in UTF-16 code units.

Type `int`

url

Optional. Url that will be opened after user taps on the text.

Type `str`

user

Optional. The mentioned user.

Type `telegram.User`

Parameters

- **type** (`str`) – Type of the entity. Can be mention (@username), hashtag, bot_command, url, email, bold (bold text), italic (italic text), code (monowidth string), pre (monowidth block), text_link (for clickable text URLs), text_mention (for users without usernames).
- **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 usertaps on the text.
- **user** (`telegram.User`, optional) – For “text_mention” only, the mentioned user.

ALL_TYPES = ['mention', 'hashtag', 'cashtag', 'phone_number', 'bot_command', 'url',
List of all the types.

Type List[`str`]

BOLD = 'bold'
'bold'

Type `str`

BOT_COMMAND = 'bot_command'
'bot_command'

Type `str`

CASHTAG = 'cashtag'
'cashtag'

Type `str`

CODE = 'code'
'code'

Type `str`

EMAIL = 'email'
'email'

Type `str`

HASHTAG = 'hashtag'
'hashtag'

Type `str`

ITALIC = 'italic'
'italic'

Type `str`

MENTION = 'mention'
'mention'

Type `str`

PHONE_NUMBER = 'phone_number'
'phone_number'

Type `str`

PRE = 'pre'
'pre'

```
        Type str
TEXT_LINK = 'text_link'
    'text_link'

        Type str
TEXT_MENTION = 'text_mention'
    'text_mention'

        Type str
URL = 'url'
    'url'

        Type str
```

1.30 telegram.ParseMode

```
class telegram.ParseMode
    Bases: object

    This object represents a Telegram Message Parse Modes.

    HTML = 'HTML'
        'HTML'

        Type str
    MARKDOWN = 'Markdown'
        'Markdown'

        Type str
```

1.31 telegram.PhotoSize

```
class telegram.PhotoSize (file_id, width, height, file_size=None, bot=None, **kwargs)
    Bases: telegram.base.TelegramObject

    This object represents one size of a photo or a file/sticker thumbnail.

    file_id
        Unique identifier for this file.

        Type str
    width
        Photo width.

        Type int
    height
        Photo height.

        Type int
    file_size
        Optional. File size.

        Type int
    bot
        Optional. The Bot to use for instance methods.

        Type telegram.Bot
```

Parameters

- **file_id** (`str`) – Unique identifier for this file.
- **width** (`int`) – Photo width.
- **height** (`int`) – Photo height.
- **file_size** (`int`, optional) – File size.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

get_file (*timeout=None*, ***kwargs*)

Convenience wrapper over `telegram.Bot.get_file`

Parameters

- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns `telegram.File`

Raises `telegram.TelegramError`

1.32 telegram.ReplyKeyboardRemove

class `telegram.ReplyKeyboardRemove` (*selective=False*, ***kwargs*)

Bases: `telegram.replymarkup.ReplyMarkup`

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`).

remove_keyboard

Requests clients to remove the custom keyboard.

Type `True`

selective

Optional. Use this parameter if you want to remove the keyboard for specific users only.

Type `bool`

Example

A user votes in a poll, bot returns confirmation message in reply to the vote and removes the keyboard for that user, while still showing the keyboard with poll options to users who haven't voted yet.

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 Message object
 - 2) if the bot's message is a reply (has `reply_to_message_id`), sender of the original message.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.33 telegram.ReplyKeyboardMarkup

```
class telegram.ReplyKeyboardMarkup (keyboard, resize_keyboard=False,
                                   one_time_keyboard=False, selective=False,
                                   **kwargs)
```

Bases: telegram.replymarkup.ReplyMarkup

This object represents a custom keyboard with reply options.

keyboard

Array of button rows.

Type List[List[*telegram.KeyboardButton* | str]]

resize_keyboard

Optional. Requests clients to resize the keyboard.

Type bool

one_time_keyboard

Optional. Requests clients to hide the keyboard as soon as it's been used.

Type bool

selective

Optional. Show the keyboard to specific users only.

Type bool

Example

A user requests to change the bot's language, bot replies to the request with a keyboard to select the new language. Other users in the group don't see the keyboard.

Parameters

- **keyboard** (List[List[str | *telegram.KeyboardButton*]]) – Array of button rows, each represented by an Array of *telegram.KeyboardButton* objects.
- **resize_keyboard** (bool, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to false, in which case the custom keyboard is always of the same height as the app's standard keyboard. Defaults to False
- **one_time_keyboard** (bool, optional) – Requests clients to hide the keyboard as soon as it's been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to False.
- **selective** (bool, optional) – Use this parameter if you want to show the keyboard to specific users only. Targets:
 - 1) users that are @mentioned in the text of the Message object
 - 2) if the bot's message is a reply (has reply_to_message_id), sender of the original message.
 Defaults to False.
- ****kwargs** (dict) – Arbitrary keyword arguments.

```
classmethod from_button (button, resize_keyboard=False, one_time_keyboard=False, selective=False, **kwargs)
```

Shortcut for:

```
ReplyKeyboardMarkup ([ [button]], **kwargs)
```

Return an 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. Defaults to *False*
- **one_time_keyboard** (*bool*, optional) – Requests clients to hide the keyboard as soon as it's been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to *False*.
- **selective** (*bool*, optional) – Use this parameter if you want to show the keyboard to specific users only. Targets:
 - 1) users that are @mentioned in the text of the Message object
 - 2) if the bot's message is a reply (has *reply_to_message_id*), sender of the original message.Defaults to *False*.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

classmethod from_column (*button_column*, *resize_keyboard=False*,
one_time_keyboard=False, *selective=False*, ***kwargs*)

Shortcut for:

```
ReplyKeyboardMarkup([button for button in button_column], **kwargs)
```

Return an ReplyKeyboardMarkup from a single column of KeyboardButtons

Parameters

- **button_column** (*List[telegram.KeyboardButton | str]*) – The button to use in the markup
- **resize_keyboard** (*bool*, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to false, in which case the custom keyboard is always of the same height as the app's standard keyboard. Defaults to *False*
- **one_time_keyboard** (*bool*, optional) – Requests clients to hide the keyboard as soon as it's been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to *False*.
- **selective** (*bool*, optional) – Use this parameter if you want to show the keyboard to specific users only. Targets:
 - 1) users that are @mentioned in the text of the Message object
 - 2) if the bot's message is a reply (has *reply_to_message_id*), sender of the original message.Defaults to *False*.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

classmethod from_row (*button_row*, *resize_keyboard=False*, *one_time_keyboard=False*, *selective=False*, ***kwargs*)

Shortcut for:

```
ReplyKeyboardMarkup([button_row], **kwargs)
```

Return an ReplyKeyboardMarkup from a single row of KeyboardButtons

Parameters

- **button_row** (List[[telegram.KeyboardButton](#) | str]) – The button to use in the markup
- **resize_keyboard** (bool, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to false, in which case the custom keyboard is always of the same height as the app’s standard keyboard. Defaults to False
- **one_time_keyboard** (bool, optional) – Requests clients to hide the keyboard as soon as it’s been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to False.
- **selective** (bool, optional) – Use this parameter if you want to show the keyboard to specific users only. Targets:
 - 1) users that are @mentioned in the text of the Message object
 - 2) if the bot’s message is a reply (has `reply_to_message_id`), sender of the original message.Defaults to False.
- ****kwargs** (dict) – Arbitrary keyword arguments.

1.34 telegram.ReplyMarkup

class telegram.ReplyMarkup

Bases: telegram.base.TelegramObject

Base class for Telegram ReplyMarkup Objects.

See [telegram.ReplyKeyboardMarkup](#) and [telegram.InlineKeyboardMarkup](#) for detailed use.

1.35 telegram.TelegramObject

class telegram.TelegramObject

Bases: object

Base class for most telegram objects.

to_json()

Returns str

1.36 telegram.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*, ***kwargs*)

Bases: telegram.base.TelegramObject

This object represents an incoming update.

Note: At most one of the optional parameters can be present in any given update.

update_id

The update's unique identifier.

Type `int`

message

Optional. New incoming message.

Type `telegram.Message`

edited_message

Optional. New version of a message.

Type `telegram.Message`

channel_post

Optional. New incoming channel post.

Type `telegram.Message`

edited_channel_post

Optional. New version of a channel post.

Type `telegram.Message`

inline_query

Optional. New incoming inline query.

Type `telegram.InlineQuery`

chosen_inline_result

Optional. The result of an inline query that was chosen by a user.

Type `telegram.ChosenInlineResult`

callback_query

Optional. New incoming callback query.

Type `telegram.CallbackQuery`

shipping_query

Optional. New incoming shipping query.

Type `telegram.ShippingQuery`

pre_checkout_query

Optional. New incoming pre-checkout query.

Type `telegram.PreCheckoutQuery`

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.
- **message** (`telegram.Message`, optional) – New incoming message of any kind - text, photo, sticker, etc.
- **edited_message** (`telegram.Message`, optional) – New version of a message that is known to the bot and was edited.
- **channel_post** (`telegram.Message`, optional) – New incoming channel post of any kind - text, photo, sticker, etc.

- **edited_channel_post** (*telegram.Message*, optional) – New version of a channel post that is known to the bot and was edited.
- **inline_query** (*telegram.InlineQuery*, optional) – New incoming inline query.
- **chosen_inline_result** (*telegram.ChosenInlineResult*, optional) – The result of an inline query that was chosen by a user and sent to their chat partner.
- **callback_query** (*telegram.CallbackQuery*, optional) – New incoming callback query.
- **shipping_query** (*telegram.ShippingQuery*, optional) – New incoming shipping query. Only for invoices with flexible price.
- **pre_checkout_query** (*telegram.PreCheckoutQuery*, optional) – New incoming pre-checkout query. Contains full information about checkout
- ****kwargs** (dict) – Arbitrary keyword arguments.

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

effective_chat

The chat that this update was sent in, no matter what kind of update this is. Will be `None` for *inline_query*, *chosen_inline_result*, *callback_query* from inline messages, *shipping_query* and *pre_checkout_query*.

Type *telegram.Chat*

effective_message

The message included in this update, no matter what kind of update this is. Will be `None` for *inline_query*, *chosen_inline_result*, *callback_query* from inline messages, *shipping_query* and *pre_checkout_query*.

Type *telegram.Message*

effective_user

The user that sent this update, no matter what kind of update this is. Will be `None` for *channel_post*.

Type *telegram.User*

1.37 telegram.User

class `telegram.User` (*id*, *first_name*, *is_bot*, *last_name=None*, *username=None*, *language_code=None*, *bot=None*, ***kwargs*)

Bases: `telegram.base.TelegramObject`

This object represents a Telegram user or bot.

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`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

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.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.

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

classmethod `de_list` (`data`, `bot`)

full_name

Convenience property. The user's `first_name`, followed by (if available) `last_name`.

Type `str`

get_profile_photos (`*args`, `**kwargs`)

Shortcut for:

```
bot.get_user_profile_photos(update.message.from_user.id, *args, **kwargs)
```

link

Convenience property. If `username` is available, returns a t.me link of the user.

Type `str`

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`)

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.

Return type `str`

name

Convenience property. If available, returns the user's `username` prefixed with "@". If `username` is not available, returns `full_name`.

Type `str`

send_animation (**args*, ***kwargs*)

Shortcut for:

```
bot.send_animation(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

send_audio (**args*, ***kwargs*)

Shortcut for:

```
bot.send_audio(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

send_document (**args*, ***kwargs*)

Shortcut for:

```
bot.send_document(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

send_message (**args*, ***kwargs*)

Shortcut for:

```
bot.send_message(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

send_photo (**args*, ***kwargs*)

Shortcut for:

```
bot.send_photo(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

send_sticker (**args*, ***kwargs*)

Shortcut for:

```
bot.send_sticker(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

send_video (*args, **kwargs)

Shortcut for:

```
bot.send_video(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_video_note (*args, **kwargs)

Shortcut for:

```
bot.send_video_note(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_voice (*args, **kwargs)

Shortcut for:

```
bot.send_voice(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

1.38 telegram.UserProfilePhotos

class telegram.**UserProfilePhotos** (total_count, photos, **kwargs)

Bases: telegram.base.TelegramObject

This object represent a user's profile pictures.

total_count

Total number of profile pictures.

Type int

photos

Requested profile pictures.

Type List[List[*telegram.PhotoSize*]]

Parameters

- **total_count** (int) – Total number of profile pictures the target user has.
- **photos** (List[List[*telegram.PhotoSize*]]) – Requested profile pictures (in up to 4 sizes each).

1.39 telegram.Venue

class telegram.**Venue** (location, title, address, foursquare_id=None, foursquare_type=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents a venue.

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`

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

1.40 telegram.Video

```
class telegram.Video(file_id, width, height, duration, thumb=None, mime_type=None,  
                    file_size=None, bot=None, **kwargs)
```

Bases: `telegram.base.TelegramObject`

This object represents a video file.

file_id

Unique identifier for this file.

Type `str`

width

Video width as defined by sender.

Type `int`

height

Video height as defined by sender.

Type `int`

duration

Duration of the video in seconds as defined by sender.

Type `int`

thumb

Optional. Video thumbnail.

Type `telegram.PhotoSize`

mime_type

Optional. Mime type of a file as defined by sender.

Type `str`

file_size

Optional. File size.

Type `int`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

Parameters

- **file_id** (`str`) – Unique identifier for this file.
- **width** (`int`) – Video width as defined by sender.
- **height** (`int`) – Video height as defined by sender.
- **duration** (`int`) – Duration of the video in seconds as defined by sender.
- **thumb** (`telegram.PhotoSize`, optional) – Video thumbnail.
- **mime_type** (`str`, optional) – Mime type of a file as defined by sender.
- **file_size** (`int`, optional) – File size.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

get_file (`timeout=None`, `**kwargs`)

Convenience wrapper over `telegram.Bot.get_file`

Parameters

- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns `telegram.File`

Raises `telegram.TelegramError`

1.41 telegram.VideoNote

class `telegram.VideoNote` (`file_id`, `length`, `duration`, `thumb=None`, `file_size=None`, `bot=None`, `**kwargs`)

Bases: `telegram.base.TelegramObject`

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

file_id

Unique identifier for this file.

Type `str`

length

Video width and height as defined by sender.

Type `int`

duration

Duration of the video in seconds as defined by sender.

Type `int`

thumb

Optional. Video thumbnail.

Type `telegram.PhotoSize`

file_size

Optional. File size.

Type `int`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

Parameters

- **file_id** (`str`) – Unique identifier for this file.
- **length** (`int`) – Video width and height as defined by sender.
- **duration** (`int`) – Duration of the video in seconds as defined by sender.
- **thumb** (`telegram.PhotoSize`, optional) – Video thumbnail.
- **file_size** (`int`, optional) – File size.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

get_file (`timeout=None`, `**kwargs`)

Convenience wrapper over `telegram.Bot.get_file`

Parameters

- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns `telegram.File`

Raises `telegram.TelegramError`

1.42 telegram.Voice

class `telegram.Voice` (`file_id`, `duration`, `mime_type=None`, `file_size=None`, `bot=None`, `**kwargs`)

Bases: `telegram.base.TelegramObject`

This object represents a voice note.

file_id

Unique identifier for this 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.

Type `int`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

Parameters

- **file_id** (`str`) – Unique identifier for this file.
- **duration** (`int`, optional) – Duration of the audio in seconds as defined by sender.
- **mime_type** (`str`, optional) – MIME type of the file as defined by sender.
- **file_size** (`int`, optional) – File size.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

get_file (`timeout=None`, `**kwargs`)

Convenience wrapper over `telegram.Bot.get_file`

Parameters

- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns `telegram.File`

Raises `telegram.TelegramError`

1.43 telegram.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, **kwargs)
```

Bases: `telegram.base.TelegramObject`

This object represents a Telegram WebhookInfo.

Contains information about the current status of a webhook.

url

Webhook URL.

Type `str`

has_custom_certificate

If a custom certificate was provided for webhook.

Type `bool`

pending_update_count

Number of updates awaiting delivery.

Type `int`

last_error_date

Optional. Unix time for the most recent error that happened.

Type `int`

last_error_message

Optional. Error message in human-readable format.

Type `str`

max_connections

Optional. Maximum allowed number of simultaneous HTTPS connections.

Type `int`

allowed_updates

Optional. A list of update types the bot is subscribed to.

Type `List[str]`

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.
- **last_error_date** (`int`, optional) – Unix time for the most recent error that happened when trying to deliver an update via webhook.
- **last_error_message** (`str`, optional) – Error message in human-readable format for the most recent error that happened when trying to deliver an update via webhook.
- **max_connections** (`int`, optional) – Maximum allowed number of simultaneous HTTPS connections to the webhook for update delivery.
- **allowed_updates** (`List[str]`, optional) – A list of update types the bot is subscribed to. Defaults to all update types.

1.44 Stickers

1.44.1 telegram.Sticker

class `telegram.Sticker` (*file_id, width, height, thumb=None, emoji=None, file_size=None, set_name=None, mask_position=None, bot=None, **kwargs*)

Bases: `telegram.base.TelegramObject`

This object represents a sticker.

file_id

Unique identifier for this file.

Type `str`

width

Sticker width.

Type `int`

height

Sticker height.

Type `int`

thumb

Optional. Sticker thumbnail in the .webp or .jpg format.

Type `telegram.PhotoSize`

emoji

Optional. Emoji associated with the sticker.

Type `str`

set_name

Optional. Name of the sticker set to which the sticker belongs.

Type `str`

mask_position

Optional. For mask stickers, the position where the mask should be placed.

Type `telegram.MaskPosition`

file_size

Optional. File size.

Type `int`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

Parameters

- **file_id** (`str`) – Unique identifier for this file.
- **width** (`int`) – Sticker width.
- **height** (`int`) – Sticker height.
- **thumb** (`telegram.PhotoSize`, optional) – Sticker thumbnail in the .webp or .jpg format.
- **emoji** (`str`, optional) – Emoji associated with the sticker
- **set_name** (`str`, optional) – Name of the sticker set to which the sticker belongs.
- **mask_position** (`telegram.MaskPosition`, optional) – For mask stickers, the position where the mask should be placed.
- **file_size** (`int`, optional) – File size.
- **(obj (**kwargs) – dict)**: Arbitrary keyword arguments.⁷
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.

get_file (`timeout=None`, `**kwargs`)

Convenience wrapper over `telegram.Bot.get_file`

Parameters

- **timeout** (`int | float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns `telegram.File`

Raises telegram.TelegramError

1.44.2 telegram.StickerSet

class telegram.**StickerSet** (*name, title, contains_masks, stickers, bot=None, **kwargs*)

Bases: telegram.base.TelegramObject

This object represents a sticker set.

name

Sticker set name.

Type str

title

Sticker set title.

Type str

contains_masks

True, if the sticker set contains masks.

Type bool

stickers

List of all set stickers.

Type List[telegram.Sticker]

Parameters

- **name** (str) – Sticker set name.
- **title** (str) – Sticker set title.
- **contains_masks** (bool) – True, if the sticker set contains masks.
- **stickers** (List[telegram.Sticker]) – List of all set stickers.

1.44.3 telegram.MaskPosition

class telegram.**MaskPosition** (*point, x_shift, y_shift, scale, **kwargs*)

Bases: telegram.base.TelegramObject

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

point

The part of the face relative to which the mask should be placed.

Type str

x_shift

Shift by X-axis measured in widths of the mask scaled to the face size, from left to right.

Type float

y_shift

Shift by Y-axis measured in heights of the mask scaled to the face size, from top to bottom.

Type float

scale

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

Type float

Notes

`type` should be one of the following: *forehead*, *eyes*, *mouth* or *chin*. You can use the class constants for those.

Parameters

- **point** (`str`) – The part of the face relative to which the mask should be placed.
- **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.

```
CHIN = 'chin'
      'chin'

      Type str
```

```
EYES = 'eyes'
      'eyes'

      Type str
```

```
FOREHEAD = 'forehead'
          'forehead'

          Type str
```

```
MOUTH = 'mouth'
        'mouth'

        Type str
```

1.45 Inline Mode

1.45.1 telegram.InlineQuery

```
class telegram.InlineQuery (id, from_user, query, offset, location=None, bot=None, **kwargs)
    Bases: telegram.base.TelegramObject
```

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

Note:

- In Python *from* is a reserved word, use *from_user* instead.
-

id
Unique identifier for this query.

```
      Type str
```

from_user
Sender.

```
      Type telegram.User
```

location
Optional. Sender location, only for bots that request user location.

Type `telegram.Location`

query

Text of the query (up to 512 characters).

Type `str`

offset

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

Type `str`

Parameters

- **id** (`str`) – Unique identifier for this query.
- **from_user** (`telegram.User`) – Sender.
- **location** (`telegram.Location`, optional) – Sender location, only for bots that request user location.
- **query** (`str`) – Text of the query (up to 512 characters).
- **offset** (`str`) – Offset of the results to be returned, can be controlled by the bot.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

answer (**args*, ***kwargs*)

Shortcut for:

```
bot.answer_inline_query(update.inline_query.id, *args, **kwargs)
```

Parameters

- **results** (List[`telegram.InlineQueryResult`]) – A list of results for the inline query.
- **cache_time** (`int`, optional) – The maximum amount of time in seconds that the result of the inline query may be cached on the server. Defaults to 300.
- **is_personal** (`bool`, optional) – Pass True, if results may be cached on the server side only for the user that sent the query. By default, results may be returned to any user who sends the same query.
- **next_offset** (`str`, optional) – Pass the offset that a client should send in the next query with the same text to receive more results. Pass an empty string if there are no more results or if you don't support pagination. Offset length can't exceed 64 bytes.
- **switch_pm_text** (`str`, optional) – If passed, clients will display a button with specified text that switches the user to a private chat with the bot and sends the bot a start message with the parameter `switch_pm_parameter`.
- **switch_pm_parameter** (`str`, optional) – Deep-linking parameter for the /start message sent to the bot when user presses the switch button. 1-64 characters, only A-Z, a-z, 0-9, _ and - are allowed.

1.45.2 telegram.InlineQueryResult

class `telegram.InlineQueryResult` (*type*, *id*, ***kwargs*)

Bases: `telegram.base.TelegramObject`

Baseclass for the `InlineQueryResult*` classes.

type

Type of the result.

Type `str`

id

Unique identifier for this result, 1-64 Bytes.

Type `str`

Parameters

- **type** (`str`) – Type of the result.
- **id** (`str`) – Unique identifier for this result, 1-64 Bytes.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.45.3 telegram.InlineQueryResultArticle

```
class telegram.InlineQueryResultArticle(id, title, input_message_content,  
                                         reply_markup=None, url=None,  
                                         hide_url=None, description=None,  
                                         thumb_url=None, thumb_width=None,  
                                         thumb_height=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.InlineQueryResult`

This object represents a Telegram InlineQueryResultArticle.

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.ReplyMarkup`

url

Optional. URL of the result.

Type `str`

hide_url

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

Type `bool`

description

Optional. Short description of the result.

Type `str`

thumb_url

Optional. Url of the thumbnail for the result.

Type str

thumb_width

Optional. Thumbnail width.

Type int

thumb_height

Optional. Thumbnail height.

Type int

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.ReplyMarkup*, optional) – Inline keyboard attached to the message
- **url** (str, optional) – URL of the result.
- **hide_url** (bool, optional) – Pass True, if you don't want the URL to be shown in the message.
- **description** (str, optional) – Short description of the result.
- **thumb_url** (str, optional) – Url of the thumbnail for the result.
- **thumb_width** (int, optional) – Thumbnail width.
- **thumb_height** (int, optional) – Thumbnail height.
- ****kwargs** (dict) – Arbitrary keyword arguments.

1.45.4 telegram.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, **kwargs)
```

Bases: telegram.inline.inlinequeryresult.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.

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. Caption, 0-200 characters.

Type `str`

audio_duration
Optional. Performer.

Type `str`

caption
Optional. Audio duration in seconds.

Type `str`

parse_mode
Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

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 audio.

Type `telegram.InputMessageContent`

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) – Caption, 0-200 characters.
- **audio_duration** (`str`, optional) – Performer.
- **caption** (`str`, optional) – Audio duration in seconds.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the audio.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.45.5 telegram.InlineQueryResultCachedAudio

```
class telegram.InlineQueryResultCachedAudio(id, audio_file_id, caption=None,
                                             reply_markup=None, in-
                                             put_message_content=None,
                                             parse_mode=None, **kwargs)
Bases: telegram.inline.inlinequeryresult.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.

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

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

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 audio.

Type `telegram.InputMessageContent`

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
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the audio.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.45.6 telegram.InlineQueryResultCachedDocument

```
class telegram.InlineQueryResultCachedDocument (id, title, document_file_id, de-  
                                                scription=None, caption=None,  
                                                reply_markup=None, in-  
                                                put_message_content=None,  
                                                parse_mode=None, **kwargs)
```

Bases: telegram.inline.inlinequeryresult.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.

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, 0-1024 characters

Type str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption.. See the constants in *telegram.ParseMode* for the available modes.

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*

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, 0-1024 characters
- **parse_mode** (*str*, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption.. See the constants in *telegram.ParseMode* for the available modes.
- **reply_markup** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- **input_message_content** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the file.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

1.45.7 telegram.InlineQueryResultCachedGif

```
class telegram.InlineQueryResultCachedGif(id, gif_file_id, title=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)
```

Bases: *telegram.inline.inlinequeryresult.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.

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, 0-1024 characters

Type *str*

parse_mode
Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.

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 gif.

Type *telegram.InputMessageContent*

Parameters

- **id** (*str*) – Unique identifier for this result, 1-64 bytes.
- **gif_file_id** (*str*) – A valid file identifier for the GIF file.
- **title** (*str*, optional) – Title for the result. *caption* (*str*, optional):
- **caption** (*str*, optional) – Caption, 0-1024 characters
- **parse_mode** (*str*, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in [telegram.ParseMode](#) for the available modes.
- **reply_markup** ([telegram.InlineKeyboardMarkup](#), optional) – Inline keyboard attached to the message.
- **input_message_content** ([telegram.InputMessageContent](#), optional) – Content of the message to be sent instead of the gif.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

1.45.8 telegram.InlineQueryResultCachedMpeg4Gif

```
class telegram.InlineQueryResultCachedMpeg4Gif (id, mpeg4_file_id, title=None, caption=None, reply_markup=None,  
                                                input_message_content=None,  
                                                parse_mode=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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, 0-1024 characters

Type *str*

parse_mode

Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in [telegram.ParseMode](#) for the available modes.

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 MPEG-4 file.

Type `telegram.InputMessageContent`

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, 0-1024 characters
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the MPEG-4 file.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.45.9 telegram.InlineQueryResultCachedPhoto

```
class telegram.InlineQueryResultCachedPhoto(id, photo_file_id, title=None, description=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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, 0-1024 characters

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

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 photo.

Type `telegram.InputMessageContent`

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, 0-1024 characters
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the photo.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.45.10 telegram.InlineQueryResultCachedSticker

```
class telegram.InlineQueryResultCachedSticker(id, sticker_file_id, re-  
                                              ply_markup=None, in-  
                                              put_message_content=None,  
                                              **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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`

Parameters

- **id**(`str`) –
- **sticker_file_id**(`str`) –
- **reply_markup**(`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content**(`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the sticker.
- ****kwargs**(`dict`) – Arbitrary keyword arguments.

1.45.11 telegram.InlineQueryResultCachedVideo

```
class telegram.InlineQueryResultCachedVideo(id, video_file_id, title, description=None,  
                                             caption=None, reply_markup=None,  
                                             input_message_content=None,  
                                             parse_mode=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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, 0-1024 characters.

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

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.

Type `telegram.InputMessageContent`

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, 0-1024 characters.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the video.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.45.12 telegram.InlineQueryResultCachedVoice

```
class telegram.InlineQueryResultCachedVoice(id, voice_file_id, title, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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.

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in [telegram.ParseMode](#) for the available modes.

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 voice.

Type [telegram.InputMessageContent](#)

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.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in [telegram.ParseMode](#) for the available modes.
- **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.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.45.13 telegram.InlineQueryResultContact

```
class telegram.InlineQueryResultContact (id, phone_number, first_name,
                                          last_name=None, reply_markup=None,
                                          input_message_content=None,
                                          thumb_url=None, thumb_width=None,
                                          thumb_height=None, vcard=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

type

'contact'.

Type `str`

id
Unique identifier for this result, 1-64 bytes.
Type `str`

phone_number
Contact's phone number.
Type `str`

first_name
Contact's first name.
Type `str`

last_name
Optional. Contact's last name.
Type `str`

vcard
Optional. Additional data about the contact in the form of a vCard, 0-2048 bytes.
Type `str`

reply_markup
Optional. Inline keyboard attached to the message.
Type `telegram.InlineKeyboardMarkup`

input_message_content
Optional. Content of the message to be sent instead of the contact.
Type `telegram.InputMessageContent`

thumb_url
Optional. Url of the thumbnail for the result.
Type `str`

thumb_width
Optional. Thumbnail width.
Type `int`

thumb_height
Optional. Thumbnail height.
Type `int`

Parameters

- **id** (`str`) – Unique identifier for this result, 1-64 bytes.
- **phone_number** (`str`) – Contact's phone number.
- **first_name** (`str`) – Contact's first name.
- **last_name** (`str`, optional) – Contact's last name.
- **vcard** (`str`, optional) – Additional data about the contact in the form of a vCard, 0-2048 bytes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the contact.
- **thumb_url** (`str`, optional) – Url of the thumbnail for the result.
- **thumb_width** (`int`, optional) – Thumbnail width.

- **thumb_height** (int, optional) – Thumbnail height.
- ****kwargs** (dict) – Arbitrary keyword arguments.

1.45.14 telegram.InlineQueryResultDocument

```
class telegram.InlineQueryResultDocument (id, document_url, title, mime_type,
                                           caption=None, description=None,
                                           reply_markup=None, input_message_content=None,
                                           thumb_url=None, thumb_width=None,
                                           thumb_height=None, parse_mode=None,
                                           **kwargs)
```

Bases: telegram.inline.inlinequeryresult.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.

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, 0-1024 characters

Type str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

Type str

document_url

A valid URL for the file.

Type str

mime_type

Mime type of the content of the file, either “application/pdf” or “application/zip”.

Type str

description

Optional. Short description of the result.

Type str

reply_markup

Optional. Inline keyboard attached to the message.

Type `telegram.InlineKeyboardMarkup`

input_message_content

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

Type *telegram.InputMessageContent*

thumb_url

Optional. URL of the thumbnail (jpeg only) for the file.

Type *str*

thumb_width

Optional. Thumbnail width.

Type *int*

thumb_height

Optional. Thumbnail height.

Type *int*

Parameters

- **id** (*str*) – Unique identifier for this result, 1-64 bytes.
- **title** (*str*) – Title for the result.
- **caption** (*str*, optional) – Caption, 0-1024 characters
- **parse_mode** (*str*, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.
- **document_url** (*str*) – A valid URL for the file.
- **mime_type** (*str*) – Mime type of the content of the file, either “application/pdf” or “application/zip”.
- **description** (*str*, optional) – Short description of the result.
- **reply_markup** (*telegram.InlineKeyboardMarkup*) – Optional. Inline keyboard attached to the message.
- **input_message_content** (*telegram.InputMessageContent*) – Optional. Content of the message to be sent instead of the file.
- **thumb_url** (*str*, optional) – URL of the thumbnail (jpeg only) for the file.
- **thumb_width** (*int*, optional) – Thumbnail width.
- **thumb_height** (*int*, optional) – Thumbnail height.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

1.45.15 telegram.InlineQueryResultGame

```
class telegram.InlineQueryResultGame (id, game_short_name, reply_markup=None,  
                                       **kwargs)
```

Bases: *telegram.inline.inlinequeryresult.InlineQueryResult*

Represents a Game.

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`

Parameters

- **id** (`str`) – Unique identifier for this result, 1-64 bytes.
- **game_short_name** (`str`) – Short name of the game.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.45.16 telegram.InlineQueryResultGif

```
class telegram.InlineQueryResultGif(id, gif_url, thumb_url, gif_width=None,
                                     gif_height=None, title=None, caption=None,
                                     reply_markup=None, input_message_content=None,
                                     gif_duration=None, parse_mode=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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.

Type `int`

thumb_url

URL of the static thumbnail for the result (jpeg or gif).

Type `str`

title

Optional. Title for the result.

Type `str`

caption

Optional. Caption, 0-1024 characters

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

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 gif.

Type `telegram.InputMessageContent`

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
- **thumb_url** (`str`) – URL of the static thumbnail for the result (jpeg or gif).
- **title** (`str`, optional) – Title for the result.caption (`str`, optional):
- **caption** (`str`, optional) – Caption, 0-1024 characters
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the gif.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.45.17 telegram.InlineQueryResultLocation

```
class telegram.InlineQueryResultLocation(id, latitude, longitude, title,
                                          live_period=None, reply_markup=None,
                                          input_message_content=None,
                                          thumb_url=None, thumb_width=None,
                                          thumb_height=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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`

live_period
Optional. Period in seconds for which the location can be updated, should be between 60 and 86400.
Type `int`

reply_markup
Optional. Inline keyboard attached to the message.
Type `telegram.InlineKeyboardMarkup`

input_message_content
Optional. Content of the message to be sent instead of the location.
Type `telegram.InputMessageContent`

thumb_url
Optional. Url of the thumbnail for the result.
Type `str`

thumb_width
Optional. Thumbnail width.
Type `int`

thumb_height
Optional. Thumbnail height.
Type `int`

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.
- **live_period** (`int`, optional) – Period in seconds for which the location can be updated, should be between 60 and 86400.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the location.
- **thumb_url** (`str`, optional) – Url of the thumbnail for the result.
- **thumb_width** (`int`, optional) – Thumbnail width.

- **thumb_height** (int, optional) – Thumbnail height.
- ****kwargs** (dict) – Arbitrary keyword arguments.

1.45.18 telegram.InlineQueryResultMpeg4Gif

```
class telegram.InlineQueryResultMpeg4Gif (id,          mpeg4_url,          thumb_url,
                                          mpeg4_width=None, mpeg4_height=None,
                                          title=None,          caption=None,
                                          reply_markup=None,      input_message_content=None,
                                          mpeg4_duration=None, parse_mode=None,
                                          **kwargs)
```

Bases: telegram.inline.inlinequeryresult.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.

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.

Type int

thumb_url

URL of the static thumbnail (jpeg or gif) for the result.

Type str

title

Optional. Title for the result.

Type str

caption

Optional. Caption, 0-1024 characters

Type str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in [*telegram.ParseMode*](#) for the available modes.

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 MPEG-4 file.

Type `telegram.InputMessageContent`

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.
- **thumb_url** (`str`) – URL of the static thumbnail (jpeg or gif) for the result.
- **title** (`str`, optional) – Title for the result.
- **caption** (`str`, optional) – Caption, 0-1024 characters
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the MPEG-4 file.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.45.19 telegram.InlineQueryResultPhoto

```
class telegram.InlineQueryResultPhoto(id, photo_url, thumb_url, photo_width=None,
                                       photo_height=None, title=None, description=None, caption=None, reply_markup=None,
                                       input_message_content=None, parse_mode=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

type

'photo'.

Type `str`

id

Unique identifier for this result, 1-64 bytes.

Type `str`

photo_url

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

Type `str`

thumb_url

URL of the thumbnail for the photo.

Type `str`

photo_width

Optional. Width of the photo.

Type `int`

photo_height

Optional. Height of the photo.

Type `int`

title

Optional. Title for the result.

Type `str`

description

Optional. Short description of the result.

Type `str`

caption

Optional. Caption, 0-1024 characters

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in [`telegram.ParseMode`](#) for the available modes.

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 photo.

Type [`telegram.InputMessageContent`](#)

Parameters

- **id** (`str`) – Unique identifier for this result, 1-64 bytes.
- **photo_url** (`str`) – A valid URL of the photo. Photo must be in jpeg format. Photo size must not exceed 5MB.
- **thumb_url** (`str`) – URL of the thumbnail for the photo.
- **photo_width** (`int`, optional) – Width of the photo.
- **photo_height** (`int`, optional) – Height of the photo.
- **title** (`str`, optional) – Title for the result.
- **description** (`str`, optional) – Short description of the result.
- **caption** (`str`, optional) – Caption, 0-1024 characters
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in [`telegram.ParseMode`](#) for the available modes.
- **reply_markup** ([`telegram.InlineKeyboardMarkup`](#), optional) – Inline keyboard attached to the message.

- **input_message_content** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the photo.
- ****kwargs** (dict) – Arbitrary keyword arguments.

1.45.20 telegram.InlineQueryResultVenue

```
class telegram.InlineQueryResultVenue (id, latitude, longitude, title,  
                                         address, foursquare_id=None,  
                                         foursquare_type=None, reply_markup=None,  
                                         input_message_content=None, thumb_url=None,  
                                         thumb_width=None, thumb_height=None,  
                                         **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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`

reply_markup

Optional. Inline keyboard attached to the message.

Type *telegram.InlineKeyboardMarkup*

input_message_content

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

Type *telegram.InputMessageContent*

thumb_url

Optional. Url of the thumbnail for the result.

Type `str`

thumb_width

Optional. Thumbnail width.

Type `int`

thumb_height

Optional. Thumbnail height.

Type `int`

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”).)
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the location.
- **thumb_url** (`str`, optional) – Url of the thumbnail for the result.
- **thumb_width** (`int`, optional) – Thumbnail width.
- **thumb_height** (`int`, optional) – Thumbnail height.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.45.21 telegram.InlineQueryResultVideo

```
class telegram.InlineQueryResultVideo(id, video_url, mime_type, thumb_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,  
                                       **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

type

‘video’.

Type `str`

id

Unique identifier for this result, 1-64 bytes.

Type `str`

video_url

A valid URL for the embedded video player or video file.

Type `str`

mime_type

Mime type of the content of video url, “text/html” or “video/mp4”.

Type `str`

thumb_url

URL of the thumbnail (jpeg only) for the video.

Type `str`

title

Title for the result.

Type `str`

caption

Optional. Caption, 0-1024 characters

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in [*telegram.ParseMode*](#) for the available modes.

Type `str`

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.

Type [*telegram.InputMessageContent*](#)

Parameters

- **id** (`str`) – Unique identifier for this result, 1-64 bytes.
- **video_url** (`str`) – A valid URL for the embedded video player or video file.
- **mime_type** (`str`) – Mime type of the content of video url, “text/html” or “video/mp4”.

- **thumb_url** (*str*) – URL of the thumbnail (jpeg only) for the video.
- **title** (*str*) – Title for the result.
- **caption** (*str*, optional) – Caption, 0-1024 characters.
- **parse_mode** (*str*, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in [telegram.ParseMode](#) for the available modes.
- **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.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

1.45.22 telegram.InlineQueryResultVoice

```
class telegram.InlineQueryResultVoice(id, voice_url, title, voice_duration=None,
                                       caption=None, reply_markup=None, input_message_content=None, parse_mode=None,
                                       **kwargs)
```

Bases: [telegram.inline.inlinequeryresult.InlineQueryResult](#)

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

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
Voice message title.

Type *str*

caption
Optional. Caption, 0-1024 characters.

Type *str*

parse_mode
Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption.. See the constants in [telegram.ParseMode](#) for the available modes.

Type *str*

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.

Type `telegram.InputMessageContent`

Parameters

- **id** (`str`) – Unique identifier for this result, 1-64 bytes.
- **voice_url** (`str`) – A valid URL for the voice recording.
- **title** (`str`) – Voice message title.
- **caption** (`str`, optional) – Caption, 0-1024 characters.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption.. See the constants in `telegram.ParseMode` for the available modes.
- **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.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.45.23 telegram.InputMessageContent

class `telegram.InputMessageContent`

Bases: `telegram.base.TelegramObject`

Base class for Telegram InputMessageContent Objects.

See: `telegram.InputContactMessageContent`, `telegram.InputLocationMessageContent`, `telegram.InputTextMessageContent` and `telegram.InputVenueMessageContent` for more details.

1.45.24 telegram.InputTextMessageContent

class `telegram.InputTextMessageContent` (`message_text`, `parse_mode=None`, `disable_web_page_preview=None`, `**kwargs`)

Bases: `telegram.inline.inputmessagecontent.InputMessageContent`

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

message_text

Text of the message to be sent, 1-4096 characters.

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.

Type `str`

disable_web_page_preview

Optional. Disables link previews for links in the sent message.

Type `bool`

Parameters

- **message_text** (`str`) – Text of the message to be sent, 1-4096 characters. Also found as `telegram.constants.MAX_MESSAGE_LENGTH`.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **disable_web_page_preview** (`bool`, optional) – Disables link previews for links in the sent message.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.45.25 telegram.InputLocationMessageContent

class `telegram.InputLocationMessageContent` (*latitude, longitude, live_period=None, **kwargs*)

Bases: `telegram.inline.inputmessagecontent.InputMessageContent`

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

latitude

Latitude of the location in degrees.

Type `float`

longitude

Longitude of the location in degrees.

Type `float`

Parameters

- **latitude** (`float`) – Latitude of the location in degrees.
- **longitude** (`float`) – Longitude of the location in degrees.
- **live_period** (`int`, optional) – Period in seconds for which the location can be updated, should be between 60 and 86400.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.45.26 telegram.InputVenueMessageContent

class `telegram.InputVenueMessageContent` (*latitude, longitude, title, address, foursquare_id=None, foursquare_type=None, **kwargs*)

Bases: `telegram.inline.inputmessagecontent.InputMessageContent`

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

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`

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

1.45.27 telegram.InputContactMessageContent

class `telegram.InputContactMessageContent` (*phone_number*, *first_name*,
last_name=None, *vcard=None*, ***kwargs*)

Bases: `telegram.inline.inputmessagecontent.InputMessageContent`

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

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`

Parameters

- **phone_number** (*str*) – Contact’s phone number.
- **first_name** (*str*) – Contact’s first name.
- **last_name** (*str*, optional) – Contact’s last name.
- **vcard** (*str*, optional) – Additional data about the contact in the form of a vCard, 0-2048 bytes.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

1.45.28 telegram.ChosenInlineResult

class telegram.ChosenInlineResult(*result_id*, *from_user*, *query*, *location=None*, *inline_message_id=None*, ***kwargs*)

Bases: telegram.base.TelegramObject

Represents a result of an inline query that was chosen by the user and sent to their chat partner.

Note: In Python *from* is a reserved word, use *from_user* instead.

result_id

The unique identifier for the result that was chosen.

Type *str*

from_user

The user that chose the result.

Type *telegram.User*

location

Optional. Sender location.

Type *telegram.Location*

inline_message_id

Optional. Identifier of the sent inline message.

Type *str*

query

The query that was used to obtain the result.

Type *str*

Parameters

- **result_id** (*str*) – The unique identifier for the result that was chosen.
- **from_user** (*telegram.User*) – The user that chose the result.
- **location** (*telegram.Location*, optional) – Sender location, only for bots that require user location.
- **inline_message_id** (*str*, optional) – Identifier of the sent inline message. Available only if there is an inline keyboard attached to the message. Will be also received in callback queries and can be used to edit the message.
- **query** (*str*) – The query that was used to obtain the result.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

1.46 Payments

1.46.1 telegram.LabeledPrice

class telegram.**LabeledPrice** (*label, amount, **kwargs*)

Bases: telegram.base.TelegramObject

This object represents a portion of the price for goods or services.

label

Portion label.

Type str

amount

Price of the product in the smallest units of the currency.

Type int

Parameters

- **label** (str) – Portion label
- **amount** (int) – Price of the product in the smallest units of the currency (integer, not float/double). For example, for a price of US\$ 1.45 pass amount = 145. See the exp parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- ****kwargs** (dict) – Arbitrary keyword arguments.

1.46.2 telegram.Invoice

class telegram.**Invoice** (*title, description, start_parameter, currency, total_amount, **kwargs*)

Bases: telegram.base.TelegramObject

This object contains basic information about an invoice.

title

Product name.

Type str

description

Product description.

Type str

start_parameter

Unique bot deep-linking parameter.

Type str

currency

Three-letter ISO 4217 currency code.

Type str

total_amount

Total price in the smallest units of the currency.

Type int

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

1.46.3 telegram.ShippingAddress

class telegram.**ShippingAddress** (*country_code*, *state*, *city*, *street_line1*, *street_line2*,
post_code, ****kwargs**)

Bases: telegram.base.TelegramObject

This object represents a Telegram ShippingAddress.

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*

Parameters

- **country_code** (*str*) – ISO 3166-1 alpha-2 country code.
- **state** (*str*) – State, if applicable.
- **city** (*str*) – City.
- **street_line1** (*str*) – First line for the address.
- **street_line2** (*str*) – Second line for the address.
- **post_code** (*str*) – Address post code.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

1.46.4 telegram.OrderInfo

```
class telegram.OrderInfo (name=None, phone_number=None, email=None, shipping_address=None, **kwargs)
```

Bases: telegram.base.TelegramObject

This object represents information about an order.

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

Parameters

- **name** (str, optional) – User name.
- **phone_number** (str, optional) – User's phone number.
- **email** (str, optional) – User email.
- **shipping_address** (telegram.ShippingAddress, optional) – User shipping address.
- ****kwargs** (dict) – Arbitrary keyword arguments.

1.46.5 telegram.ShippingOption

```
class telegram.ShippingOption (id, title, prices, **kwargs)
```

Bases: telegram.base.TelegramObject

This object represents one shipping option.

id

Shipping option identifier.

Type str

title

Option title.

Type str

prices

List of price portions.

Type List[telegram.LabeledPrice]

Parameters

- **id** (str) – Shipping option identifier.
- **title** (str) – Option title.

- **prices** (List[*telegram.LabeledPrice*]) – List of price portions.
- ****kwargs** (dict) – Arbitrary keyword arguments.

1.46.6 telegram.SuccessfulPayment

```
class telegram.SuccessfulPayment (currency, total_amount, invoice_payload, telegram_payment_charge_id, provider_payment_charge_id, shipping_option_id=None, order_info=None, **kwargs)
```

Bases: telegram.base.TelegramObject

This object contains basic information about a successful payment.

currency

Three-letter ISO 4217 currency code.

Type `str`

total_amount

Total price in the smallest units of the currency.

Type `int`

invoice_payload

Bot specified invoice payload.

Type `str`

shipping_option_id

Optional. Identifier of the shipping option chosen by the user.

Type `str`

order_info

Optional. Order info provided by the user.

Type *telegram.OrderInfo*

telegram_payment_charge_id

Telegram payment identifier.

Type `str`

provider_payment_charge_id

Provider payment identifier.

Type `str`

Parameters

- **currency** (`str`) – Three-letter ISO 4217 currency code.
- **total_amount** (`int`) – Total price in the smallest units of the currency (integer, not float/double). For example, for a price of US\$ 1.45 pass amount = 145. See the exp parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- **invoice_payload** (`str`) – Bot specified invoice payload.
- **shipping_option_id** (`str`, optional) – Identifier of the shipping option chosen by the user.
- **order_info** (*telegram.OrderInfo*, optional) – Order info provided by the user
- **telegram_payment_charge_id** (`str`) – Telegram payment identifier.
- **provider_payment_charge_id** (`str`) – Provider payment identifier.
- ****kwargs** (dict) – Arbitrary keyword arguments.

1.46.7 telegram.ShippingQuery

class telegram.**ShippingQuery** (*id, from_user, invoice_payload, shipping_address, bot=None, **kwargs*)

Bases: telegram.base.TelegramObject

This object contains information about an incoming shipping query.

Note:

- In Python *from* is a reserved word, use *from_user* instead.
-

id

Unique query identifier.

Type `str`

from_user

User who sent the query.

Type `telegram.User`

invoice_payload

Bot specified invoice payload.

Type `str`

shipping_address

User specified shipping address.

Type `telegram.ShippingAddress`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

Parameters

- **id** (`str`) – Unique query identifier.
- **from_user** (`telegram.User`) – User who sent the query.
- **invoice_payload** (`str`) – Bot specified invoice payload.
- **shipping_address** (`telegram.ShippingAddress`) – User specified shipping address.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

answer (**args, **kwargs*)

Shortcut for:

`bot.answer_shipping_query(update.shipping_query.id, *args, **kwargs)`

Parameters

- **ok** (`bool`) – Specify True if delivery to the specified address is possible and False if there are any problems (for example, if delivery to the specified address is not possible).
- **shipping_options** (`List[telegram.ShippingOption]`, optional) – Required if ok is True. A JSON-serialized array of available shipping options.

- **error_message** (`str`, optional) – Required if `ok` is `False`. Error message in human readable form that explains why it is impossible to complete the order (e.g. “Sorry, delivery to your desired address is unavailable”). Telegram will display this message to the user.

1.46.8 telegram.PreCheckoutQuery

```
class telegram.PreCheckoutQuery(id, from_user, currency, total_amount, invoice_payload,  
                                shipping_option_id=None, order_info=None, bot=None,  
                                **kwargs)
```

Bases: `telegram.base.TelegramObject`

This object contains information about an incoming pre-checkout query.

Note:

- In Python *from* is a reserved word, use *from_user* instead.
-

id

Unique query identifier.

Type `str`

from_user

User who sent the query.

Type `telegram.User`

currency

Three-letter ISO 4217 currency code.

Type `str`

total_amount

Total price in the smallest units of the currency.

Type `int`

invoice_payload

Bot specified invoice payload.

Type `str`

shipping_option_id

Optional. Identifier of the shipping option chosen by the user.

Type `str`

order_info

Optional. Order info provided by the user.

Type `telegram.OrderInfo`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

Parameters

- **id** (`str`) – Unique query identifier.
- **from_user** (`telegram.User`) – User who sent the query.
- **currency** (`str`) – Three-letter ISO 4217 currency code

- **total_amount** (`int`) – Total price in the smallest units of the currency (integer, not float/double). For example, for a price of US\$ 1.45 pass amount = 145. See the exp parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- **invoice_payload** (`str`) – Bot specified invoice payload.
- **shipping_option_id** (`str`, optional) – Identifier of the shipping option chosen by the user.
- **order_info** (`telegram.OrderInfo`, optional) – Order info provided by the user.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

answer (**args*, ***kwargs*)

Shortcut for:

```
bot.answer_pre_checkout_query(update.pre_checkout_query.id, *args,   
↪ **kwargs)
```

Parameters

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

1.47 Games

1.47.1 telegram.Game

class `telegram.Game` (*title, description, photo, text=None, text_entities=None, animation=None, **kwargs*)

Bases: `telegram.base.TelegramObject`

This object represents a game. Use BotFather to create and edit games, their short names will act as unique identifiers.

title

Title of the game.

Type `str`

description

Description of the game.

Type `str`

photo

Photo that will be displayed in the game message in chats.

Type `List[telegram.PhotoSize]`

text

Optional. Brief description of the game or high scores included in the game message. Can be automatically edited to include current high scores for the game when the bot calls `set_game_score`, or manually edited using `edit_message_text`.

Type `str`

text_entities

Optional. Special entities that appear in text, such as usernames, URLs, bot commands, etc.

Type `List[telegram.MessageEntity]`

animation

Optional. Animation that will be displayed in the game message in chats. Upload via BotFather.

Type `telegram.Animation`

Parameters

- **title** (`str`) – Title of the game.
- **description** (`str`) – Description of the game.
- **photo** (`List[telegram.PhotoSize]`) – Photo that will be displayed in the game message in chats.
- **text** (`str`, optional) – Brief description of the game or high scores included in the game message. Can be automatically edited to include current high scores for the game when the bot calls `set_game_score`, or manually edited using `edit_message_text`. 0-4096 characters. Also found as `telegram.constants.MAX_MESSAGE_LENGTH`.
- **text_entities** (`List[telegram.MessageEntity]`, optional) – Special entities that appear in text, such as usernames, URLs, bot commands, etc.
- **animation** (`telegram.Animation`, optional) – Animation that will be displayed in the game message in chats. Upload via BotFather.

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 `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`

1.47.2 telegram.CallbackGame

class `telegram.CallbackGame`

Bases: `telegram.base.TelegramObject`

A placeholder, currently holds no information. Use BotFather to set up your game.

1.47.3 telegram.GameHighScore

class `telegram.GameHighScore` (*position, user, score*)

Bases: `telegram.base.TelegramObject`

This object represents one row of the high scores table for a game.

position

Position in high score table for the game.

Type `int`

user

User.

Type *telegram.User*

score

Score.

Type `int`

Parameters

- **position** (`int`) – Position in high score table for the game.
- **user** (*telegram.User*) – User.
- **score** (`int`) – Score.

1.48 Passport

1.48.1 telegram.PassportElementError

class `telegram.PassportElementError` (*source, type, message, **kwargs*)

Bases: `telegram.base.TelegramObject`

Baseclass for the PassportElementError* classes.

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`

Parameters

- **source** (`str`) – Error source.
- **type** (`str`) – The section of the user’s Telegram Passport which has the error.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.48.2 telegram.PassportElementErrorFile

class `telegram.PassportElementErrorFile` (*type, file_hash, message, **kwargs*)

Bases: `telegram.passport.passportelementerrors.PassportElementError`

Represents an issue with a document scan. The error is considered resolved when the file with the document scan changes.

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`

Parameters

- **type** (`str`) – The section of the user’s Telegram Passport which has the issue, one of “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.
- **file_hash** (`str`) – Base64-encoded file hash.
- **message** (`str`) – Error message.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.48.3 telegram.PassportElementErrorReverseSide

class `telegram.PassportElementErrorReverseSide` (*type, file_hash, message, **kwargs*)

Bases: `telegram.passport.passportelementerrors.PassportElementError`

Represents an issue with the front side of a document. The error is considered resolved when the file with the reverse side of the document changes.

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 reverse side of the document.

Type `str`

message

Error message.

Type `str`

Parameters

- **type** (`str`) – The section of the user’s Telegram Passport which has the issue, one of “driver_license”, “identity_card”.
- **file_hash** (`str`) – Base64-encoded hash of the file with the reverse side of the document.
- **message** (`str`) – Error message.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.48.4 telegram.PassportElementErrorFrontSide

class `telegram.PassportElementErrorFrontSide` (*type, file_hash, message, **kwargs*)

Bases: `telegram.passport.passportelementerrors.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.

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`

Parameters

- **type** (`str`) – The section of the user’s Telegram Passport which has the issue, one of “passport”, “driver_license”, “identity_card”, “internal_passport”.
- **file_hash** (`str`) – Base64-encoded hash of the file with the front side of the document.
- **message** (`str`) – Error message.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.48.5 telegram.PassportElementErrorFiles

class `telegram.PassportElementErrorFiles` (*type, file_hashes, message, **kwargs*)

Bases: `telegram.passport.passportelementerrors.PassportElementError`

Represents an issue with a list of scans. The error is considered resolved when the file with the document scan changes.

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`

Parameters

- **type** (`str`) – The section of the user’s Telegram Passport which has the issue, one of “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.
- **file_hashes** (`List[str]`) – List of base64-encoded file hashes.
- **message** (`str`) – Error message.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.48.6 telegram.PassportElementErrorDataField

```
class telegram.PassportElementErrorDataField(type, field_name, data_hash, message,  
                                              **kwargs)
```

Bases: `telegram.passport.passportelementerrors.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.

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`

Parameters

- **type** (`str`) – The section of the user’s Telegram Passport which has the error, one of “personal_details”, “passport”, “driver_license”, “identity_card”, “internal_passport”, “address”.
- **field_name** (`str`) – Name of the data field which has the error.
- **data_hash** (`str`) – Base64-encoded data hash.
- **message** (`str`) – Error message.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.48.7 telegram.Credentials

class telegram.**Credentials** (*secure_data, nonce, bot=None, **kwargs*)
Bases: telegram.base.TelegramObject

secure_data
Credentials for encrypted data
Type *telegram.SecureData*

nonce
Bot-specified nonce
Type *str*

1.48.8 telegram.DataCredentials

class telegram.**DataCredentials** (*data_hash, secret, **kwargs*)
Bases: telegram.passport.credentials._CredentialsBase

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*

1.48.9 telegram.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, bot=None, **kwargs*)
Bases: telegram.base.TelegramObject

This object represents the credentials that were used to decrypt the encrypted data. All fields are optional and depend on fields that were requested.

personal_details
Credentials for encrypted personal details.
Type *telegram.SecureValue*, optional

passport
Credentials for encrypted passport.
Type *telegram.SecureValue*, optional

internal_passport
Credentials for encrypted internal passport.
Type *telegram.SecureValue*, optional

driver_license
Credentials for encrypted driver license.

Type telegram.SecureValue, optional

identity_card
Credentials for encrypted ID card

Type telegram.SecureValue, optional

address
Credentials for encrypted residential address.

Type telegram.SecureValue, optional

utility_bill
Credentials for encrypted utility bill.

Type telegram.SecureValue, optional

bank_statement
Credentials for encrypted bank statement.

Type telegram.SecureValue, optional

rental_agreement
Credentials for encrypted rental agreement.

Type telegram.SecureValue, optional

passport_registration
Credentials for encrypted registration from internal passport.

Type telegram.SecureValue, optional

temporary_registration
Credentials for encrypted temporary registration.

Type telegram.SecureValue, optional

1.48.10 telegram.FileCredentials

class telegram.**FileCredentials** (*file_hash*, *secret*, ***kwargs*)

Bases: telegram.passport.credentials._CredentialsBase

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*

1.48.11 telegram.IdDocumentData

class telegram.**IdDocumentData** (*document_no*, *expiry_date*, *bot=None*, ***kwargs*)

Bases: telegram.base.TelegramObject

This object represents the data of an identity document.

document_no
Document number.
Type `str`

expiry_date
Optional. Date of expiry, in DD.MM.YYYY format.
Type `str`

1.48.12 telegram.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, bot=None, **kwargs*)

Bases: `telegram.base.TelegramObject`

This object represents personal details.

first_name
First Name.
Type `str`

middle_name
Optional. First Name.
Type `str`

last_name
Last Name.
Type `str`

birth_date
Date of birth in DD.MM.YYYY format.
Type `str`

gender
Gender, male or female.
Type `str`

country_code
Citizenship (ISO 3166-1 alpha-2 country code).
Type `str`

residence_country_code
Country of residence (ISO 3166-1 alpha-2 country code).
Type `str`

first_name
First Name in the language of the user's country of residence.
Type `str`

middle_name
Optional. Middle Name in the language of the user's country of residence.
Type `str`

last_name
Last Name in the language of the user's country of residence.
Type `str`

1.48.13 telegram.ResidentialAddress

```
class telegram.ResidentialAddress(street_line1, street_line2, city, state, country_code,
                                   post_code, bot=None, **kwargs)
```

Bases: telegram.base.TelegramObject

This object represents a residential address.

street_line1

First line for the address.

Type str

street_line2

Optional. Second line for the address.

Type str

city

City.

Type str

state

Optional. State.

Type str

country_code

ISO 3166-1 alpha-2 country code.

Type str

post_code

Address post code.

Type str

1.48.14 telegram.PassportData

```
class telegram.PassportData(data, credentials, bot=None, **kwargs)
```

Bases: telegram.base.TelegramObject

Contains information about Telegram Passport data shared with the bot by the user.

data

Array with encrypted information about documents and other Telegram Passport elements that was shared with the bot.

Type List[*telegram.EncryptedPassportElement*]

credentials

Encrypted credentials.

Type *telegram.EncryptedCredentials*

bot

The Bot to use for instance methods.

Type *telegram.Bot*, optional

Parameters

- **data** (List[*telegram.EncryptedPassportElement*]) – Array with encrypted information about documents and other Telegram Passport elements that was shared with the bot.
- **credentials** (str) – Encrypted credentials.

- **bot** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- ****kwargs** (dict) – Arbitrary keyword arguments.

Note: To be able to decrypt this object, you must pass your `private_key` to either `telegram.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.payload`.

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.TelegramDecryptionError` – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type *telegram.Credentials*

decrypted_data

Lazily decrypt and return information about documents and other Telegram Passport elements which were shared with the bot.

Raises `telegram.TelegramDecryptionError` – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type List[*telegram.EncryptedPassportElement*]

1.48.15 telegram.PassportFile

class `telegram.PassportFile` (*file_id, file_date, file_size=None, bot=None, credentials=None, **kwargs*)

Bases: `telegram.base.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.

file_id

Unique identifier for this file.

Type `str`

file_size

File size.

Type `int`

file_date

Unix time when the file was uploaded.

Type `int`

bot

Optional. The Bot to use for instance methods.

Type *telegram.Bot*

Parameters

- **file_id** (`str`) – Unique identifier for this file.
- **file_size** (`int`) – File size.
- **file_date** (`int`) – Unix time when the file was uploaded.

- **bot** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- ****kwargs** (dict) – Arbitrary keyword arguments.

get_file (*timeout=None*, ***kwargs*)

Wrapper over *telegram.Bot.get_file*. Will automatically assign the correct credentials to the returned *telegram.File* if originating from *telegram.PassportData.decrypted_data*.

Parameters

- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns *telegram.File*

Raises *telegram.TelegramError*

1.48.16 telegram.EncryptedPassportElement

```
class telegram.EncryptedPassportElement (type, data=None, phone_number=None,  
                                          email=None, files=None, front_side=None,  
                                          reverse_side=None, selfie=None, translation=None, hash=None, bot=None, credentials=None, **kwargs)
```

Bases: *telegram.base.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.

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*

data

Optional. Decrypted or encrypted data, available for “personal_details”, “passport”, “driver_license”, “identity_card”, “identity_passport” and “address” types.

Type *telegram.PersonalDetails* or *telegram.IdDocument* or *telegram.ResidentialAddress* or *str*

phone_number

Optional. User’s verified phone number, available only for “phone_number” type.

Type *str*

email

Optional. User’s verified email address, available only for “email” type.

Type *str*

files

Optional. Array of encrypted/decrypted files with documents provided by the user, available for “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.

Type List[*telegram.PassportFile*]

front_side

Optional. Encrypted/decrypted file with the front side of the document, provided by the user. Available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

Type `telegram.PassportFile`

reverse_side

Optional. Encrypted/decrypted file with the reverse side of the document, provided by the user. Available for “driver_license” and “identity_card”.

Type `telegram.PassportFile`

selfie

Optional. Encrypted/decrypted file with the selfie of the user holding a document, provided by the user; available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

Type `telegram.PassportFile`

translation

Optional. Array of encrypted/decrypted files with translated versions of documents provided by the user. Available if requested for “passport”, “driver_license”, “identity_card”, “internal_passport”, “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.

Type `List[telegram.PassportFile]`

hash

Base64-encoded element hash for using in `telegram.PassportElementErrorUnspecified`.

Type `str`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

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”.
- **data** (`telegram.PersonalDetails` or `telegram.IdDocument` or `telegram.ResidentialAddress` or `str`, optional) – Decrypted or encrypted data, available for “personal_details”, “passport”, “driver_license”, “identity_card”, “identity_passport” and “address” types.
- **phone_number** (`str`, optional) – User’s verified phone number, available only for “phone_number” type.
- **email** (`str`, optional) – User’s verified email address, available only for “email” type.
- **files** (`List[telegram.PassportFile]`, optional) – Array of encrypted/decrypted files with documents provided by the user, available for “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.
- **front_side** (`telegram.PassportFile`, optional) – Encrypted/decrypted file with the front side of the document, provided by the user. Available for “passport”, “driver_license”, “identity_card” and “internal_passport”.
- **reverse_side** (`telegram.PassportFile`, optional) – Encrypted/decrypted file with the reverse side of the document, provided by the user. Available for “driver_license” and “identity_card”.

- **selfie** (*telegram.PassportFile*, optional) – Encrypted/decrypted file with the selfie of the user holding a document, provided by the user; available for “passport”, “driver_license”, “identity_card” and “internal_passport”.
- **translation** (List[*telegram.PassportFile*], optional) – Array of encrypted/decrypted files with translated versions of documents provided by the user. Available if requested for “passport”, “driver_license”, “identity_card”, “internal_passport”, “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.
- **hash** (str) – Base64-encoded element hash for using in telegram.PassportElementErrorUnspecified.
- **bot** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- ****kwargs** (dict) – Arbitrary keyword arguments.

Note: This object is decrypted only when originating from *telegram.PassportData.decrypted_data*.

1.48.17 telegram.EncryptedCredentials

class telegram.**EncryptedCredentials** (*data, hash, secret, bot=None, **kwargs*)

Bases: telegram.base.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.

data

Decrypted data with unique user’s nonce, data hashes and secrets used for EncryptedPassportElement decryption and authentication or base64 encrypted data.

Type *telegram.Credentials* or str

hash

Base64-encoded data hash for data authentication.

Type str

secret

Decrypted or encrypted secret used for decryption.

Type str

Parameters

- **data** (*telegram.Credentials* or str) – Decrypted data with unique user’s nonce, data hashes and secrets used for EncryptedPassportElement decryption and authentication or base64 encrypted data.
- **hash** (str) – Base64-encoded data hash for data authentication.
- **secret** (str) – Decrypted or encrypted secret used for decryption.
- ****kwargs** (dict) – Arbitrary keyword arguments.

Note: This object is decrypted only when originating from *telegram.PassportData.decrypted_credentials*.

decrypted_data

Lazily decrypt and return credentials data. This object also contains the user specified nonce as *decrypted_data.nonce*.

Raises telegram.TelegramDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type *telegram.Credentials*

decrypted_secret

Lazily decrypt and return secret.

Raises telegram.TelegramDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type `str`

1.49 Module contents

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

```
class telegram.Audio(file_id, duration, performer=None, title=None, mime_type=None,
                    file_size=None, thumb=None, bot=None, **kwargs)
Bases: telegram.base.TelegramObject
```

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

file_id

Unique identifier for this file.

Type `str`

duration

Duration of the audio in seconds.

Type `int`

performer

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

Type `str`

title

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

Type `str`

mime_type

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

Type `str`

file_size

Optional. File size.

Type `int`

thumb

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

Type *telegram.PhotoSize*

bot

Optional. The Bot to use for instance methods.

Type *telegram.Bot*

Parameters

- **file_id** (str) – Unique identifier for this 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.
- **mime_type** (str, optional) – MIME type of the file as defined by sender.
- **file_size** (int, optional) – File size.
- **thumb** (*telegram.PhotoSize*, optional) – Thumbnail of the album cover to which the music file belongs
- **bot** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- ****kwargs** (dict) – Arbitrary keyword arguments.

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

get_file (*timeout=None*, ***kwargs*)

Convenience wrapper over *telegram.Bot.get_file*

Parameters

- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns *telegram.File*

Raises *telegram.TelegramError*

class *telegram.Bot* (*token*, *base_url=None*, *base_file_url=None*, *request=None*, *private_key=None*, *private_key_password=None*)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram Bot.

Parameters

- **token** (str) – Bot's unique authentication.
- **base_url** (str, optional) – Telegram Bot API service URL.
- **base_file_url** (str, optional) – Telegram Bot API file URL.
- **request** (*telegram.utils.request.Request*, optional) – Pre initialized *telegram.utils.request.Request*.
- **private_key** (bytes, optional) – Private key for decryption of telegram passport data.
- **private_key_password** (bytes, optional) – Password for above private key.

addStickerToSet (*user_id*, *name*, *png_sticker*, *emojis*, *mask_position=None*, *timeout=20*, ***kwargs*)

Alias for *add_sticker_to_set*

add_sticker_to_set (*user_id*, *name*, *png_sticker*, *emojis*, *mask_position=None*, *timeout=20*, ***kwargs*)

Use this method to add a new sticker to a set created by the bot.

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

Parameters

- **user_id** (int) – User identifier of created sticker set owner.
- **name** (str) – Sticker set name.
- **png_sticker** (str | *filelike object*) – Png image with the sticker, must be up to 512 kilobytes in size, dimensions must not exceed 512px, and either width or height must be exactly 512px. Pass a file_id as a String to send a file that already exists on the Telegram servers, pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data.
- **emojis** (str) – One or more emoji corresponding to the sticker.
- **mask_position** (*telegram.MaskPosition*, optional) – Position where the mask should be placed on faces.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, True is returned.

Return type bool

Raises telegram.TelegramError

answerCallbackQuery (callback_query_id, text=None, show_alert=False, url=None, cache_time=None, timeout=None, **kwargs)

Alias for `answer_callback_query`

answerInlineQuery (inline_query_id, results, cache_time=300, is_personal=None, next_offset=None, switch_pm_text=None, switch_pm_parameter=None, timeout=None, **kwargs)

Alias for `answer_inline_query`

answerPreCheckoutQuery (pre_checkout_query_id, ok, error_message=None, timeout=None, **kwargs)

Alias for `answer_pre_checkout_query`

answerShippingQuery (shipping_query_id, ok, shipping_options=None, error_message=None, timeout=None, **kwargs)

Alias for `answer_shipping_query`

answer_callback_query (callback_query_id, text=None, show_alert=False, url=None, cache_time=None, timeout=None, **kwargs)

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.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns bool On success, True is returned.

Raises telegram.TelegramError

answer_inline_query (*inline_query_id*, *results*, *cache_time=300*,
is_personal=None, *next_offset=None*, *switch_pm_text=None*,
switch_pm_parameter=None, *timeout=None*, ***kwargs*)

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

Parameters

- **inline_query_id** (str) – Unique identifier for the answered query.
- **results** (List[*telegram.InlineQueryResult*]) – A list of results for the inline query.
- **cache_time** (int, optional) – The maximum amount of time in seconds that the result of the inline query may be cached on the server. Defaults to 300.
- **is_personal** (bool, optional) – Pass True, if results may be cached on the server side only for the user that sent the query. By default, results may be returned to any user who sends the same query.
- **next_offset** (str, optional) – Pass the offset that a client should send in the next query with the same text to receive more results. Pass an empty string if there are no more results or if you don't support pagination. Offset length can't exceed 64 bytes.
- **switch_pm_text** (str, optional) – If passed, clients will display a button with specified text that switches the user to a private chat with the bot and sends the bot a start message with the parameter switch_pm_parameter.
- **switch_pm_parameter** (str, optional) – Deep-linking parameter for the /start message sent to the bot when user presses the switch button. 1-64 characters, only A-Z, a-z, 0-9, _ and - are allowed.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Example

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

Returns bool On success, True is returned.

Raises telegram.TelegramError

answer_pre_checkout_query (*pre_checkout_query_id*, *ok*, *error_message=None*, *timeout=None*, ***kwargs*)

Once the user has confirmed their payment and shipping details, the Bot API sends the final confirmation in the form of an Update with the field 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.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns On success, True is returned.

Return type `bool`

Raises `telegram.TelegramError`

answer_shipping_query (*shipping_query_id*, *ok*, *shipping_options=None*, *error_message=None*, *timeout=None*, ***kwargs*)

If you sent an invoice requesting a shipping address and the parameter `is_flexible` was specified, the Bot API will send an Update with a `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** (`List[telegram.ShippingOption]`) – Required if `ok` is True. A JSON-serialized array of available shipping options.
- **error_message** (`str`, optional) – Required if `ok` is False. Error message in human readable form that explains why it is impossible to complete the order (e.g. “Sorry, delivery to your desired address is unavailable”). Telegram will display this message to the user.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

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

Raises `telegram.TelegramError`

createNewStickerSet (*user_id*, *name*, *title*, *png_sticker*, *emojis*, *contains_masks=None*, *mask_position=None*, *timeout=20*, ***kwargs*)

Alias for `create_new_sticker_set`

create_new_sticker_set (*user_id*, *name*, *title*, *png_sticker*, *emojis*, *contains_masks=None*, *mask_position=None*, *timeout=20*, ***kwargs*)

Use this method to create new sticker set owned by a user.

The bot will be able to edit the created sticker set.

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

Parameters

- **user_id** (`int`) – User identifier of created sticker set owner.
- **name** (`str`) – Short name of sticker set, to be used in `t.me/addstickers/` URLs (e.g., animals). Can contain only english letters, digits and underscores. Must begin with a letter, can't contain consecutive underscores and must end in “_by_<bot username>”. <bot_username> is case insensitive. 1-64 characters.
- **title** (`str`) – Sticker set title, 1-64 characters.
- **png_sticker** (`str` | *filelike object*) – Png image with the sticker, must be up to 512 kilobytes in size, dimensions must not exceed 512px, and either width or height must be exactly 512px. Pass a `file_id` as a String to send a file that already exists on the Telegram servers, pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data.
- **emojis** (`str`) – One or more emoji corresponding to the sticker.
- **contains_masks** (`bool`, optional) – Pass True, if a set of mask stickers should be created.
- **mask_position** (`telegram.MaskPosition`, optional) – Position where the mask should be placed on faces.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns On success, True is returned.

Return type `bool`

Raises `telegram.TelegramError`

deleteChatPhoto (`chat_id`, `timeout=None`, ****kwargs**)

Alias for `delete_chat_photo`

deleteChatStickerSet (`chat_id`, `timeout=None`, ****kwargs**)

Alias for `delete_chat_sticker_set`

deleteMessage (`chat_id`, `message_id`, `timeout=None`, ****kwargs**)

Alias for `delete_message`

deleteStickerFromSet (`sticker`, `timeout=None`, ****kwargs**)

Alias for `delete_sticker_from_set`

deleteWebhook (`timeout=None`, ****kwargs**)

Alias for `delete_webhook`

delete_chat_photo (`chat_id`, `timeout=None`, ****kwargs**)

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`).

- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments

Note: In regular groups (non-supergroups), this method will only work if the ‘All Members Are Admins’ setting is off in the target group.

Returns Returns `True` on success.

Return type `bool`

Raises `telegram.TelegramError`

delete_chat_sticker_set (*chat_id*, *timeout=None*, ***kwargs*)

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`).
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns `True` on success.

Return type `bool`

delete_message (*chat_id*, *message_id*, *timeout=None*, ***kwargs*)

Use this method to delete a message. A message can only be deleted if it was sent less than 48 hours ago. Any such recently sent outgoing message may be deleted. Additionally, if the bot is an administrator in a group chat, it can delete any message. If the bot is an administrator in a supergroup, it can delete messages from any other user and service messages about people joining or leaving the group (other types of service messages may only be removed by the group creator). In channels, bots can only remove their own messages.

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.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as
- **read timeout** (*the*) – from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns On success, `True` is returned.

Return type `bool`

Raises `telegram.TelegramError`

delete_sticker_from_set (*sticker*, *timeout=None*, ***kwargs*)

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

Parameters

- **sticker** (*str*) – File identifier of the sticker.
- **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns On success, `True` is returned.

Return type `bool`

Raises `telegram.TelegramError`

delete_webhook (*timeout=None*, ***kwargs*)

Use this method to remove webhook integration if you decide to switch back to `getUpdates`. Requires no parameters.

Parameters

- **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

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

Raises `telegram.TelegramError`

editMessageCaption (*chat_id=None*, *message_id=None*, *inline_message_id=None*, *caption=None*, *reply_markup=None*, *timeout=None*, *parse_mode=None*, ***kwargs*)

Alias for `edit_message_caption`

editMessageLiveLocation (*chat_id=None*, *message_id=None*, *inline_message_id=None*, *latitude=None*, *longitude=None*, *location=None*, *reply_markup=None*, *timeout=None*, ***kwargs*)

Alias for `edit_message_live_location`

editMessageMedia (*chat_id=None*, *message_id=None*, *inline_message_id=None*, *media=None*, *reply_markup=None*, *timeout=None*, ***kwargs*)

Alias for `edit_message_media`

editMessageReplyMarkup (*chat_id=None*, *message_id=None*, *inline_message_id=None*, *reply_markup=None*, *timeout=None*, ***kwargs*)

Alias for `edit_message_reply_markup`

editMessageText (*text*, *chat_id=None*, *message_id=None*, *inline_message_id=None*, *parse_mode=None*, *disable_web_page_preview=None*, *reply_markup=None*, *timeout=None*, ***kwargs*)

Alias for `edit_message_text`

edit_message_caption (*chat_id=None*, *message_id=None*, *inline_message_id=None*, *caption=None*, *reply_markup=None*, *timeout=None*, *parse_mode=None*, ***kwargs*)

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

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) – 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.
- **caption** (*str*, optional) – New caption of the message.

- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

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 `telegram.TelegramError`

```
edit_message_live_location (chat_id=None, message_id=None, in-  
                             line_message_id=None, latitude=None, longitude=None,  
                             location=None, reply_markup=None, timeout=None,  
                             **kwargs)
```

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 `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) – 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.
- **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.
- **location** (`telegram.Location`, optional) – The location to send.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

Returns On success the edited message.

Return type `telegram.Message`

```
edit_message_media (chat_id=None, message_id=None, inline_message_id=None, me-  
                     dia=None, reply_markup=None, timeout=None, **kwargs)
```

Use this method to edit audio, document, photo, or video messages. If a message is a part of a message album, then it can be edited only to a photo or a video. Otherwise, message type can be changed arbitrarily. When inline message is edited, new file can't be uploaded. Use previously uploaded file

via its `file_id` or specify a URL. On success, if the edited message was sent by the bot, the edited Message is returned, otherwise True is returned.

Parameters

- **chat_id** (`int` | `str`, optional) – 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.
- **inline_message_id** (`str`, optional) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.
- **media** (`telegram.InputMedia`) – An object for a new media content of the message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

edit_message_reply_markup (`chat_id=None`, `message_id=None`, `inline_message_id=None`, `reply_markup=None`, `timeout=None`, ****kwargs**)

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

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) – 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.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

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

Return type `telegram.Message`

Raises `telegram.TelegramError`

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`, `timeout=None`, ****kwargs**)

Use this method to edit text and game messages sent by the bot or via the bot (for inline bots).

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) – 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.
- **text** (str) – New text of the message.
- **parse_mode** (str) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message. See the constants in `telegram.ParseMode` for the available modes.
- **disable_web_page_preview** (bool, optional) – Disables link previews for links in this message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

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 `telegram.TelegramError`

exportChatInviteLink (*chat_id*, *timeout=None*, ***kwargs*)

Alias for `export_chat_invite_link`

export_chat_invite_link (*chat_id*, *timeout=None*, ***kwargs*)

Use this method to export an invite link to 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).
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments

Returns Exported invite link on success.

Return type str

Raises `telegram.TelegramError`

first_name

Bot's first name.

Type str

forwardMessage (*chat_id*, *from_chat_id*, *message_id*, *disable_notification=False*, *timeout=None*, ***kwargs*)

Alias for `forward_message`

forward_message (*chat_id*, *from_chat_id*, *message_id*, *disable_notification=False*, *timeout=None*, ***kwargs*)

Use this method to forward messages of any kind.

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).
- **disable_notification** (*bool, optional*) – Sends the message silently. Users will receive a notification with no sound.
- **message_id** (*int*) – Message identifier in the chat specified in `from_chat_id`.
- **timeout** (*int | float, optional*) – If this value is specified, use it as
- **read timeout** (*the*) – from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

getChat (*chat_id, timeout=None, **kwargs*)

Alias for `get_chat`

getChatAdministrators (*chat_id, timeout=None, **kwargs*)

Alias for `get_chat_administrators`

getChatMember (*chat_id, user_id, timeout=None, **kwargs*)

Alias for `get_chat_member`

getChatMembersCount (*chat_id, timeout=None, **kwargs*)

Alias for `get_chat_members_count`

getFile (*file_id, timeout=None, **kwargs*)

Alias for `get_file`

getGameHighScores (*user_id, chat_id=None, message_id=None, inline_message_id=None, timeout=None, **kwargs*)

Alias for `get_game_high_scores`

getMe (*timeout=None, **kwargs*)

Alias for `get_me`

getStickerSet (*name, timeout=None, **kwargs*)

Alias for `get_sticker_set`

getUpdates (*offset=None, limit=100, timeout=0, read_latency=2.0, allowed_updates=None, **kwargs*)

Alias for `get_updates`

getUserProfilePhotos (*user_id, offset=None, limit=100, timeout=None, **kwargs*)

Alias for `get_user_profile_photos`

getWebhookInfo (*timeout=None, **kwargs*)

Alias for `get_webhook_info`

get_chat (*chat_id, timeout=None, **kwargs*)

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).

- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns `telegram.Chat`

Raises `telegram.TelegramError`

get_chat_administrators (*chat_id*, *timeout=None*, ***kwargs*)

Use this method to get a list of administrators in a chat. On success, returns an Array 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.

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns List[`telegram.ChatMember`]

Raises `telegram.TelegramError`

get_chat_member (*chat_id*, *user_id*, *timeout=None*, ***kwargs*)

Use this method to get information about a member of a 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.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns `telegram.ChatMember`

Raises `telegram.TelegramError`

get_chat_members_count (*chat_id*, *timeout=None*, ***kwargs*)

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

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns Number of members in the chat.

Return type int

Raises `telegram.TelegramError`

get_file (*file_id*, *timeout=None*, ***kwargs*)

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 20MB in size. The file can then be downloaded with `telegram.File.download`. It is guaranteed that the link will be valid for at least 1 hour. When the link expires, a new one can be requested by calling `get_file` again.

Parameters

- **file_id** (*str* | `telegram.Audio` | `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.
- **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns `telegram.File`

Raises `telegram.TelegramError`

get_game_high_scores (*user_id*, *chat_id=None*, *message_id=None*, *inline_message_id=None*, *timeout=None*, ***kwargs*)

Use this method to get data for high score tables. Will return the score of the specified user and several of his neighbors in a game

Parameters

- **user_id** (*int*) – User identifier.
- **chat_id** (*int* | *str*, optional) – Required if `inline_message_id` is not specified. Unique identifier for the target chat.
- **message_id** (*int*, optional) – Required if `inline_message_id` is not specified. Identifier of the sent message.
- **inline_message_id** (*str*, optional) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.
- **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns `List[telegram.GameHighScore]`

Raises `telegram.TelegramError`

get_me (*timeout=None*, ***kwargs*)

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

Parameters **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

Return type `telegram.User`

Raises `telegram.TelegramError`

get_sticker_set (*name*, *timeout=None*, ***kwargs*)

Use this method to get a sticker set.

Parameters

- **name** (`str`) – Short name of the sticker set that is used in `t.me/addstickers/` URLs (e.g., `animals`)
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns `telegram.StickerSet`

Raises `telegram.TelegramError`

get_updates (`offset=None`, `limit=100`, `timeout=0`, `read_latency=2.0`, `allowed_updates=None`, ****kwargs**)

Use this method to receive incoming updates using long polling.

Parameters

- **offset** (`int`, optional) – Identifier of the first update to be returned. Must be greater by one than the highest among the identifiers of previously received updates. By default, updates starting with the earliest unconfirmed update are returned. An update is considered confirmed as soon as `getUpdates` is called with an offset higher than its `update_id`. The negative offset can be specified to retrieve updates starting from -offset update from the end of the updates queue. All previous updates will forgotten.
- **limit** (`int`, optional) – Limits the number of updates to be retrieved. Values between 1-100 are accepted. Defaults to 100.
- **timeout** (`int`, optional) – Timeout in seconds for long polling. Defaults to 0, i.e. usual short polling. Should be positive, short polling should be used for testing purposes only.
- **allowed_updates** (`List[str]`), optional) – List the types of updates you want your bot to receive. For example, specify `["message", "edited_channel_post", "callback_query"]` to only receive updates of these types. See `telegram.Update` for a complete list of available update types. Specify an empty list to receive all updates regardless of type (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.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Notes

1. This method will not work if an outgoing webhook is set up.
2. In order to avoid getting duplicate updates, recalculate offset after each server response.
3. To take full advantage of this library take a look at `telegram.ext.Updater`

Returns `List[telegram.Update]`

Raises `telegram.TelegramError`

get_user_profile_photos (`user_id`, `offset=None`, `limit=100`, `timeout=None`, ****kwargs**)

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.

- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns `telegram.UserProfilePhotos`

Raises `telegram.TelegramError`

get_webhook_info (timeout=None, **kwargs)

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

If the bot is using getUpdates, will return an object with the url field empty.

Parameters

- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns `telegram.WebhookInfo`

id

Unique identifier for this bot.

Type int

kickChatMember (chat_id, user_id, timeout=None, until_date=None, **kwargs)

Alias for `kick_chat_member`

kick_chat_member (chat_id, user_id, timeout=None, until_date=None, **kwargs)

Use this method to kick a user from a group or a supergroup. In the case of supergroups, 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 group for this to work.

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.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **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.
- ****kwargs** (dict) – Arbitrary keyword arguments.

Note: In regular groups (non-supergroups), this method will only work if the ‘All Members Are Admins’ setting is off in the target group. Otherwise members may only be removed by the group’s creator or by the member that added them.

Returns bool On success, True is returned.

Raises `telegram.TelegramError`

last_name

Optional. Bot’s last name.

Type str

leaveChat (*chat_id*, *timeout=None*, ***kwargs*)

Alias for `leave_chat`

leave_chat (*chat_id*, *timeout=None*, ***kwargs*)

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).
- **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

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

Raises `telegram.TelegramError`

name

Bot's @username.

Type *str*

pinChatMessage (*chat_id*, *message_id*, *disable_notification=None*, *timeout=None*, ***kwargs*)

Alias for `pin_chat_message`

pin_chat_message (*chat_id*, *message_id*, *disable_notification=None*, *timeout=None*, ***kwargs*)

Use this method to pin a message in a supergroup. 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).
- **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 group members about the new pinned message.
- **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments

Returns Returns `True` on success.

Return type *bool*

Raises `telegram.TelegramError`

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*, *timeout=None*, ***kwargs*)

Alias for `promote_chat_member`

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*, *timeout=None*, ***kwargs*)

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

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.
- **can_change_info** (`bool`, optional) – Pass True, if the administrator can change chat title, photo and other settings.
- **can_post_messages** (`bool`, optional) – Pass True, if the administrator can create channel posts, channels only.
- **can_edit_messages** (`bool`, optional) – Pass True, if the administrator can edit messages of other users, channels only.
- **can_delete_messages** (`bool`, optional) – Pass True, if the administrator can delete messages of other users.
- **can_invite_users** (`bool`, optional) – Pass True, if the administrator can invite new users to the chat.
- **can_restrict_members** (`bool`, optional) – Pass True, if the administrator can restrict, ban or unban chat members.
- **can_pin_messages** (`bool`, optional) – Pass True, if the administrator can pin messages, supergroups only.
- **can_promote_members** (`bool`, optional) – Pass True, if the administrator can add new administrators with a subset of his own privileges or demote administrators that he has promoted, directly or indirectly (promoted by administrators that were appointed by him).
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments

Returns Returns True on success.

Return type `bool`

Raises `telegram.TelegramError`

request

restrictChatMember (*chat_id*, *user_id*, *until_date=None*, *can_send_messages=None*, *can_send_media_messages=None*, *can_send_other_messages=None*, *can_add_web_page_previews=None*, *timeout=None*, ***kwargs*)

Alias for `restrict_chat_member`

restrict_chat_member (*chat_id*, *user_id*, *until_date=None*, *can_send_messages=None*, *can_send_media_messages=None*, *can_send_other_messages=None*, *can_add_web_page_previews=None*, *timeout=None*, ***kwargs*)

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.

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.
- **can_send_messages** (bool, optional) – Pass True, if the user can send text messages, contacts, locations and venues.
- **can_send_media_messages** (bool, optional) – Pass True, if the user can send audios, documents, photos, videos, video notes and voice notes, implies can_send_messages.
- **can_send_other_messages** (bool, optional) – Pass True, if the user can send animations, games, stickers and use inline bots, implies can_send_media_messages.
- **can_add_web_page_previews** (bool, optional) – Pass True, if the user may add web page previews to their messages, implies can_send_media_messages.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments

Returns Returns True on success.

Return type bool

Raises telegram.TelegramError

sendAnimation (chat_id, animation, duration=None, width=None, height=None, thumb=None, caption=None, parse_mode=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, **kwargs)

Alias for [send_animation](#)

sendAudio (chat_id, audio, duration=None, performer=None, title=None, caption=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, thumb=None, **kwargs)

Alias for [send_audio](#)

sendChatAction (chat_id, action, timeout=None, **kwargs)

Alias for [send_chat_action](#)

sendContact (chat_id, phone_number=None, first_name=None, last_name=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=None, contact=None, vcard=None, **kwargs)

Alias for [send_contact](#)

sendDocument (chat_id, document, filename=None, caption=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, thumb=None, **kwargs)

Alias for [send_document](#)

sendGame (chat_id, game_short_name, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=None, **kwargs)

Alias for [send_game](#)

sendInvoice (chat_id, title, description, payload, provider_token, start_parameter, currency, prices, 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=False, reply_to_message_id=None, reply_markup=None, provider_data=None, send_phone_number_to_provider=None, send_email_to_provider=None, timeout=None, **kwargs)

Alias for [send_invoice](#)

sendLocation (*chat_id*, *latitude=None*, *longitude=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=None*, *location=None*, *live_period=None*, ***kwargs*)

Alias for `send_location`

sendMediaGroup (*chat_id*, *media*, *disable_notification=None*, *reply_to_message_id=None*, *timeout=20*, ***kwargs*)

Alias for `send_media_group`

sendMessage (*chat_id*, *text*, *parse_mode=None*, *disable_web_page_preview=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=None*, ***kwargs*)

Alias for `send_message`

sendPhoto (*chat_id*, *photo*, *caption=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, *parse_mode=None*, ***kwargs*)

Alias for `send_photo`

sendSticker (*chat_id*, *sticker*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, ***kwargs*)

Alias for `send_sticker`

sendVenue (*chat_id*, *latitude=None*, *longitude=None*, *title=None*, *address=None*, *foursquare_id=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=None*, *venue=None*, *foursquare_type=None*, ***kwargs*)

Alias for `send_venue`

sendVideo (*chat_id*, *video*, *duration=None*, *caption=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, *width=None*, *height=None*, *parse_mode=None*, *supports_streaming=None*, *thumb=None*, ***kwargs*)

Alias for `send_video`

sendVideoNote (*chat_id*, *video_note*, *duration=None*, *length=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, *thumb=None*, ***kwargs*)

Alias for `send_video_note`

sendVoice (*chat_id*, *voice*, *duration=None*, *caption=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, *parse_mode=None*, ***kwargs*)

Alias for `send_voice`

send_animation (*chat_id*, *animation*, *duration=None*, *width=None*, *height=None*, *thumb=None*, *caption=None*, *parse_mode=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, ***kwargs*)

Use this method to send animation files (GIF or H.264/MPEG-4 AVC video without sound).

Parameters

- **chat_id** (*int* | *str*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **animation** (*str* | *filelike object* | *telegram.Animation*) – Animation to send. Pass a *file_id* as *String* to send an animation that exists on the Telegram servers (recommended), pass an *HTTP URL* as a *String* for Telegram to get an animation from the Internet, or upload a new animation using *multipart/form-data*. Lastly you can pass an existing *telegram.Animation* object to send.
- **duration** (*int*, optional) – Duration of sent animation in seconds.
- **width** (*int*, optional) – Animation width.
- **height** (*int*, optional) – Animation height.

- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not passed as a string or file_id.
- **caption** (*str*, optional) – Animation caption (may also be used when resending animations by file_id), 0-1024 characters.
- **parse_mode** (*str*, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **disable_notification** (*bool*, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (*int*, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (*int* | *float*, optional) – Send file timeout (default: 20 seconds).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

send_audio (*chat_id*, *audio*, *duration=None*, *performer=None*, *title=None*, *caption=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, *parse_mode=None*, *thumb=None*, ***kwargs*)

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 format. On success, the sent Message is returned. Bots can currently send audio files of up to 50 MB in size, this limit may be changed in the future.

For sending voice messages, use the `sendVoice` method instead.

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

Parameters

- **chat_id** (*int* | *str*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **audio** (*str* | *filelike object* | `telegram.Audio`) – Audio file to send. Pass a file_id as String to send an audio file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an audio file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing `telegram.Audio` object to send.
- **caption** (*str*, optional) – Audio caption, 0-1024 characters.
- **parse_mode** (*str*, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **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.
- **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (*telegram.ReplyMarkup*, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.
- **timeout** (int | float, optional) – Send file timeout (default: 20 seconds).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type *telegram.Message*

Raises *telegram.TelegramError*

send_chat_action (*chat_id*, *action*, *timeout=None*, ***kwargs*)

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).

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **action** (*telegram.ChatAction* | 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.ChatAction*
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns True on success.

Return type bool

Raises *telegram.TelegramError*

send_contact (*chat_id*, *phone_number=None*, *first_name=None*, *last_name=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=None*, *contact=None*, *vcard=None*, ***kwargs*)

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.
- **contact** (*telegram.Contact*, optional) – The contact to send.
- **disable_notification** (*bool*, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (*int*, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (*telegram.ReplyMarkup*, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (*int | float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns On success, the sent *Message* is returned.

Return type *telegram.Message*

Raises *telegram.TelegramError*

send_document (*chat_id*, *document*, *filename=None*, *caption=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, *parse_mode=None*, *thumb=None*, ***kwargs*)

Use this method to send general files.

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

Parameters

- **chat_id** (*int | str*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **document** (*str | filelike object | telegram.Document*) – File to send. Pass a *file_id* as String to send a file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing *telegram.Document* object to send.
- **filename** (*str*, optional) – File name that shows in telegram message (it is useful when you send file generated by temp module, for example). Undocumented.
- **caption** (*str*, optional) – Document caption (may also be used when resending documents by *file_id*), 0-1024 characters.
- **parse_mode** (*str*, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.
- **disable_notification** (*bool*, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (*int*, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (*telegram.ReplyMarkup*, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.
- **timeout** (*int | float*, optional) – Send file timeout (default: 20 seconds).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

send_game (*chat_id, game_short_name, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=None, **kwargs*)

Use this method to send a game.

Parameters

- **chat_id** (*int | str*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **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.
- **reply_to_message_id** (*int*, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (*telegram.ReplyMarkup*, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (*int | float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

send_invoice (*chat_id, title, description, payload, provider_token, start_parameter, currency, prices, 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=False, reply_to_message_id=None, reply_markup=None, provider_data=None, send_phone_number_to_provider=None, send_email_to_provider=None, timeout=None, **kwargs*)

Use this method to send invoices.

Parameters

- **chat_id** (*int | str*) – Unique identifier for the target private chat.
- **title** (*str*) – Product name.
- **description** (*str*) – Product description.
- **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.

- **start_parameter** (*str*) – Unique deep-linking parameter that can be used to generate this invoice when used as a start parameter.
- **currency** (*str*) – Three-letter ISO 4217 currency code.
- **prices** (*List[[telegram.LabeledPrice](#)]*) – Price breakdown, a list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.).
- **provider_data** (*str | object*, optional) – JSON-encoded 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.
- **reply_to_message_id** (*int*, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (*[telegram.ReplyMarkup](#)*, optional) – Additional interface options. An inlinekeyboard. If empty, one 'Pay total price' button will be shown. If not empty, the first button must be a Pay button.
- **timeout** (*int | float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type [telegram.Message](#)

Raises [telegram.TelegramError](#)

send_location (*chat_id*, *latitude=None*, *longitude=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=None*, *location=None*, *live_period=None*, ***kwargs*)

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.
- **location** (`telegram.Location`, optional) – The location to send.
- **live_period** (`int`, optional) – Period in seconds for which the location will be updated, should be between 60 and 86400.
- **disable_notification** (`bool`, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (`int`, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns On success, the sent `Message` is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

send_media_group (`chat_id`, `media`, `disable_notification=None`, `reply_to_message_id=None`, `timeout=20`, `**kwargs`)

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

Parameters

- **chat_id** (`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **media** (`List[telegram.InputMedia]`) – An array describing photos and videos to be sent, must include 2–10 items.
- **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.
- **timeout** (`int` | `float`, optional) – Send file timeout (default: 20 seconds).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns An array of the sent `Messages`.

Return type `List[telegram.Message]`

Raises `telegram.TelegramError`


```
send_message(chat_id, text, parse_mode=None, disable_web_page_preview=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=None, **kwargs)
```

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. Also found as `telegram.constants.MAX_MESSAGE_LENGTH`.
- **parse_mode** (str) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message. See the constants in `telegram.ParseMode` for the available modes.
- **disable_web_page_preview** (bool, optional) – Disables link previews for links in this message.
- **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, the sent message is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

```
send_photo(chat_id, photo, caption=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, **kwargs)
```

Use this method to send photos.

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

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **photo** (str | filelike object | `telegram.PhotoSize`) – Photo to send. Pass a file_id as String to send a photo that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a photo from the Internet, or upload a new photo using multipart/form-data. Lastly you can pass an existing `telegram.PhotoSize` object to send.
- **caption** (str, optional) – Photo caption (may also be used when resending photos by file_id), 0-1024 characters.

- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – Send file timeout (default: 20 seconds).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

send_sticker (*chat_id*, *sticker*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, ***kwargs*)

Use this method to send .webp stickers.

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

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **sticker** (str | filelike object `telegram.Sticker`) – Sticker to send. Pass a file_id as String to send a file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a .webp file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing `telegram.Sticker` object to send.
- **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – Send file timeout (default: 20 seconds).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

send_venue (*chat_id*, *latitude=None*, *longitude=None*, *title=None*, *address=None*, *foursquare_id=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=None*, *venue=None*, *foursquare_type=None*, ***kwargs*)

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 optionally `foursquare_type`.

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”.)
- **venue** (`telegram.Venue`, optional) – The venue to send.
- **disable_notification** (`bool`, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (`int`, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns On success, the sent `Message` is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

send_video (`chat_id`, `video`, `duration=None`, `caption=None`, `disable_notification=False`, `reply_to_message_id=None`, `reply_markup=None`, `timeout=20`, `width=None`, `height=None`, `parse_mode=None`, `supports_streaming=None`, `thumb=None`, ****kwargs**)

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

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

Parameters

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

a new one using multipart/form-data. Lastly you can pass an existing `telegram.Video` object to send.

- **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.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **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.
- **reply_to_message_id** (`int`, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (`telegram.ReplyMarkup`, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or `file_id`.
- **timeout** (`int` | `float`, optional) – Send file timeout (default: 20 seconds).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

```
send_video_note(chat_id, video_note, duration=None, length=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, thumb=None, **kwargs)
```

Use this method to send video messages.

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

Parameters

- **chat_id** (`int` | `str`) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **video_note** (`str` | *filelike object* | `telegram.VideoNote`) – Video note to send. Pass a `file_id` as String to send a video note that exists on the Telegram servers (recommended) or upload a new video using multipart/form-data. Or you can pass an existing `telegram.VideoNote` object to send. Sending video notes by a URL is currently unsupported.
- **duration** (`int`, optional) – Duration of sent video in seconds.
- **length** (`int`, optional) – Video width and height

- **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (*telegram.ReplyMarkup*, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.
- **timeout** (int | float, optional) – Send file timeout (default: 20 seconds).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type *telegram.Message*

Raises *telegram.TelegramError*

send_voice (*chat_id*, *voice*, *duration=None*, *caption=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=20*, *parse_mode=None*, ***kwargs*)

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).

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

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **voice** (str | *filelike object* | *telegram.Voice*) – Voice file to send. Pass a file_id as String to send an voice file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an voice file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing *telegram.Voice* object to send.
- **caption** (str, optional) – Voice message caption, 0-1024 characters.
- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.
- **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.
- **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (*telegram.ReplyMarkup*, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – Send file timeout (default: 20 seconds).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

setChatDescription (*chat_id*, *description*, *timeout=None*, ***kwargs*)

Alias for `set_chat_description`

setChatPhoto (*chat_id*, *photo*, *timeout=20*, ***kwargs*)

Alias for `set_chat_photo`

setChatStickerSet (*chat_id*, *sticker_set_name*, *timeout=None*, ***kwargs*)

Alias for `set_chat_sticker_set`

setChatTitle (*chat_id*, *title*, *timeout=None*, ***kwargs*)

Alias for `set_chat_title`

setGameScore (*user_id*, *score*, *chat_id=None*, *message_id=None*, *inline_message_id=None*,
force=None, *disable_edit_message=None*, *timeout=None*, ***kwargs*)

Alias for `set_game_score`

setPassportDataErrors (*user_id*, *errors*, *timeout=None*, ***kwargs*)

Alias for `set_passport_data_errors`

setStickerPositionInSet (*sticker*, *position*, *timeout=None*, ***kwargs*)

Alias for `set_sticker_position_in_set`

setWebhook (*url=None*, *certificate=None*, *timeout=None*, *max_connections=40*, *al-*
lowed_updates=None, ***kwargs*)

Alias for `set_webhook`

set_chat_description (*chat_id*, *description*, *timeout=None*, ***kwargs*)

Use this method to change the description of 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*) – New chat description, 1-255 characters.
- **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments

Returns Returns True on success.

Return type `bool`

Raises `telegram.TelegramError`

set_chat_photo (*chat_id*, *photo*, *timeout=20*, ***kwargs*)

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** (*filelike object*) – New chat photo.
- **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

Note: In regular groups (non-supergroups), this method will only work if the ‘All Members Are Admins’ setting is off in the target group.

Returns Returns True on success.

Return type bool

Raises telegram.TelegramError

set_chat_sticker_set (*chat_id*, *sticker_set_name*, *timeout=None*, ***kwargs*)

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.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns True on success.

Return type bool

set_chat_title (*chat_id*, *title*, *timeout=None*, ***kwargs*)

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, 1-255 characters.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments

Note: In regular groups (non-supergroups), this method will only work if the ‘All Members Are Admins’ setting is off in the target group.

Returns Returns True on success.

Return type bool

Raises telegram.TelegramError

set_game_score (*user_id*, *score*, *chat_id=None*, *message_id=None*, *inline_message_id=None*, *force=None*, *disable_edit_message=None*, *timeout=None*, ***kwargs*)

Use this method to set the score of the specified user in a game. On success, if the message was sent

by the bot, returns the edited Message, otherwise returns True. Returns an error, if the new score is not greater than the user's current score in the chat and force is False.

Parameters

- **user_id** (int) – User identifier.
- **score** (int) – New score, must be non-negative.
- **force** (bool, optional) – Pass True, if the high score is allowed to decrease. This can be useful when fixing mistakes or banning cheaters
- **disable_edit_message** (bool, optional) – Pass True, if the game message should not be automatically edited to include the current scoreboard.
- **chat_id** (int/str, optional) – Required if inline_message_id is not specified. Unique identifier for the target chat.
- **message_id** (int, optional) – Required if inline_message_id is not specified. Identifier of the sent message.
- **inline_message_id** (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns The edited message, or if the message wasn't sent by the bot, True.

Return type `telegram.Message`

Raises

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

set_passport_data_errors (user_id, errors, timeout=None, **kwargs)

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

Use this if the data submitted by the user doesn't satisfy the standards your service requires for any reason. For example, if a birthday date seems invalid, a submitted document is blurry, a scan shows evidence of tampering, etc. Supply some details in the error message to make sure the user knows how to correct the issues.

Parameters

- **user_id** (int) – User identifier
- **errors** (List[`PassportElementError`]) – A JSON-serialized array describing the errors.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, True is returned.

Return type bool

Raises `telegram.TelegramError`

set_sticker_position_in_set (sticker, position, timeout=None, **kwargs)

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.
- **timeout** (`int | float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns On success, `True` is returned.

Return type `bool`

Raises `telegram.TelegramError`

set_webhook (`url=None`, `certificate=None`, `timeout=None`, `max_connections=40`, `allowed_updates=None`, ****kwargs**)

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

If you'd like to make sure that the Webhook request comes from Telegram, we recommend using a secret path in the URL, e.g. <https://www.example.com/<token>>. Since nobody else knows your bot's token, you can be pretty sure it's us.

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

Parameters

- **url** (`str`) – HTTPS url to send updates to. Use an empty string to remove webhook integration.
- **certificate** (`filelike`) – Upload your public key certificate so that the root certificate in use can be checked. See our self-signed guide for details. (<https://goo.gl/rw7w6Y>)
- **max_connections** (`int`, optional) – Maximum allowed number of simultaneous HTTPS connections to the webhook for update delivery, 1-100. Defaults to 40. Use lower values to limit the load on your bot's server, and higher values to increase your bot's throughput.
- **allowed_updates** (`List[str]`, optional) – List the types of updates you want your bot to receive. For example, specify `["message", "edited_channel_post", "callback_query"]` to only receive updates of these types. See [telegram.Update](#) for a complete list of available update types. Specify an empty list to receive all updates regardless of type (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.
- **timeout** (`int | float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Note:

1. You will not be able to receive updates using `get_updates` for as long as an outgoing webhook is set up.

2. To use a self-signed certificate, you need to upload your public key certificate using certificate parameter. Please upload as InputFile, sending a String will not work.
 3. Ports currently supported for Webhooks: 443, 80, 88, 8443.
-

Returns bool On success, True is returned.

Raises telegram.TelegramError

stopMessageLiveLocation (*chat_id=None, message_id=None, inline_message_id=None, reply_markup=None, timeout=None, **kwargs*)
Alias for *stop_message_live_location*

stop_message_live_location (*chat_id=None, message_id=None, inline_message_id=None, reply_markup=None, timeout=None, **kwargs*)

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) – 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.
- **inline_message_id** (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.
- **reply_markup** (*telegram.ReplyMarkup*, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

Returns On success the edited message.

Return type *telegram.Message*

to_dict ()

unbanChatMember (*chat_id, user_id, timeout=None, **kwargs*)
Alias for *unban_chat_member*

unban_chat_member (*chat_id, user_id, timeout=None, **kwargs*)
Use this method to unban a previously kicked user in a supergroup.

The user will not return to the group automatically, but will be able to join via link, etc. The bot must be an administrator in the group for this to work.

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.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns bool On success, True is returned.

Raises telegram.TelegramError

unpinChatMessage (*chat_id*, *timeout=None*, ***kwargs*)

Alias for `unpin_chat_message`

unpin_chat_message (*chat_id*, *timeout=None*, ***kwargs*)

Use this method to unpin a message in a supergroup. 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`).
- **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments

Returns Returns `True` on success.

Return type `bool`

Raises `telegram.TelegramError`

uploadStickerFile (*user_id*, *png_sticker*, *timeout=20*, ***kwargs*)

Alias for `upload_sticker_file`

upload_sticker_file (*user_id*, *png_sticker*, *timeout=20*, ***kwargs*)

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

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

Parameters

- **user_id** (*int*) – User identifier of sticker file owner.
- **png_sticker** (*str* | *filelike object*) – Png image with the sticker, must be up to 512 kilobytes in size, dimensions must not exceed 512px, and either width or height must be exactly 512px.
- **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns The uploaded File

Return type `telegram.File`

Raises `telegram.TelegramError`

username

Bot's username.

Type `str`

class `telegram.Chat` (*id*, *type*, *title=None*, *username=None*, *first_name=None*, *last_name=None*, *all_members_are_administrators=None*, *bot=None*, *photo=None*, *description=None*, *invite_link=None*, *pinned_message=None*, *sticker_set_name=None*, *can_set_sticker_set=None*, ***kwargs*)

Bases: `telegram.base.TelegramObject`

This object represents a chat.

id
Unique identifier for this chat.
Type `int`

type
Type of chat.
Type `str`

title
Optional. Title, for supergroups, channels and group chats.
Type `str`

username
Optional. Username.
Type `str`

first_name
Optional. First name of the other party in a private chat.
Type `str`

last_name
Optional. Last name of the other party in a private chat.
Type `str`

all_members_are_administrators
Optional.
Type `bool`

photo
Optional. Chat photo.
Type `telegram.ChatPhoto`

description
Optional. Description, for supergroups and channel chats.
Type `str`

invite_link
Optional. Chat invite link, for supergroups and channel chats.
Type `str`

pinned_message
Optional. Pinned message, for supergroups. Returned only in `get_chat`.
Type `telegram.Message`

sticker_set_name
Optional. For supergroups, name of Group sticker set.
Type `str`

can_set_sticker_set
Optional. True, if the bot can change group the sticker set.
Type `bool`

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.
- **all_members_are_administrators** (`bool`, optional) – True if a group has *All Members Are Admins* enabled.
- **photo** (`telegram.ChatPhoto`, optional) – Chat photo. Returned only in `getChat`.
- **description** (`str`, optional) – Description, for supergroups and channel chats. Returned only in `get_chat`.
- **invite_link** (`str`, optional) – Chat invite link, for supergroups and channel chats. Returned only in `get_chat`.
- **pinned_message** (`telegram.Message`, optional) – Pinned message, for supergroups. Returned only in `get_chat`.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.
- **sticker_set_name** (`str`, optional) – For supergroups, name of Group sticker set. Returned only in `get_chat`.
- **can_set_sticker_set** (`bool`, optional) – True, if the bot can change group the sticker set. Returned only in `get_chat`.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
CHANNEL = 'channel'
        'channel'
```

```
        Type str
```

```
GROUP = 'group'
        'group'
```

```
        Type str
```

```
PRIVATE = 'private'
        'private'
```

```
        Type str
```

```
SUPERGROUP = 'supergroup'
        'supergroup'
```

```
        Type str
```

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

```
get_administrators (*args, **kwargs)
    Shortcut for:
```

```
bot.get_chat_administrators(update.message.chat.id, *args, **kwargs)
```

Returns A list of administrators in a chat. An Array of `telegram.ChatMember` objects that contains information about all chat administrators except other bots. If the chat is a group or a supergroup and no administrators were appointed, only the creator will be returned

Return type List[`telegram.ChatMember`]

get_member (*args, **kwargs)

Shortcut for:

```
bot.get_chat_member(update.message.chat.id, *args, **kwargs)
```

Returns *telegram.ChatMember*

get_members_count (*args, **kwargs)

Shortcut for:

```
bot.get_chat_members_count(update.message.chat.id, *args, **kwargs)
```

Returns int

kick_member (*args, **kwargs)

Shortcut for:

```
bot.kick_chat_member(update.message.chat.id, *args, **kwargs)
```

Returns If the action was sent successfully.

Return type bool

Note: This method will only work if the *All Members Are Admins* setting is off in the target group. Otherwise members may only be removed by the group's creator or by the member that added them.

leave (*args, **kwargs)

Shortcut for:

```
bot.leave_chat(update.message.chat.id, *args, **kwargs)
```

Returns bool If the action was sent successfully.

link

Convenience property. If the chat has a *username*, returns a t.me link of the chat.

Type str

send_action (*args, **kwargs)

Shortcut for:

```
bot.send_chat_action(update.message.chat.id, *args, **kwargs)
```

Returns If the action was sent successfully.

Return type bool

send_animation (*args, **kwargs)

Shortcut for:

```
bot.send_animation(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_audio (*args, **kwargs)

Shortcut for:

```
bot.send_audio(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_document (*args, **kwargs)

Shortcut for:

```
bot.send_document(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_message (*args, **kwargs)

Shortcut for:

```
bot.send_message(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_photo (*args, **kwargs)

Shortcut for:

```
bot.send_photo(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_sticker (*args, **kwargs)

Shortcut for:

```
bot.send_sticker(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_video (*args, **kwargs)

Shortcut for:

```
bot.send_video(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_video_note (*args, **kwargs)

Shortcut for:

```
bot.send_video_note(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

send_voice (*args, **kwargs)

Shortcut for:

```
bot.send_voice(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

unban_member (*args, **kwargs)

Shortcut for:

```
bot.unban_chat_member(update.message.chat.id, *args, **kwargs)
```

Returns If the action was sent successfully.

Return type `bool`

```
class telegram.ChatMember(user, status, until_date=None, can_be_edited=None,
                           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,
                           can_send_messages=None, can_send_media_messages=None,
                           can_send_other_messages=None, can_add_web_page_previews=None,
                           **kwargs)
```

Bases: `telegram.base.TelegramObject`

This object contains information about one member of the chat.

user

Information about the user.

Type `telegram.User`

status

The member's status in the chat.

Type `str`

until_date

Optional. Date when restrictions will be lifted for this user.

Type `datetime.datetime`

can_be_edited

Optional. If the bot is allowed to edit administrator privileges of that user.

Type `bool`

can_change_info

Optional. If the administrator can change the chat title, photo and other settings.

Type `bool`

can_post_messages

Optional. If the administrator can post in the channel.

Type `bool`

can_edit_messages

Optional. If the administrator can edit messages of other users.

Type `bool`

can_delete_messages

Optional. If the administrator can delete messages of other users.

Type `bool`

can_invite_users

Optional. If the administrator can invite new users to the chat.

Type `bool`

can_restrict_members

Optional. If the administrator can restrict, ban or unban chat members.

Type `bool`

can_pin_messages

Optional. If the administrator can pin messages.

Type `bool`

can_promote_members

Optional. If the administrator can add new administrators.

Type `bool`

can_send_messages

Optional. If the user can send text messages, contacts, locations and venues.

Type `bool`

can_send_media_messages

Optional. If the user can send media messages, implies `can_send_messages`.

Type `bool`

can_send_other_messages

Optional. If the user can send animations, games, stickers and use inline bots, implies `can_send_media_messages`.

Type `bool`

can_add_web_page_previews

Optional. If user may add web page previews to his messages, implies `can_send_media_messages`

Type `bool`

Parameters

- **user** (`telegram.User`) – Information about the user.
- **status** (`str`) – The member's status in the chat. Can be 'creator', 'administrator', 'member', 'restricted', 'left' or 'kicked'.
- **until_date** (`datetime.datetime`, optional) – Restricted and kicked only. Date when restrictions will be lifted for this user.
- **can_be_edited** (`bool`, optional) – Administrators only. True, if the bot is allowed to edit administrator privileges of that user.
- **can_change_info** (`bool`, optional) – Administrators only. True, if the administrator can change the chat title, photo and other settings.
- **can_post_messages** (`bool`, optional) – Administrators only. True, if the administrator can post in the channel, channels only.

- **can_edit_messages** (bool, optional) – Administrators only. True, if the administrator can edit messages of other users, channels only.
- **can_delete_messages** (bool, optional) – Administrators only. True, if the administrator can delete messages of other user.
- **can_invite_users** (bool, optional) – Administrators only. True, if the administrator can invite new users to the chat.
- **can_restrict_members** (bool, optional) – Administrators only. True, if the administrator can restrict, ban or unban chat members.
- **can_pin_messages** (bool, optional) – Administrators only. True, if the administrator can pin messages, supergroups only.
- **can_promote_members** (bool, optional) – Administrators only. 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_send_messages** (bool, optional) – Restricted only. True, if the user can send text messages, contacts, locations and venues.
- **can_send_media_messages** (bool, optional) – Restricted only. True, if the user can send audios, documents, photos, videos, video notes and voice notes, implies can_send_messages.
- **can_send_other_messages** (bool, optional) – Restricted only. True, if the user can send animations, games, stickers and use inline bots, implies can_send_media_messages.
- **can_add_web_page_previews** (bool, optional) – Restricted only. True, if user may add web page previews to his messages, implies can_send_media_messages.

```
ADMINISTRATOR = 'administrator'  
    'administrator'
```

```
    Type str
```

```
CREATOR = 'creator'  
    'creator'
```

```
    Type str
```

```
KICKED = 'kicked'  
    'kicked'
```

```
    Type str
```

```
LEFT = 'left'  
    'left'
```

```
    Type str
```

```
MEMBER = 'member'  
    'member'
```

```
    Type str
```

```
RESTRICTED = 'restricted'  
    'restricted'
```

```
    Type str
```

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

```
to_dict()
```

```
class telegram.ChatAction
```

Bases: object

Helper class to provide constants for different chatactions.

```
FIND_LOCATION = 'find_location'  
    'find_location'
```

 Type str

```
RECORD_AUDIO = 'record_audio'  
    'record_audio'
```

 Type str

```
RECORD_VIDEO = 'record_video'  
    'record_video'
```

 Type str

```
RECORD_VIDEO_NOTE = 'record_video_note'  
    'record_video_note'
```

 Type str

```
TYPING = 'typing'  
    'typing'
```

 Type str

```
UPLOAD_AUDIO = 'upload_audio'  
    'upload_audio'
```

 Type str

```
UPLOAD_DOCUMENT = 'upload_document'  
    'upload_document'
```

 Type str

```
UPLOAD_PHOTO = 'upload_photo'  
    'upload_photo'
```

 Type str

```
UPLOAD_VIDEO = 'upload_video'  
    'upload_video'
```

 Type str

```
UPLOAD_VIDEO_NOTE = 'upload_video_note'  
    'upload_video_note'
```

 Type str

```
class telegram.ChosenInlineResult(result_id, from_user, query, location=None, in-  
                                line_message_id=None, **kwargs)
```

Bases: telegram.base.TelegramObject

Represents a result of an inline query that was chosen by the user and sent to their chat partner.

Note: In Python *from* is a reserved word, use *from_user* instead.

result_id

The unique identifier for the result that was chosen.

 Type str

from_user

The user that chose the result.

Type `telegram.User`

location

Optional. Sender location.

Type `telegram.Location`

inline_message_id

Optional. Identifier of the sent inline message.

Type `str`

query

The query that was used to obtain the result.

Type `str`

Parameters

- **result_id** (`str`) – The unique identifier for the result that was chosen.
- **from_user** (`telegram.User`) – The user that chose the result.
- **location** (`telegram.Location`, optional) – Sender location, only for bots that require user location.
- **inline_message_id** (`str`, optional) – Identifier of the sent inline message. Available only if there is an inline keyboard attached to the message. Will be also received in callback queries and can be used to edit the message.
- **query** (`str`) – The query that was used to obtain the result.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

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

class `telegram.CallbackQuery` (`id`, `from_user`, `chat_instance`, `message=None`, `data=None`, `inline_message_id=None`, `game_short_name=None`, `bot=None`, `**kwargs`)

Bases: `telegram.base.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.

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.
-

id

Unique identifier for this query.

Type `str`

from_user

Sender.

Type `telegram.User`

message

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

Type `telegram.Message`

inline_message_id

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

Type `str`

chat_instance

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

Type `str`

data

Optional. Data associated with the callback button.

Type `str`

game_short_name

Optional. Short name of a Game to be returned.

Type `str`

Parameters

- **id** (`str`) – Unique identifier for this query.
- **from_user** (`telegram.User`) – Sender.
- **message** (`telegram.Message`, optional) – Message with the callback button that originated the query. Note that message content and message date will not be available if the message is too old.
- **inline_message_id** (`str`, optional) – Identifier of the message sent via the bot in inline mode, that originated the query.
- **chat_instance** (`str`, optional) – Global identifier, uniquely corresponding to the chat to which the message with the callback button was sent. Useful for high scores in games.
- **data** (`str`, optional) – Data associated with the callback button. Be aware that a bad client can send arbitrary data in this field.
- **game_short_name** (`str`, optional) – Short name of a Game to be returned, serves as the unique identifier for the game

Note: 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).

answer (**args*, ***kwargs*)

Shortcut for:

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

Returns On success, `True` is returned.

Return type `bool`

classmethod de_json (*data*, *bot*)**edit_message_caption** (*caption*, **args*, ***kwargs*)

Shortcut for either:

```
bot.edit_message_caption(caption=caption,
                        chat_id=update.callback_query.message.chat_id,
                        message_id=update.callback_query.message.message_id,
                        *args, **kwargs)
```

or:

```
bot.edit_message_caption(caption=caption
                        inline_message_id=update.callback_query.inline_
↪message_id,
                        *args, **kwargs)
```

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

Return type *telegram.Message*

edit_message_reply_markup (*reply_markup*, *args, **kwargs)

Shortcut for either:

```
bot.edit_message_replyMarkup(chat_id=update.callback_query.message.chat_id,
                             message_id=update.callback_query.message.
↪message_id,
                             reply_markup=reply_markup,
                             *args, **kwargs)
```

or:

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

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

Return type *telegram.Message*

edit_message_text (*text*, *args, **kwargs)

Shortcut for either:

```
bot.edit_message_text(text, chat_id=update.callback_query.message.chat_id,
                     message_id=update.callback_query.message.message_id,
                     *args, **kwargs)
```

or:

```
bot.edit_message_text(text, inline_message_id=update.callback_query.inline_
↪message_id,
                     *args, **kwargs)
```

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

Return type *telegram.Message*

class telegram.**Contact** (*phone_number*, *first_name*, *last_name=None*, *user_id=None*,
 vcards=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents a phone contact.

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`

Parameters

- **phone_number** (`str`) – Contact's phone number.
- **first_name** (`str`) – Contact's first name.
- **last_name** (`str`, optional) – Contact's last name.
- **user_id** (`int`, optional) – Contact's user identifier in Telegram.
- **vcard** (`str`, optional) – Additional data about the contact in the form of a vCard.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

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

```
class telegram.Document (file_id, thumb=None, file_name=None, mime_type=None,
                          file_size=None, bot=None, **kwargs)
```

Bases: `telegram.base.TelegramObject`

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

file_id

Unique file identifier.

Type `str`

thumb

Optional. Document thumbnail.

Type `telegram.PhotoSize`

file_name

Original filename.

Type `str`

mime_type

Optional. MIME type of the file.

Type `str`

file_size

Optional. File size.

Type `int`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

Parameters

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

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

get_file (`timeout=None`, ****kwargs**)

Convenience wrapper over `telegram.Bot.get_file`

Parameters

- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns `telegram.File`

Raises `telegram.TelegramError`

class `telegram.File` (`file_id`, `bot=None`, `file_size=None`, `file_path=None`, ****kwargs**)

Bases: `telegram.base.TelegramObject`

This object represents a file ready to be downloaded. The file can be downloaded with `download`. It is guaranteed that the link will be valid for at least 1 hour. When the link expires, a new one can be requested by calling `getFile`.

Note: Maximum file size to download is 20 MB

file_id

Unique identifier for this file.

Type `str`

file_size

Optional. File size.

Type `str`

file_path

Optional. File path. Use `download` to get the file.

Type `str`

Parameters

- **file_id** (`str`) – Unique identifier for this file.
- **file_size** (`int`, optional) – Optional. File size, if known.
- **file_path** (`str`, optional) – File path. Use `download` to get the file.

- **bot** (*telegram.Bot*, optional) – Bot to use with shortcut method.
- ****kwargs** (dict) – Arbitrary keyword arguments.

Note: If you obtain an instance of this class from *telegram.PassportFile.get_file*, then it will automatically be decrypted as it downloads when you call *download()*.

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

download (*custom_path=None*, *out=None*, *timeout=None*)

Download this file. By default, the file is saved in the current working directory with its original filename as reported by Telegram. If a *custom_path* is supplied, it will be saved to that path instead. If *out* is defined, the file contents will be saved to that object using the *out.write* method.

Note: *custom_path* and *out* are mutually exclusive.

Parameters

- **custom_path** (str, optional) – Custom path.
- **out** (io.BufferedWriter, optional) – A file-like object. Must be opened for writing in binary mode, if applicable.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

Returns The same object as *out* if specified. Otherwise, returns the filename downloaded to.

Return type str | io.BufferedWriter

Raises ValueError – If both *custom_path* and *out* are passed.

download_as_bytearray (*buf=None*)

Download this file and return it as a bytearray.

Parameters *buf* (bytearray, optional) – Extend the given bytearray with the downloaded data.

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

Return type bytearray

set_credentials (*credentials*)

class telegram.**ForceReply** (*force_reply=True*, *selective=False*, ****kwargs**)

Bases: telegram.replymarkup.ReplyMarkup

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.

force_reply

Shows reply interface to the user.

Type True

selective

Optional. Force reply from specific users only.

Type bool

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 Message object
 - 2) if the bot's message is a reply (has `reply_to_message_id`), sender of the original message.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.InlineKeyboardButton (text,          url=None,          callback_data=None,
                                     switch_inline_query=None,
                                     switch_inline_query_current_chat=None,      call-
                                     back_game=None, pay=None, **kwargs)
```

Bases: `telegram.base.TelegramObject`

This object represents one button of an inline keyboard.

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.

text

Label text on the button.

Type `str`

url

Optional. HTTP url to be opened when button is pressed.

Type `str`

callback_data

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

Type `str`

switch_inline_query

Optional. Will prompt the user to select one of their chats, open that chat and insert the bot's username and the specified inline query in the input field.

Type `str`

switch_inline_query_current_chat

Optional. Will insert the bot's username and the specified inline query in the current chat's input field.

Type `str`

callback_game

Optional. Description of the game that will be launched when the user presses the button.

Type `telegram.CallbackGame`

pay

Optional. Specify True, to send a Pay button.

Type `bool`

Parameters

- **text** (`str`) – Label text on the button.
- **url** (`str`) – HTTP url to be opened when button is pressed.
- **callback_data** (`str`, optional) – Data to be sent in a callback query to the bot when button is pressed, 1-64 bytes.

- **switch_inline_query** (`str`, optional) – If set, pressing the button will prompt the user to select one of their chats, open that chat and insert the bot’s username and the specified inline query in the input field. Can be empty, in which case just the bot’s username will be inserted. This offers an easy way for users to start using your bot in inline mode when they are currently in a private chat with it. Especially useful when combined with `switch_pm*` actions - in this case the user will be automatically returned to the chat they switched from, skipping the chat selection screen.
- **switch_inline_query_current_chat** (`str`, optional) – If set, pressing the button will insert the bot’s username and the specified inline query in the current chat’s input field. Can be empty, in which case only the bot’s username will be inserted. This offers a quick way for the user to open your bot in inline mode in the same chat - good for selecting something from multiple options.
- **callback_game** (`telegram.CallbackGame`, optional) – Description of the game that will be launched when the user presses the button. This type of button must always be the `first` button in the first row.
- **pay** (`bool`, optional) – Specify `True`, to send a Pay button. This type of button must always be the `first` button in the first row.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

class telegram.**InlineKeyboardMarkup** (`inline_keyboard`, ****kwargs**)

Bases: telegram.replymarkup.ReplyMarkup

This object represents an inline keyboard that appears right next to the message it belongs to.

inline_keyboard

Array of button rows, each represented by an Array of `InlineKeyboardButton` objects.

Type List[List[`telegram.InlineKeyboardButton`]]

Parameters

- **inline_keyboard** (List[List[`telegram.InlineKeyboardButton`]]) – Array of button rows, each represented by an Array of `InlineKeyboardButton` objects.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

classmethod **from_button** (`button`, ****kwargs**)

Shortcut for:

```
InlineKeyboardMarkup([[button]], **kwargs)
```

Return an `InlineKeyboardMarkup` from a single `InlineKeyboardButton`

Parameters

- **button** (`telegram.InlineKeyboardButton`) – The button to use in the markup
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

classmethod **from_column** (`button_column`, ****kwargs**)

Shortcut for:

```
InlineKeyboardMarkup([[button] for button in button_column], **kwargs)
```

Return an `InlineKeyboardMarkup` from a single column of `InlineKeyboardButtons`

Parameters

- **button_column** (List[`telegram.InlineKeyboardButton`]) – The button to use in the markup
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

classmethod `from_row(button_row, **kwargs)`

Shortcut for:

`InlineKeyboardMarkup([button_row], **kwargs)`

Return an `InlineKeyboardMarkup` from a single row of `InlineKeyboardButtons`

Parameters

- **button_row** (List[`telegram.InlineKeyboardButton`]) – The button to use in the markup
- ****kwargs** (dict) – Arbitrary keyword arguments.

to_dict()

class `telegram.InlineQuery(id, from_user, query, offset, location=None, bot=None, **kwargs)`

Bases: `telegram.base.TelegramObject`

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

Note:

- In Python *from* is a reserved word, use *from_user* instead.
-

id

Unique identifier for this query.

Type `str`

from_user

Sender.

Type `telegram.User`

location

Optional. Sender location, only for bots that request user location.

Type `telegram.Location`

query

Text of the query (up to 512 characters).

Type `str`

offset

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

Type `str`

Parameters

- **id** (`str`) – Unique identifier for this query.
- **from_user** (`telegram.User`) – Sender.
- **location** (`telegram.Location`, optional) – Sender location, only for bots that request user location.
- **query** (`str`) – Text of the query (up to 512 characters).
- **offset** (`str`) – Offset of the results to be returned, can be controlled by the bot.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.
- ****kwargs** (dict) – Arbitrary keyword arguments.

answer (*args, **kwargs)

Shortcut for:

```
bot.answer_inline_query(update.inline_query.id, *args, **kwargs)
```

Parameters

- **results** (List[*telegram.InlineQueryResult*]) – A list of results for the inline query.
- **cache_time** (int, optional) – The maximum amount of time in seconds that the result of the inline query may be cached on the server. Defaults to 300.
- **is_personal** (bool, optional) – Pass True, if results may be cached on the server side only for the user that sent the query. By default, results may be returned to any user who sends the same query.
- **next_offset** (str, optional) – Pass the offset that a client should send in the next query with the same text to receive more results. Pass an empty string if there are no more results or if you don't support pagination. Offset length can't exceed 64 bytes.
- **switch_pm_text** (str, optional) – If passed, clients will display a button with specified text that switches the user to a private chat with the bot and sends the bot a start message with the parameter switch_pm_parameter.
- **switch_pm_parameter** (str, optional) – Deep-linking parameter for the /start message sent to the bot when user presses the switch button. 1-64 characters, only A-Z, a-z, 0-9, _ and - are allowed.

classmethod de_json (data, bot)

class telegram.*InlineQueryResult* (type, id, **kwargs)

Bases: telegram.base.TelegramObject

Baseclass for the *InlineQueryResult** classes.

type

Type of the result.

Type str

id

Unique identifier for this result, 1-64 Bytes.

Type str

Parameters

- **type** (str) – Type of the result.
- **id** (str) – Unique identifier for this result, 1-64 Bytes.
- ****kwargs** (dict) – Arbitrary keyword arguments.

class telegram.*InlineQueryResult* (type, id, **kwargs)

Bases: telegram.base.TelegramObject

Baseclass for the *InlineQueryResult** classes.

type

Type of the result.

Type str

id

Unique identifier for this result, 1-64 Bytes.

Type str

Parameters

- **type** (*str*) – Type of the result.
- **id** (*str*) – Unique identifier for this result, 1-64 Bytes.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultArticle (id, title, input_message_content,  
                                         reply_markup=None, url=None,  
                                         hide_url=None, description=None,  
                                         thumb_url=None, thumb_width=None,  
                                         thumb_height=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.InlineQueryResult`

This object represents a Telegram `InlineQueryResultArticle`.

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.ReplyMarkup`

url

Optional. URL of the result.

Type `str`

hide_url

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

Type `bool`

description

Optional. Short description of the result.

Type `str`

thumb_url

Optional. Url of the thumbnail for the result.

Type `str`

thumb_width

Optional. Thumbnail width.

Type `int`

thumb_height

Optional. Thumbnail height.

Type `int`

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.ReplyMarkup*, optional) – Inline keyboard attached to the message
- **url** (*str*, optional) – URL of the result.
- **hide_url** (*bool*, optional) – Pass True, if you don't want the URL to be shown in the message.
- **description** (*str*, optional) – Short description of the result.
- **thumb_url** (*str*, optional) – Url of the thumbnail for the result.
- **thumb_width** (*int*, optional) – Thumbnail width.
- **thumb_height** (*int*, optional) – Thumbnail height.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultAudio(id, audio_url, title, performer=None,
                                       audio_duration=None, caption=None,
                                       reply_markup=None, input_message_content=None, parse_mode=None,
                                       **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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. Caption, 0-200 characters.

Type *str*

audio_duration

Optional. Performer.

Type *str*

caption

Optional. Audio duration in seconds.

Type *str*

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

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 audio.

Type `telegram.InputMessageContent`

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) – Caption, 0-200 characters.
- **audio_duration** (`str`, optional) – Performer.
- **caption** (`str`, optional) – Audio duration in seconds.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the audio.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultCachedAudio(id, audio_file_id, caption=None,
                                             reply_markup=None, input_message_content=None,
                                             parse_mode=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

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 audio.

Type `telegram.InputMessageContent`

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
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the audio.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultCachedDocument (id, title, document_file_id, de-  
                                                scription=None, caption=None,  
                                                reply_markup=None, in-  
                                                put_message_content=None,  
                                                parse_mode=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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, 0-1024 characters

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption.. See the constants in `telegram.ParseMode` for the available modes.

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`

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, 0-1024 characters
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption.. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the file.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultCachedGif(id, gif_file_id, title=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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, 0-1024 characters

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

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 gif.

Type `telegram.InputMessageContent`

Parameters

- **id** (`str`) – Unique identifier for this result, 1-64 bytes.
- **gif_file_id** (`str`) – A valid file identifier for the GIF file.
- **title** (`str`, optional) – Title for the result.caption (`str`, optional):
- **caption** (`str`, optional) – Caption, 0-1024 characters
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the gif.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultCachedMpeg4Gif(id, mpeg4_file_id, title=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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, 0-1024 characters

Type `str`

parse_mode

Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

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 MPEG-4 file.

Type `telegram.InputMessageContent`

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, 0-1024 characters
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the MPEG-4 file.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultCachedPhoto(id, photo_file_id, title=None, de-
                                             scription=None, caption=None,
                                             reply_markup=None, in-
                                             put_message_content=None,
                                             parse_mode=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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, 0-1024 characters
 Type `str`

parse_mode
 Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
 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 photo.
 Type `telegram.InputMessageContent`

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, 0-1024 characters
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the photo.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultCachedSticker(id, sticker_file_id, re-  
                                              ply_markup=None, in-  
                                              put_message_content=None,  
                                              **kwargs)
```

Bases: telegram.inline.inlinequeryresult.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.

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`

Parameters

- **id**(str) –
- **sticker_file_id**(str) –
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the sticker.
- ****kwargs** (dict) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultCachedVideo(id, video_file_id, title, description=None,  
                                              caption=None, reply_markup=None,  
                                              input_message_content=None,  
                                              parse_mode=None, **kwargs)
```

Bases: telegram.inline.inlinequeryresult.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.

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, 0-1024 characters.

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

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.

Type `telegram.InputMessageContent`

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, 0-1024 characters.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the video.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultCachedVoice(id, voice_file_id, title, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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.
 Type str

parse_mode
 Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.
 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 voice.
 Type *telegram.InputMessageContent*

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.
- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.
- **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.
- ****kwargs** (dict) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultContact (id, phone_number, first_name,
                                         last_name=None, reply_markup=None,
                                         input_message_content=None,
                                         thumb_url=None, thumb_width=None,
                                         thumb_height=None, vcard=None, **kwargs)
Bases: telegram.inline.inlinequeryresult.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.

type

'contact'.

Type `str`

id

Unique identifier for this result, 1-64 bytes.

Type `str`

phone_number

Contact's phone number.

Type `str`

first_name

Contact's first name.

Type `str`

last_name

Optional. Contact's last name.

Type `str`

vcard

Optional. Additional data about the contact in the form of a vCard, 0-2048 bytes.

Type `str`

reply_markup

Optional. Inline keyboard attached to the message.

Type `telegram.InlineKeyboardMarkup`

input_message_content

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

Type `telegram.InputMessageContent`

thumb_url

Optional. Url of the thumbnail for the result.

Type `str`

thumb_width

Optional. Thumbnail width.

Type `int`

thumb_height

Optional. Thumbnail height.

Type `int`

Parameters

- **id** (`str`) – Unique identifier for this result, 1-64 bytes.
- **phone_number** (`str`) – Contact's phone number.
- **first_name** (`str`) – Contact's first name.
- **last_name** (`str`, optional) – Contact's last name.
- **vcard** (`str`, optional) – Additional data about the contact in the form of a vCard, 0-2048 bytes.

- **reply_markup** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
- **input_message_content** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the contact.
- **thumb_url** (str, optional) – Url of the thumbnail for the result.
- **thumb_width** (int, optional) – Thumbnail width.
- **thumb_height** (int, optional) – Thumbnail height.
- ****kwargs** (dict) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultDocument (id, document_url, title, mime_type,
                                           caption=None, description=None,
                                           reply_markup=None, input_message_content=None,
                                           thumb_url=None, thumb_width=None,
                                           thumb_height=None, parse_mode=None,
                                           **kwargs)
```

Bases: telegram.inline.inlinequeryresult.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.

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, 0-1024 characters

Type str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.

Type str

document_url

A valid URL for the file.

Type str

mime_type

Mime type of the content of the file, either “application/pdf” or “application/zip”.

Type str

description

Optional. Short description of the result.

Type str

reply_markup

Optional. Inline keyboard attached to the message.

Type `telegram.InlineKeyboardMarkup`

input_message_content

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

Type `telegram.InputMessageContent`

thumb_url

Optional. URL of the thumbnail (jpeg only) for the file.

Type `str`

thumb_width

Optional. Thumbnail width.

Type `int`

thumb_height

Optional. Thumbnail height.

Type `int`

Parameters

- **id** (`str`) – Unique identifier for this result, 1-64 bytes.
- **title** (`str`) – Title for the result.
- **caption** (`str`, optional) – Caption, 0-1024 characters
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **document_url** (`str`) – A valid URL for the file.
- **mime_type** (`str`) – Mime type of the content of the file, either “application/pdf” or “application/zip”.
- **description** (`str`, optional) – Short description of the result.
- **reply_markup** (`telegram.InlineKeyboardMarkup`) – Optional. Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`) – Optional. Content of the message to be sent instead of the file.
- **thumb_url** (`str`, optional) – URL of the thumbnail (jpeg only) for the file.
- **thumb_width** (`int`, optional) – Thumbnail width.
- **thumb_height** (`int`, optional) – Thumbnail height.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultGif(id, gif_url, thumb_url, gif_width=None,
                                     gif_height=None, title=None, caption=None, reply_markup=None,
                                     input_message_content=None, gif_duration=None, parse_mode=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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.

Type `int`

thumb_url
URL of the static thumbnail for the result (jpeg or gif).

Type `str`

title
Optional. Title for the result.

Type `str`

caption
Optional. Caption, 0-1024 characters

Type `str`

parse_mode
Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in [*telegram.ParseMode*](#) for the available modes.

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 gif.

Type [*telegram.InputMessageContent*](#)

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
- **thumb_url** (`str`) – URL of the static thumbnail for the result (jpeg or gif).

- **title** (str, optional) – Title for the result.caption (str, optional):
- **caption** (str, optional) – Caption, 0-1024 characters
- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the gif.
- ****kwargs** (dict) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultLocation(id, latitude, longitude, title,
                                          live_period=None, reply_markup=None,
                                          input_message_content=None,
                                          thumb_url=None, thumb_width=None,
                                          thumb_height=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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

live_period

Optional. Period in seconds for which the location can be updated, should be between 60 and 86400.

Type int

reply_markup

Optional. Inline keyboard attached to the message.

Type `telegram.InlineKeyboardMarkup`

input_message_content

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

Type `telegram.InputMessageContent`

thumb_url

Optional. Url of the thumbnail for the result.

Type str

thumb_width
Optional. Thumbnail width.

Type `int`

thumb_height
Optional. Thumbnail height.

Type `int`

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.
- **live_period** (`int`, optional) – Period in seconds for which the location can be updated, should be between 60 and 86400.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the location.
- **thumb_url** (`str`, optional) – Url of the thumbnail for the result.
- **thumb_width** (`int`, optional) – Thumbnail width.
- **thumb_height** (`int`, optional) – Thumbnail height.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultMpeg4Gif (id, mpeg4_url, thumb_url,
                                           mpeg4_width=None, mpeg4_height=None,
                                           title=None, caption=None,
                                           reply_markup=None, input_message_content=None,
                                           mpeg4_duration=None, parse_mode=None,
                                           **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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.

Type `int`

thumb_url

URL of the static thumbnail (jpeg or gif) for the result.

Type `str`

title

Optional. Title for the result.

Type `str`

caption

Optional. Caption, 0-1024 characters

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

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 MPEG-4 file.

Type `telegram.InputMessageContent`

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.
- **thumb_url** (`str`) – URL of the static thumbnail (jpeg or gif) for the result.
- **title** (`str`, optional) – Title for the result.
- **caption** (`str`, optional) – Caption, 0-1024 characters
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the MPEG-4 file.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultPhoto(id, photo_url, thumb_url, photo_width=None,
                                     photo_height=None, title=None, description=None,
                                     caption=None, reply_markup=None,
                                     input_message_content=None,
                                     parse_mode=None, **kwargs)
```

Bases: telegram.inline.inlinequeryresult.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.

type
 'photo'.

Type str

id
 Unique identifier for this result, 1-64 bytes.

Type str

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

Type str

thumb_url
 URL of the thumbnail for the photo.

Type str

photo_width
 Optional. Width of the photo.

Type int

photo_height
 Optional. Height of the photo.

Type int

title
 Optional. Title for the result.

Type str

description
 Optional. Short description of the result.

Type str

caption
 Optional. Caption, 0-1024 characters

Type str

parse_mode
 Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.

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 photo.

Type `telegram.InputMessageContent`

Parameters

- **id** (`str`) – Unique identifier for this result, 1-64 bytes.
- **photo_url** (`str`) – A valid URL of the photo. Photo must be in jpeg format. Photo size must not exceed 5MB.
- **thumb_url** (`str`) – URL of the thumbnail for the photo.
- **photo_width** (`int`, optional) – Width of the photo.
- **photo_height** (`int`, optional) – Height of the photo.
- **title** (`str`, optional) – Title for the result.
- **description** (`str`, optional) – Short description of the result.
- **caption** (`str`, optional) – Caption, 0-1024 characters
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the photo.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultVenue (id, latitude, longitude, title,
                                         address, foursquare_id=None,
                                         foursquare_type=None, reply_markup=None,
                                         input_message_content=None, thumb_url=None,
                                         thumb_width=None, thumb_height=None,
                                         **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.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.

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`

reply_markup

Optional. Inline keyboard attached to the message.

Type `telegram.InlineKeyboardMarkup`

input_message_content

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

Type `telegram.InputMessageContent`

thumb_url

Optional. Url of the thumbnail for the result.

Type `str`

thumb_width

Optional. Thumbnail width.

Type `int`

thumb_height

Optional. Thumbnail height.

Type `int`

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”).)
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the location.
- **thumb_url** (`str`, optional) – Url of the thumbnail for the result.
- **thumb_width** (`int`, optional) – Thumbnail width.
- **thumb_height** (`int`, optional) – Thumbnail height.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultVideo(id, video_url, mime_type, thumb_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,  
                                     **kwargs)
```

Bases: telegram.inline.inlinequeryresult.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.

type

‘video’.

Type str

id

Unique identifier for this result, 1-64 bytes.

Type str

video_url

A valid URL for the embedded video player or video file.

Type str

mime_type

Mime type of the content of video url, “text/html” or “video/mp4”.

Type str

thumb_url

URL of the thumbnail (jpeg only) for the video.

Type str

title

Title for the result.

Type str

caption

Optional. Caption, 0-1024 characters

Type str

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

Type str

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.

Type `telegram.InputMessageContent`

Parameters

- **id** (`str`) – Unique identifier for this result, 1-64 bytes.
- **video_url** (`str`) – A valid URL for the embedded video player or video file.
- **mime_type** (`str`) – Mime type of the content of video url, “text/html” or “video/mp4”.
- **thumb_url** (`str`) – URL of the thumbnail (jpeg only) for the video.
- **title** (`str`) – Title for the result.
- **caption** (`str`, optional) – Caption, 0-1024 characters.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **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.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultVoice(id, voice_url, title, voice_duration=None,  
                                     caption=None, reply_markup=None, input_message_content=None, parse_mode=None,  
                                     **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.InlineQueryResult`

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

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

Voice message title.

Type `str`

caption

Optional. Caption, 0-1024 characters.

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption.. See the constants in `telegram.ParseMode` for the available modes.

Type `str`

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.

Type `telegram.InputMessageContent`

Parameters

- **id** (`str`) – Unique identifier for this result, 1-64 bytes.
- **voice_url** (`str`) – A valid URL for the voice recording.
- **title** (`str`) – Voice message title.
- **caption** (`str`, optional) – Caption, 0-1024 characters.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption.. See the constants in `telegram.ParseMode` for the available modes.
- **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.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.InlineQueryResultGame(id, game_short_name, reply_markup=None,
                                     **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.InlineQueryResult`

Represents a Game.

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`

Parameters

- **id** (`str`) – Unique identifier for this result, 1-64 bytes.
- **game_short_name** (`str`) – Short name of the game.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.InputContactMessageContent (phone_number, first_name,  
                                           last_name=None, vcard=None, **kwargs)
```

Bases: `telegram.inline.inputmessagecontent.InputMessageContent`

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

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`

Parameters

- **phone_number** (`str`) – Contact's phone number.
- **first_name** (`str`) – Contact's first name.
- **last_name** (`str`, optional) – Contact's last name.
- **vcard** (`str`, optional) – Additional data about the contact in the form of a vCard, 0-2048 bytes.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

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

Bases: `object`

This object represents a Telegram InputFile.

input_file_content

The binary content of the file to send.

Type bytes

filename

Optional, Filename for the file to be sent.

Type str

attach

Optional, attach id for sending multiple files.

Type str

Parameters

- **obj** (File handler) – An open file descriptor.
- **filename** (str, optional) – Filename for this InputFile.
- **attach** (bool, optional) – Whether this should be send as one file or is part of a collection of files.

Raises TelegramError

field_tuple

static is_file (obj)

static is_image (stream)

Check if the content file is an image by analyzing its headers.

Parameters **stream** (str) – A str representing the content of a file.

Returns The str mime-type of an image.

Return type str

to_dict ()

class telegram.**InputLocationMessageContent** (*latitude, longitude, live_period=None, **kwargs*)

Bases: telegram.inline.inputmessagecontent.InputMessageContent

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

latitude

Latitude of the location in degrees.

Type float

longitude

Longitude of the location in degrees.

Type float

Parameters

- **latitude** (float) – Latitude of the location in degrees.
- **longitude** (float) – Longitude of the location in degrees.
- **live_period** (int, optional) – Period in seconds for which the location can be updated, should be between 60 and 86400.
- ****kwargs** (dict) – Arbitrary keyword arguments.

class telegram.**InputMessageContent**

Bases: telegram.base.TelegramObject

Base class for Telegram InputMessageContent Objects.

See: `telegram.InputContactMessageContent`, `telegram.InputLocationMessageContent`, `telegram.InputTextMessageContent` and `telegram.InputVenueMessageContent` for more details.

class `telegram.InputTextMessageContent` (`message_text`, `parse_mode=None`, `disable_web_page_preview=None`, `**kwargs`)

Bases: `telegram.inline.inputmessagecontent.InputMessageContent`

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

message_text

Text of the message to be sent, 1-4096 characters.

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.

Type `str`

disable_web_page_preview

Optional. Disables link previews for links in the sent message.

Type `bool`

Parameters

- **message_text** (`str`) – Text of the message to be sent, 1-4096 characters. Also found as `telegram.constants.MAX_MESSAGE_LENGTH`.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
- **disable_web_page_preview** (`bool`, optional) – Disables link previews for links in the sent message.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

class `telegram.InputVenueMessageContent` (`latitude`, `longitude`, `title`, `address`, `foursquare_id=None`, `foursquare_type=None`, `**kwargs`)

Bases: `telegram.inline.inputmessagecontent.InputMessageContent`

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

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`

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

```
class telegram.KeyboardButton(text, request_contact=None, request_location=None,
                             **kwargs)
```

Bases: `telegram.base.TelegramObject`

This object represents one button of the reply keyboard. For simple text buttons String can be used instead of this object to specify text of the button.

Note: Optional fields are mutually exclusive.

text

Text of the button.

Type `str`

request_contact

Optional. If the user’s phone number will be sent.

Type `bool`

request_location

Optional. If the user’s current location will be sent.

Type `bool`

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.

Note: `request_contact` and `request_location` options will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

```
class telegram.Location(longitude, latitude, **kwargs)
```

Bases: `telegram.base.TelegramObject`

This object represents a point on the map.

longitude

Longitude as defined by sender.

Type float

latitude

Latitude as defined by sender.

Type float

Parameters

- **longitude** (float) – Longitude as defined by sender.
- **latitude** (float) – Latitude as defined by sender.
- ****kwargs** (dict) – Arbitrary keyword arguments.

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

class telegram.**EncryptedCredentials** (*data*, *hash*, *secret*, *bot=None*, ***kwargs*)

Bases: telegram.base.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.

data

Decrypted data with unique user's nonce, data hashes and secrets used for EncryptedPassportElement decryption and authentication or base64 encrypted data.

Type *telegram.Credentials* or str

hash

Base64-encoded data hash for data authentication.

Type str

secret

Decrypted or encrypted secret used for decryption.

Type str

Parameters

- **data** (*telegram.Credentials* or str) – Decrypted data with unique user's nonce, data hashes and secrets used for EncryptedPassportElement decryption and authentication or base64 encrypted data.
- **hash** (str) – Base64-encoded data hash for data authentication.
- **secret** (str) – Decrypted or encrypted secret used for decryption.
- ****kwargs** (dict) – Arbitrary keyword arguments.

Note: This object is decrypted only when originating from *telegram.PassportData.decrypted_credentials*.

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

decrypted_data

Lazily decrypt and return credentials data. This object also contains the user specified nonce as *decrypted_data.nonce*.

Raises telegram.TelegramDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type *telegram.Credentials*

decrypted_secret

Lazily decrypt and return secret.

Raises telegram.TelegramDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type *str*

class telegram.PassportFile(*file_id, file_date, file_size=None, bot=None, credentials=None, **kwargs*)

Bases: telegram.base.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.

file_id

Unique identifier for this file.

Type *str*

file_size

File size.

Type *int*

file_date

Unix time when the file was uploaded.

Type *int*

bot

Optional. The Bot to use for instance methods.

Type *telegram.Bot*

Parameters

- **file_id** (*str*) – Unique identifier for this file.
- **file_size** (*int*) – File size.
- **file_date** (*int*) – Unix time when the file was uploaded.
- **bot** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

classmethod de_json(*data, bot*)

classmethod de_json_decrypted(*data, bot, credentials*)

classmethod de_list(*data, bot*)

classmethod de_list_decrypted(*data, bot, credentials*)

get_file (*timeout=None, **kwargs*)

Wrapper over *telegram.Bot.get_file*. Will automatically assign the correct credentials to the returned *telegram.File* if originating from *telegram.PassportData.decrypted_data*.

Parameters

- **timeout** (*int | float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns *telegram.File*

Raises telegram.TelegramError

```
class telegram.EncryptedPassportElement (type, data=None, phone_number=None,
                                         email=None, files=None, front_side=None,
                                         reverse_side=None, selfie=None, translation=None, hash=None, bot=None, credentials=None, **kwargs)
```

Bases: telegram.base.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.

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

data

Optional. Decrypted or encrypted data, available for “personal_details”, “passport”, “driver_license”, “identity_card”, “identity_passport” and “address” types.

Type *telegram.PersonalDetails* or telegram.IdDocument or *telegram.ResidentialAddress* or str

phone_number

Optional. User’s verified phone number, available only for “phone_number” type.

Type str

email

Optional. User’s verified email address, available only for “email” type.

Type str

files

Optional. Array of encrypted/decrypted files with documents provided by the user, available for “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.

Type List[*telegram.PassportFile*]

front_side

Optional. Encrypted/decrypted file with the front side of the document, provided by the user. Available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

Type *telegram.PassportFile*

reverse_side

Optional. Encrypted/decrypted file with the reverse side of the document, provided by the user. Available for “driver_license” and “identity_card”.

Type *telegram.PassportFile*

selfie

Optional. Encrypted/decrypted file with the selfie of the user holding a document, provided by the user; available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

Type *telegram.PassportFile*

translation

Optional. Array of encrypted/decrypted files with translated versions of documents provided by the user. Available if requested for “passport”, “driver_license”, “identity_card”, “internal_passport”, “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.

Type List[*telegram.PassportFile*]

hash

Base64-encoded element hash for using in `telegram.PassportElementErrorUnspecified`.

Type `str`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

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”.
- **data** (`telegram.PersonalDetails` or `telegram.IdDocument` or `telegram.ResidentialAddress` or `str`, optional) – Decrypted or encrypted data, available for “personal_details”, “passport”, “driver_license”, “identity_card”, “identity_passport” and “address” types.
- **phone_number** (`str`, optional) – User’s verified phone number, available only for “phone_number” type.
- **email** (`str`, optional) – User’s verified email address, available only for “email” type.
- **files** (`List[telegram.PassportFile]`, optional) – Array of encrypted/decrypted files with documents provided by the user, available for “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.
- **front_side** (`telegram.PassportFile`, optional) – Encrypted/decrypted file with the front side of the document, provided by the user. Available for “passport”, “driver_license”, “identity_card” and “internal_passport”.
- **reverse_side** (`telegram.PassportFile`, optional) – Encrypted/decrypted file with the reverse side of the document, provided by the user. Available for “driver_license” and “identity_card”.
- **selfie** (`telegram.PassportFile`, optional) – Encrypted/decrypted file with the selfie of the user holding a document, provided by the user; available for “passport”, “driver_license”, “identity_card” and “internal_passport”.
- **translation** (`List[telegram.PassportFile]`, optional) – Array of encrypted/decrypted files with translated versions of documents provided by the user. Available if requested for “passport”, “driver_license”, “identity_card”, “internal_passport”, “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.
- **hash** (`str`) – Base64-encoded element hash for using in `telegram.PassportElementErrorUnspecified`.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Note: This object is decrypted only when originating from `telegram.PassportData.decrypted_data`.

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

classmethod `de_json_decrypted` (`data`, `bot`, `credentials`)

classmethod `de_list` (`data`, `bot`)

`to_dict()`

class telegram.PassportData(*data*, *credentials*, *bot=None*, ***kwargs*)

Bases: telegram.base.TelegramObject

Contains information about Telegram Passport data shared with the bot by the user.

data

Array with encrypted information about documents and other Telegram Passport elements that was shared with the bot.

Type List[*telegram.EncryptedPassportElement*]

credentials

Encrypted credentials.

Type *telegram.EncryptedCredentials*

bot

The Bot to use for instance methods.

Type *telegram.Bot*, optional

Parameters

- **data** (List[*telegram.EncryptedPassportElement*]) – Array with encrypted information about documents and other Telegram Passport elements that was shared with the bot.
- **credentials** (str) – Encrypted credentials.
- **bot** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- ****kwargs** (dict) – Arbitrary keyword arguments.

Note: To be able to decrypt this object, you must pass your `private_key` to either `telegram.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.payload`.

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

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.TelegramDecryptionError` – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type *telegram.Credentials*

decrypted_data

Lazily decrypt and return information about documents and other Telegram Passport elements which were shared with the bot.

Raises `telegram.TelegramDecryptionError` – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type List[*telegram.EncryptedPassportElement*]

`to_dict()`

```
class telegram.Message(message_id, from_user, date, chat, 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=False, group_chat_created=False, supergroup_chat_created=False, channel_chat_created=False, 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, bot=None, **kwargs)
```

Bases: telegram.base.TelegramObject

This object represents a message.

Note:

- In Python *from* is a reserved word, use *from_user* instead.
-

message_id

Unique message identifier inside this chat.

Type `int`

from_user

Optional. Sender.

Type `telegram.User`

date

Date the message was sent.

Type `datetime.datetime`

chat

Conversation the message belongs to.

Type `telegram.Chat`

forward_from

Optional. Sender of the original message.

Type `telegram.User`

forward_from_chat

Optional. Information about the original channel.

Type `telegram.Chat`

forward_from_message_id

Optional. Identifier of the original message in the channel.

Type `int`

forward_date

Optional. Date the original message was sent.

Type `datetime.datetime`

reply_to_message

Optional. The original message.

Type *telegram.Message*

edit_date

Optional. Date the message was last edited.

Type *datetime.datetime*

media_group_id

Optional. The unique identifier of a media message group this message belongs to.

Type *str*

text

Optional. The actual UTF-8 text of the message.

Type *str*

entities

Optional. Special entities like usernames, URLs, bot commands, etc. that appear in the text. See *Message.parse_entity* and *parse_entities* methods for how to use properly.

Type *List[telegram.MessageEntity]*

caption_entities

Optional. 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.

Type *List[telegram.MessageEntity]*

audio

Optional. Information about the file.

Type *telegram.Audio*

document

Optional. Information about the file.

Type *telegram.Document*

animation

For backward compatibility, when this field is set, the document field will also be set.

Type *telegram.Animation*

game

Optional. Information about the game.

Type *telegram.Game*

photo

Optional. Available sizes of the photo.

Type *List[telegram.PhotoSize]*

sticker

Optional. Information about the sticker.

Type *telegram.Sticker*

video

Optional. Information about the video.

Type *telegram.Video*

voice

Optional. Information about the file.

Type *telegram.Voice*

video_note

Optional. Information about the video message.

Type `telegram.VideoNote`

new_chat_members

Optional. Information about new members to the chat. (the bot itself may be one of these members).

Type List[`telegram.User`]

caption

Optional. Caption for the document, photo or video, 0-1024 characters.

Type `str`

contact

Optional. Information about the contact.

Type `telegram.Contact`

location

Optional. Information about the location.

Type `telegram.Location`

venue

Optional. Information about the venue.

Type `telegram.Venue`

left_chat_member

Optional. Information about the user that left the group. (this member may be the bot itself).

Type `telegram.User`

new_chat_title

Optional. A chat title was changed to this value.

Type `str`

new_chat_photo

Optional. A chat photo was changed to this value.

Type List[`telegram.PhotoSize`]

delete_chat_photo

Optional. The chat photo was deleted.

Type `bool`

group_chat_created

Optional. The group has been created.

Type `bool`

supergroup_chat_created

Optional. The supergroup has been created.

Type `bool`

channel_chat_created

Optional. The channel has been created.

Type `bool`

migrate_to_chat_id

Optional. The group has been migrated to a supergroup with the specified identifier.

Type `int`

migrate_from_chat_id

Optional. The supergroup has been migrated from a group with the specified identifier.

Type `int`

pinned_message

Optional. Specified message was pinned.

Type `telegram.message`

invoice

Optional. Information about the invoice.

Type `telegram.Invoice`

successful_payment

Optional. Information about the payment.

Type `telegram.SuccessfulPayment`

connected_website

Optional. The domain name of the website on which the user has logged in.

Type `str`

forward_signature

Optional. Signature of the post author for messages forwarded from channels.

Type `str`

author_signature

Optional. Signature of the post author for messages in channels.

Type `str`

passport_data

Optional. Telegram Passport data

Type `telegram.PassportData`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

Parameters

- **message_id** (`int`) – Unique message identifier inside this chat.
- **from_user** (`telegram.User`, optional) – Sender, can be empty for messages sent to channels.
- **date** (`datetime.datetime`) – Date the message was sent in Unix time. Converted to `datetime.datetime`.
- **chat** (`telegram.Chat`) – Conversation the message belongs to.
- **forward_from** (`telegram.User`, optional) – For forwarded messages, sender of the original message.
- **forward_from_chat** (`telegram.Chat`, optional) – For messages forwarded from a channel, information about the original channel.
- **forward_from_message_id** (`int`, optional) – For forwarded channel posts, identifier of the original message in the channel.
- **forward_date** (`datetime.datetime`, optional) – For forwarded messages, date the original message was sent in Unix time. Converted to `datetime.datetime`.
- **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`.

- **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. Also found as `telegram.constants.MAX_MESSAGE_LENGTH`.
- **entities** (`List[telegram.MessageEntity]`, optional) – For text messages, special entities like usernames, URLs, bot commands, etc. that appear in the text. See `attr:parse_entity` and `attr:parse_entities` methods for how to use properly.
- **caption_entities** (`List[telegram.MessageEntity]`) – Optional. For Messages with a Caption. Special entities like usernames, URLs, bot commands, etc. that appear in the caption. See `Message.parse_caption_entity` and `parse_caption_entities` methods for how to use properly.
- **audio** (`telegram.Audio`, optional) – Message is an audio file, information about the file.
- **document** (`telegram.Document`, optional) – Message is a general file, information about the file.
- **animation** (`telegram.Animation`, optional) – Message is an animation, information about the animation. For backward compatibility, when this field is set, the document field will also be set.
- **game** (`telegram.Game`, optional) – Message is a game, information about the game.
- **photo** (`List[telegram.PhotoSize]`, optional) – Message is a photo, available sizes of the photo.
- **sticker** (`telegram.Sticker`, optional) – Message is a sticker, information about the sticker.
- **video** (`telegram.Video`, optional) – Message is a video, information about the video.
- **voice** (`telegram.Voice`, optional) – Message is a voice message, information about the file.
- **video_note** (`telegram.VideoNote`, optional) – Message is a video note, information about the video message.
- **new_chat_members** (`List[telegram.User]`, optional) – New members that were added to the group or supergroup and information about them (the bot itself may be one of these members).
- **caption** (`str`, optional) – Caption for the document, photo or video, 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.
- **left_chat_member** (`telegram.User`, optional) – A member was removed from the group, information about them (this member may be the bot itself).
- **new_chat_title** (`str`, optional) – A chat title was changed to this value.
- **new_chat_photo** (`List[telegram.PhotoSize]`, optional) – A chat photo was change to this value.
- **delete_chat_photo** (`bool`, optional) – Service message: The chat photo was deleted.

- **group_chat_created** (`bool`, optional) – Service message: The group has been created.
- **supergroup_chat_created** (`bool`, optional) – Service message: The supergroup has been created. This field can't be received in a message coming through updates, because bot can't be a member of a supergroup when it is created. It can only be found in `reply_to_message` if someone replies to a very first message in a directly created supergroup.
- **channel_chat_created** (`bool`, optional) – Service message: The channel has been created. This field can't be received in a message coming through updates, because bot can't be a member of a channel when it is created. It can only be found in `attr:reply_to_message` if someone replies to a very first message in a channel.
- **migrate_to_chat_id** (`int`, optional) – The group has been migrated to a supergroup with the specified identifier. This number may be greater than 32 bits and some programming languages may have difficulty/silent defects in interpreting it. But it is smaller than 52 bits, so a signed 64 bit integer or double-precision float type are safe for storing this identifier.
- **migrate_from_chat_id** (`int`, optional) – The supergroup has been migrated from a group with the specified identifier. This number may be greater than 32 bits and some programming languages may have difficulty/silent defects in interpreting it. But it is smaller than 52 bits, so a signed 64 bit integer or double-precision float type are safe for storing this identifier.
- **pinned_message** (`telegram.message`, optional) – Specified message was pinned. Note that the Message object in this field will not contain further `attr:reply_to_message` fields even if it is itself a reply.
- **invoice** (`telegram.Invoice`, optional) – Message is an invoice for a payment, information about the invoice.
- **successful_payment** (`telegram.SuccessfulPayment`, optional) – Message is a service message about a successful payment, information about the payment.
- **connected_website** (`str`, optional) – The domain name of the website on which the user has logged in.
- **forward_signature** (`str`, optional) – Signature of the post author for messages forwarded from channels.
- **author_signature** (`str`, optional) – Signature of the post author for messages in channels.
- **passport_data** (`telegram.PassportData`, optional) – Telegram Passport data

ATTACHMENT_TYPES = ['audio', 'game', 'animation', 'document', 'photo', 'sticker', 'video', 'voice', 'video_note']

MESSAGE_TYPES = ['text', 'new_chat_members', 'left_chat_member', 'new_chat_title', 'new_chat_members', 'left_chat_member', 'new_chat_title']

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.

Returns Message caption with caption entities formatted as HTML.

Return type `str`

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.

Returns Message caption with caption entities formatted as HTML.

Return type `str`

caption_markdown

Creates an Markdown-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 Markdown in the same way the original message was formatted.

Returns Message caption with caption entities formatted as Markdown.

Return type `str`

caption_markdown_urled

Creates an Markdown-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 Markdown. This also formats `telegram.MessageEntity.URL` as a hyperlink.

Returns Message caption with caption entities formatted as Markdown.

Return type `str`

chat_id

Shortcut for `telegram.Chat.id` for `chat`.

Type `int`

classmethod de_json (*data*, *bot*)

delete (**args*, ***kwargs*)

Shortcut for:

```
bot.delete_message(chat_id=message.chat_id,
                   message_id=message.message_id,
                   *args,
                   **kwargs)
```

Returns On success, `True` is returned.

Return type `bool`

edit_caption (**args*, ***kwargs*)

Shortcut for:

```
bot.edit_message_caption(chat_id=message.chat_id,
                        message_id=message.message_id,
                        *args,
                        **kwargs)
```

Note: You can only edit messages that the bot sent itself, therefore this method can only be used on the return value of the `bot.send_*` family of methods.

Returns On success, instance representing the edited message.

Return type `telegram.Message`

edit_media (*media*, **args*, ***kwargs*)

Shortcut for:

```
bot.edit_message_media(chat_id=message.chat_id,
                      message_id=message.message_id,
                      *args,
                      **kwargs)
```

Note: You can only edit messages that the bot sent itself, therefore this method can only be used on the return value of the `bot.send_*` family of methods.

Returns On success, instance representing the edited message.

Return type `telegram.Message`

edit_reply_markup (*args, **kwargs)

Shortcut for:

```
bot.edit_message_reply_markup(chat_id=message.chat_id,
                             message_id=message.message_id,
                             *args,
                             **kwargs)
```

Note: You can only edit messages that the bot sent itself, therefore this method can only be used on the return value of the `bot.send_*` family of methods.

Returns On success, instance representing the edited message.

Return type `telegram.Message`

edit_text (*args, **kwargs)

Shortcut for:

```
bot.edit_message_text(chat_id=message.chat_id,
                     message_id=message.message_id,
                     *args,
                     **kwargs)
```

Note: You can only edit messages that the bot sent itself, therefore this method can only be used on the return value of the `bot.send_*` family of methods.

Returns On success, instance representing the edited message.

Return type `telegram.Message`

effective_attachment

`telegram.Audio` or `telegram.Contact` or `telegram.Document` or `telegram.Animation` or `telegram.Game` or `telegram.Invoice` or `telegram.Location` or `List[telegram.PhotoSize]` or `telegram.Sticker` or `telegram.SuccessfulPayment` or `telegram.Venue` or `telegram.Video` or `telegram.VideoNote` or `telegram.Voice`: The attachment that this message was sent with. May be None if no attachment was sent.

forward (chat_id, disable_notification=False)

Shortcut for:

```
bot.forward_message(chat_id=chat_id,
                   from_chat_id=update.message.chat_id,
                   disable_notification=disable_notification,
                   message_id=update.message.message_id)
```

Returns On success, instance representing the message forwarded.

Return type `telegram.Message`

link

Convenience property. If the chat of the message is a supergroup or a channel and has a `Chat.username`, returns a t.me link of the message.

Type `str`

parse_caption_entities (*types=None*)

Returns a dict that maps `telegram.MessageEntity` to `str`. It contains entities from this message's caption filtered by their `telegram.MessageEntity.type` attribute as the key, and the text that each entity belongs to as the value of the dict.

Note: This method should always be used instead of the `caption_entities` attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See `parse_entity` for more info.

Parameters **types** (List[`str`], optional) – List of `telegram.MessageEntity` types as strings. If the `type` attribute of an entity is contained in this list, it will be returned. Defaults to a list of all types. All types can be found as constants in `telegram.MessageEntity`.

Returns A dictionary of entities mapped to the text that belongs to them, calculated based on UTF-16 codepoints.

Return type Dict[`telegram.MessageEntity`, `str`]

parse_caption_entity (*entity*)

Returns the text from a given `telegram.MessageEntity`.

Note: This method is present because Telegram calculates the offset and length in UTF-16 codepoint pairs, which some versions of Python don't handle automatically. (That is, you can't just slice `Message.caption` with the offset and length.)

Parameters

- **entity** (`telegram.MessageEntity`) – The entity to extract the text from. It must
- **an entity that belongs to this message.** (*be*) –

Returns The text of the given entity

Return type `str`

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 code-point 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
- **an entity that belongs to this message.** (*be*) –

Returns The text of the given entity

Return type str

reply_animation (*args, **kwargs)

Shortcut for:

```
bot.send_animation(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

reply_audio (*args, **kwargs)

Shortcut for:

```
bot.send_audio(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

reply_contact (*args, **kwargs)

Shortcut for:

```
bot.send_contact(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

reply_document (*args, **kwargs)

Shortcut for:

```
bot.send_document(update.message.chat_id, *args, **kwargs)
```


Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

reply_html (*args, **kwargs)

Shortcut for:

```
bot.send_message(update.message.chat_id, parse_mode=ParseMode.HTML, *args,
↪ **kwargs)
```

Sends a message with HTML formatting.

Keyword Arguments **quote** (bool, optional) – If set to True, the message is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

reply_location (*args, **kwargs)

Shortcut for:

```
bot.send_location(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

reply_markdown (*args, **kwargs)

Shortcut for:

```
bot.send_message(update.message.chat_id, parse_mode=ParseMode.MARKDOWN,
↪ *args,
**kwargs)
```

Sends a message with markdown formatting.

Keyword Arguments **quote** (bool, optional) – If set to True, the message is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

reply_media_group (*args, **kwargs)

Shortcut for:

```
bot.reply_media_group(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the media group is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns An array of the sent Messages.

Return type List[`telegram.Message`]

Raises `telegram.TelegramError`

reply_photo (*args, **kwargs)

Shortcut for:

```
bot.send_photo(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

reply_sticker (*args, **kwargs)

Shortcut for:

```
bot.send_sticker(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

reply_text (*args, **kwargs)

Shortcut for:

```
bot.send_message(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the message is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

reply_venue (*args, **kwargs)

Shortcut for:

```
bot.send_venue(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

reply_video (*args, **kwargs)

Shortcut for:

```
bot.send_video(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

reply_video_note (*args, **kwargs)

Shortcut for:

```
bot.send_video_note(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

reply_voice (*args, **kwargs)

Shortcut for:

```
bot.send_voice(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

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.

Returns Message text with entities formatted as HTML.

Return type `str`

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.

Returns Message text with entities formatted as HTML.

Return type `str`

text_markdown

Creates an Markdown-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 Markdown in the same way the original message was formatted.

Returns Message text with entities formatted as Markdown.

Return type `str`

text_markdown_urled

Creates an Markdown-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 Markdown. This also formats `telegram.MessageEntity.URL` as a hyperlink.

Returns Message text with entities formatted as Markdown.

Return type `str`

to_dict ()

class `telegram.MessageEntity` (type, offset, length, url=None, user=None, **kwargs)

Bases: `telegram.base.TelegramObject`

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

type

Type of the entity.

Type `str`

offset

Offset in UTF-16 code units to the start of the entity.

Type `int`

length

Length of the entity in UTF-16 code units.

Type `int`

url

Optional. Url that will be opened after user taps on the text.

Type `str`

user

Optional. The mentioned user.

Type `telegram.User`

Parameters

- **type** (`str`) – Type of the entity. Can be mention (@username), hashtag, bot_command, url, email, bold (bold text), italic (italic text), code (monowidth string), pre (monowidth block), text_link (for clickable text URLs), text_mention (for users without usernames).
- **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 usertaps on the text.
- **user** (`telegram.User`, optional) – For “text_mention” only, the mentioned user.

ALL_TYPES = ['mention', 'hashtag', 'cashtag', 'phone_number', 'bot_command', 'url',
List of all the types.

Type `List[str]`

BOLD = 'bold'

'bold'

Type `str`

BOT_COMMAND = 'bot_command'

'bot_command'

Type `str`

CASHTAG = 'cashtag'

'cashtag'

Type `str`

CODE = 'code'

'code'

Type `str`

EMAIL = 'email'

'email'

Type `str`

HASHTAG = 'hashtag'

'hashtag'

Type `str`

```
ITALIC = 'italic'
    'italic'

    Type str

MENTION = 'mention'
    'mention'

    Type str

PHONE_NUMBER = 'phone_number'
    'phone_number'

    Type str

PRE = 'pre'
    'pre'

    Type str

TEXT_LINK = 'text_link'
    'text_link'

    Type str

TEXT_MENTION = 'text_mention'
    'text_mention'

    Type str

URL = 'url'
    'url'

    Type str

classmethod de_json(data, bot)

classmethod de_list(data, bot)

class telegram.ParseMode
    Bases: object

    This object represents a Telegram Message Parse Modes.

    HTML = 'HTML'
        'HTML'

        Type str

    MARKDOWN = 'Markdown'
        'Markdown'

        Type str

class telegram.PhotoSize(file_id, width, height, file_size=None, bot=None, **kwargs)
    Bases: telegram.base.TelegramObject

    This object represents one size of a photo or a file/sticker thumbnail.

    file_id
        Unique identifier for this file.

        Type str

    width
        Photo width.

        Type int

    height
        Photo height.

        Type int
```

file_size

Optional. File size.

Type `int`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

Parameters

- **file_id** (`str`) – Unique identifier for this file.
- **width** (`int`) – Photo width.
- **height** (`int`) – Photo height.
- **file_size** (`int`, optional) – File size.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

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

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

get_file (*timeout=None*, ***kwargs*)

Convenience wrapper over `telegram.Bot.get_file`

Parameters

- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns `telegram.File`

Raises `telegram.TelegramError`

class `telegram.ReplyKeyboardRemove` (*selective=False*, ***kwargs*)

Bases: `telegram.replymarkup.ReplyMarkup`

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`).

remove_keyboard

Requests clients to remove the custom keyboard.

Type `True`

selective

Optional. Use this parameter if you want to remove the keyboard for specific users only.

Type `bool`

Example

A user votes in a poll, bot returns confirmation message in reply to the vote and removes the keyboard for that user, while still showing the keyboard with poll options to users who haven't voted yet.

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 Message object
- 2) if the bot's message is a reply (has `reply_to_message_id`), sender of the original message.

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

```
class telegram.ReplyKeyboardMarkup (keyboard, resize_keyboard=False,
                                     one_time_keyboard=False, selective=False,
                                     **kwargs)
```

Bases: `telegram.replymarkup.ReplyMarkup`

This object represents a custom keyboard with reply options.

keyboard

Array of button rows.

Type `List[List[telegram.KeyboardButton | str]]`

resize_keyboard

Optional. Requests clients to resize the keyboard.

Type `bool`

one_time_keyboard

Optional. Requests clients to hide the keyboard as soon as it's been used.

Type `bool`

selective

Optional. Show the keyboard to specific users only.

Type `bool`

Example

A user requests to change the bot's language, bot replies to the request with a keyboard to select the new language. Other users in the group don't see the keyboard.

Parameters

- **keyboard** (`List[List[str | telegram.KeyboardButton]]`) – Array of button rows, each represented by an Array of [telegram.KeyboardButton](#) objects.
- **resize_keyboard** (`bool`, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to `false`, in which case the custom keyboard is always of the same height as the app's standard keyboard. Defaults to `False`
- **one_time_keyboard** (`bool`, optional) – Requests clients to hide the keyboard as soon as it's been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to `False`.
- **selective** (`bool`, optional) – Use this parameter if you want to show the keyboard to specific users only. Targets:
 - 1) users that are @mentioned in the text of the Message object
 - 2) if the bot's message is a reply (has `reply_to_message_id`), sender of the original message.Defaults to `False`.
- ****kwargs** (dict) – Arbitrary keyword arguments.

```
classmethod from_button (button, resize_keyboard=False, one_time_keyboard=False, selective=False, **kwargs)
```

Shortcut for:

```
ReplyKeyboardMarkup([[button]], **kwargs)
```

Return an 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. Defaults to *False*
- **one_time_keyboard** (*bool*, optional) – Requests clients to hide the keyboard as soon as it's been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to *False*.
- **selective** (*bool*, optional) – Use this parameter if you want to show the keyboard to specific users only. Targets:
 - 1) users that are @mentioned in the text of the Message object
 - 2) **if the bot's message is a reply (has `reply_to_message_id`), sender of the original message.**Defaults to *False*.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

```
classmethod from_column(button_column, resize_keyboard=False,
                        one_time_keyboard=False, selective=False, **kwargs)
```

Shortcut for:

```
ReplyKeyboardMarkup([[button] for button in button_column], **kwargs)
```

Return an ReplyKeyboardMarkup from a single column of KeyboardButtons

Parameters

- **button_column** (*List*[*telegram.KeyboardButton* | *str*]) – The button to use in the markup
- **resize_keyboard** (*bool*, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to false, in which case the custom keyboard is always of the same height as the app's standard keyboard. Defaults to *False*
- **one_time_keyboard** (*bool*, optional) – Requests clients to hide the keyboard as soon as it's been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to *False*.
- **selective** (*bool*, optional) – Use this parameter if you want to show the keyboard to specific users only. Targets:
 - 1) users that are @mentioned in the text of the Message object
 - 2) **if the bot's message is a reply (has `reply_to_message_id`), sender of the original message.**Defaults to *False*.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

```
classmethod from_row(button_row, resize_keyboard=False, one_time_keyboard=False, selective=False, **kwargs)
```

Shortcut for:


```
ReplyKeyboardMarkup([button_row], **kwargs)
```

Return an `ReplyKeyboardMarkup` from a single row of `KeyboardButtons`

Parameters

- **button_row** (List[[`telegram.KeyboardButton`](#) | str]) – The button to use in the markup
- **resize_keyboard** (bool, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to false, in which case the custom keyboard is always of the same height as the app's standard keyboard. Defaults to `False`
- **one_time_keyboard** (bool, optional) – Requests clients to hide the keyboard as soon as it's been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to `False`.
- **selective** (bool, optional) – Use this parameter if you want to show the keyboard to specific users only. Targets:
 - 1) users that are @mentioned in the text of the `Message` object
 - 2) if the bot's message is a reply (has `reply_to_message_id`), sender of the original message.
 Defaults to `False`.
- ****kwargs** (dict) – Arbitrary keyword arguments.

`to_dict()`

class `telegram.ReplyMarkup`

Bases: `telegram.base.TelegramObject`

Base class for Telegram `ReplyMarkup` Objects.

See [`telegram.ReplyKeyboardMarkup`](#) and [`telegram.InlineKeyboardMarkup`](#) for detailed use.

class `telegram.Sticker` (*file_id*, *width*, *height*, *thumb=None*, *emoji=None*, *file_size=None*, *set_name=None*, *mask_position=None*, *bot=None*, ***kwargs*)

Bases: `telegram.base.TelegramObject`

This object represents a sticker.

file_id

Unique identifier for this file.

Type `str`

width

Sticker width.

Type `int`

height

Sticker height.

Type `int`

thumb

Optional. Sticker thumbnail in the .webp or .jpg format.

Type [`telegram.PhotoSize`](#)

emoji

Optional. Emoji associated with the sticker.

Type `str`

set_name

Optional. Name of the sticker set to which the sticker belongs.

Type `str`

mask_position

Optional. For mask stickers, the position where the mask should be placed.

Type `telegram.MaskPosition`

file_size

Optional. File size.

Type `int`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

Parameters

- **file_id** (`str`) – Unique identifier for this file.
- **width** (`int`) – Sticker width.
- **height** (`int`) – Sticker height.
- **thumb** (`telegram.PhotoSize`, optional) – Sticker thumbnail in the .webp or .jpg format.
- **emoji** (`str`, optional) – Emoji associated with the sticker
- **set_name** (`str`, optional) – Name of the sticker set to which the sticker belongs.
- **mask_position** (`telegram.MaskPosition`, optional) – For mask stickers, the position where the mask should be placed.
- **file_size** (`int`, optional) – File size.
- **(obj (**kwargs) – dict)**: Arbitrary keyword arguments.⁷
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.

classmethod de_json (`data`, `bot`)

classmethod de_list (`data`, `bot`)

get_file (`timeout=None`, ****kwargs**)

Convenience wrapper over `telegram.Bot.get_file`

Parameters

- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns `telegram.File`

Raises `telegram.TelegramError`

exception telegram.TelegramError (`message`)

Bases: `Exception`

class telegram.TelegramObject

Bases: `object`

Base class for most telegram objects.

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

```
to_dict ()
```

```
to_json ()
```

Returns str

```
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, **kwargs)
```

Bases: telegram.base.TelegramObject

This object represents an incoming update.

Note: At most one of the optional parameters can be present in any given update.

update_id

The update's unique identifier.

Type int

message

Optional. New incoming message.

Type *telegram.Message*

edited_message

Optional. New version of a message.

Type *telegram.Message*

channel_post

Optional. New incoming channel post.

Type *telegram.Message*

edited_channel_post

Optional. New version of a channel post.

Type *telegram.Message*

inline_query

Optional. New incoming inline query.

Type *telegram.InlineQuery*

chosen_inline_result

Optional. The result of an inline query that was chosen by a user.

Type *telegram.ChosenInlineResult*

callback_query

Optional. New incoming callback query.

Type *telegram.CallbackQuery*

shipping_query

Optional. New incoming shipping query.

Type *telegram.ShippingQuery*

pre_checkout_query

Optional. New incoming pre-checkout query.

Type *telegram.PreCheckoutQuery*

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.
- **message** (*telegram.Message*, optional) – New incoming message of any kind - text, photo, sticker, etc.
- **edited_message** (*telegram.Message*, optional) – New version of a message that is known to the bot and was edited.
- **channel_post** (*telegram.Message*, optional) – New incoming channel post of any kind - text, photo, sticker, etc.
- **edited_channel_post** (*telegram.Message*, optional) – New version of a channel post that is known to the bot and was edited.
- **inline_query** (*telegram.InlineQuery*, optional) – New incoming inline query.
- **chosen_inline_result** (*telegram.ChosenInlineResult*, optional) – The result of an inline query that was chosen by a user and sent to their chat partner.
- **callback_query** (*telegram.CallbackQuery*, optional) – New incoming callback query.
- **shipping_query** (*telegram.ShippingQuery*, optional) – New incoming shipping query. Only for invoices with flexible price.
- **pre_checkout_query** (*telegram.PreCheckoutQuery*, optional) – New incoming pre-checkout query. Contains full information about checkout
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

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

effective_chat

The chat that this update was sent in, no matter what kind of update this is. Will be *None* for *inline_query*, *chosen_inline_result*, *callback_query* from inline messages, *shipping_query* and *pre_checkout_query*.

Type *telegram.Chat*

effective_message

The message included in this update, no matter what kind of update this is. Will be *None* for *inline_query*, *chosen_inline_result*, *callback_query* from inline messages, *shipping_query* and *pre_checkout_query*.

Type *telegram.Message*

effective_user

The user that sent this update, no matter what kind of update this is. Will be *None* for *channel_post*.

Type *telegram.User*

```
class telegram.User (id, first_name, is_bot, last_name=None, username=None, language_code=None, bot=None, **kwargs)  
Bases: telegram.base.TelegramObject
```

This object represents a Telegram user or bot.

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`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

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.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.

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

classmethod `de_list` (`data`, `bot`)

full_name

Convenience property. The user's `first_name`, followed by (if available) `last_name`.

Type `str`

get_profile_photos (`*args`, `**kwargs`)

Shortcut for:

```
bot.get_user_profile_photos(update.message.from_user.id, *args, **kwargs)
```

link

Convenience property. If `username` is available, returns a t.me link of the user.

Type `str`

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`)

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.

Return type `str`

name

Convenience property. If available, returns the user's `username` prefixed with "@". If `username` is not available, returns `full_name`.

Type `str`

send_animation (**args*, ***kwargs*)

Shortcut for:

```
bot.send_animation(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

send_audio (**args*, ***kwargs*)

Shortcut for:

```
bot.send_audio(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

send_document (**args*, ***kwargs*)

Shortcut for:

```
bot.send_document(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

send_message (**args*, ***kwargs*)

Shortcut for:

```
bot.send_message(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

send_photo (**args*, ***kwargs*)

Shortcut for:

```
bot.send_photo(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

send_sticker (**args*, ***kwargs*)

Shortcut for:

```
bot.send_sticker(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_video (*args, **kwargs)

Shortcut for:

```
bot.send_video(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_video_note (*args, **kwargs)

Shortcut for:

```
bot.send_video_note(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

send_voice (*args, **kwargs)

Shortcut for:

```
bot.send_voice(User.id, *args, **kwargs)
```

Where User is the current instance.

Returns On success, instance representing the message posted.

Return type *telegram.Message*

class *telegram.UserProfilePhotos* (total_count, photos, **kwargs)

Bases: *telegram.base.TelegramObject*

This object represent a user's profile pictures.

total_count

Total number of profile pictures.

Type *int*

photos

Requested profile pictures.

Type *List[List[telegram.PhotoSize]]*

Parameters

- **total_count** (*int*) – Total number of profile pictures the target user has.
- **photos** (*List[List[telegram.PhotoSize]]*) – Requested profile pictures (in up to 4 sizes each).

classmethod *de_json* (data, bot)

to_dict ()

```
class telegram.Venue(location, title, address, foursquare_id=None, foursquare_type=None,
                    **kwargs)
    Bases: telegram.base.TelegramObject

    This object represents a venue.

    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
```

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

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

```
class telegram.Video(file_id, width, height, duration, thumb=None, mime_type=None,
                    file_size=None, bot=None, **kwargs)
    Bases: telegram.base.TelegramObject
```

This object represents a video file.

```
file_id
    Unique identifier for this file.

    Type str

width
    Video width as defined by sender.

    Type int

height
    Video height as defined by sender.

    Type int
```

```
duration
    Duration of the video in seconds as defined by sender.
```


Type `int`

thumb

Optional. Video thumbnail.

Type `telegram.PhotoSize`

mime_type

Optional. Mime type of a file as defined by sender.

Type `str`

file_size

Optional. File size.

Type `int`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

Parameters

- **file_id** (`str`) – Unique identifier for this file.
- **width** (`int`) – Video width as defined by sender.
- **height** (`int`) – Video height as defined by sender.
- **duration** (`int`) – Duration of the video in seconds as defined by sender.
- **thumb** (`telegram.PhotoSize`, optional) – Video thumbnail.
- **mime_type** (`str`, optional) – Mime type of a file as defined by sender.
- **file_size** (`int`, optional) – File size.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

classmethod `de_json(data, bot)`

get_file (`timeout=None`, ****kwargs**)

Convenience wrapper over `telegram.Bot.get_file`

Parameters

- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns `telegram.File`

Raises `telegram.TelegramError`

class `telegram.Voice` (`file_id`, `duration`, `mime_type=None`, `file_size=None`, `bot=None`, ****kwargs**)

Bases: `telegram.base.TelegramObject`

This object represents a voice note.

file_id

Unique identifier for this 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.

Type `int`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

Parameters

- **file_id** (`str`) – Unique identifier for this file.
- **duration** (`int`, optional) – Duration of the audio in seconds as defined by sender.
- **mime_type** (`str`, optional) – MIME type of the file as defined by sender.
- **file_size** (`int`, optional) – File size.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

classmethod `de_json(data, bot)`

get_file (`timeout=None`, ****kwargs**)

Convenience wrapper over `telegram.Bot.get_file`

Parameters

- **timeout** (`int` | `float`, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

Returns `telegram.File`

Raises `telegram.TelegramError`

```
class telegram.WebhookInfo(url, has_custom_certificate, pending_update_count,
                           last_error_date=None, last_error_message=None,
                           max_connections=None, allowed_updates=None, **kwargs)
```

Bases: `telegram.base.TelegramObject`

This object represents a Telegram WebhookInfo.

Contains information about the current status of a webhook.

url

Webhook URL.

Type `str`

has_custom_certificate

If a custom certificate was provided for webhook.

Type `bool`

pending_update_count

Number of updates awaiting delivery.

Type `int`

last_error_date

Optional. Unix time for the most recent error that happened.

Type `int`

last_error_message

Optional. Error message in human-readable format.

Type `str`

max_connections

Optional. Maximum allowed number of simultaneous HTTPS connections.

Type `int`

allowed_updates

Optional. A list of update types the bot is subscribed to.

Type `List[str]`

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.
- **last_error_date** (`int`, optional) – Unix time for the most recent error that happened when trying to deliver an update via webhook.
- **last_error_message** (`str`, optional) – Error message in human-readable format for the most recent error that happened when trying to deliver an update via webhook.
- **max_connections** (`int`, optional) – Maximum allowed number of simultaneous HTTPS connections to the webhook for update delivery.
- **allowed_updates** (`List[str]`, optional) – A list of update types the bot is subscribed to. Defaults to all update types.

classmethod `de_json(data, bot)`

class `telegram.Animation(file_id, width, height, duration, thumb=None, file_name=None, mime_type=None, file_size=None, **kwargs)`

Bases: `telegram.base.TelegramObject`

This object represents an animation file to be displayed in the message containing a game.

file_id

Unique file identifier.

Type `str`

width

Video width as defined by sender.

Type `int`

height

Video height as defined by sender.

Type `int`

duration

Duration of the video in seconds as defined by sender.

Type `int`

thumb

Optional. Animation thumbnail as defined by sender.

Type `telegram.PhotoSize`

file_name

Optional. Original animation filename as defined by sender.

Type `str`

mime_type

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

Type `str`

file_size

Optional. File size.

Type `int`

Parameters

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

classmethod `de_json(data, bot)`

class `telegram.Game` (*title, description, photo, text=None, text_entities=None, animation=None, **kwargs*)

Bases: `telegram.base.TelegramObject`

This object represents a game. Use BotFather to create and edit games, their short names will act as unique identifiers.

title

Title of the game.

Type `str`

description

Description of the game.

Type `str`

photo

Photo that will be displayed in the game message in chats.

Type `List[telegram.PhotoSize]`

text

Optional. Brief description of the game or high scores included in the game message. Can be automatically edited to include current high scores for the game when the bot calls `set_game_score`, or manually edited using `edit_message_text`.

Type `str`

text_entities

Optional. Special entities that appear in text, such as usernames, URLs, bot commands, etc.

Type `List[telegram.MessageEntity]`

animation

Optional. Animation that will be displayed in the game message in chats. Upload via BotFather.

Type `telegram.Animation`

Parameters

- **title** (`str`) – Title of the game.
- **description** (`str`) – Description of the game.
- **photo** (`List[telegram.PhotoSize]`) – Photo that will be displayed in the game message in chats.
- **text** (`str`, optional) – Brief description of the game or high scores included in the game message. Can be automatically edited to include current high scores for the game when the bot calls `set_game_score`, or manually edited using `edit_message_text`. 0-4096 characters. Also found as `telegram.constants.MAX_MESSAGE_LENGTH`.
- **text_entities** (`List[telegram.MessageEntity]`, optional) – Special entities that appear in text, such as usernames, URLs, bot commands, etc.
- **animation** (`telegram.Animation`, optional) – Animation that will be displayed in the game message in chats. Upload via BotFather.

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

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 `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 code-point 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`

to_dict ()

class `telegram.GameHighScore` (`position`, `user`, `score`)

Bases: `telegram.base.TelegramObject`

This object represents one row of the high scores table for a game.

position

Position in high score table for the game.

Type `int`

user

User.

Type `telegram.User`

score

Score.

Type `int`

Parameters

- **position** (`int`) – Position in high score table for the game.
- **user** (`telegram.User`) – User.
- **score** (`int`) – Score.

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

class `telegram.VideoNote` (`file_id`, `length`, `duration`, `thumb=None`, `file_size=None`, `bot=None`,
 **`kwargs`)

Bases: `telegram.base.TelegramObject`

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

file_id

Unique identifier for this file.

Type `str`

length

Video width and height as defined by sender.

Type `int`

duration

Duration of the video in seconds as defined by sender.

Type `int`

thumb

Optional. Video thumbnail.

Type `telegram.PhotoSize`

file_size

Optional. File size.

Type `int`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

Parameters

- **file_id** (`str`) – Unique identifier for this file.
- **length** (`int`) – Video width and height as defined by sender.
- **duration** (`int`) – Duration of the video in seconds as defined by sender.
- **thumb** (`telegram.PhotoSize`, optional) – Video thumbnail.
- **file_size** (`int`, optional) – File size.

- **bot** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- ****kwargs** (dict) – Arbitrary keyword arguments.

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

get_file (*timeout=None*, ***kwargs*)

Convenience wrapper over *telegram.Bot.get_file*

Parameters

- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns *telegram.File*

Raises *telegram.TelegramError*

class *telegram.LabeledPrice* (*label*, *amount*, ***kwargs*)

Bases: *telegram.base.TelegramObject*

This object represents a portion of the price for goods or services.

label

Portion label.

Type *str*

amount

Price of the product in the smallest units of the currency.

Type *int*

Parameters

- **label** (*str*) – Portion label
- **amount** (*int*) – Price of the product in the smallest units of the currency (integer, not float/double). For example, for a price of US\$ 1.45 pass `amount = 145`. See the `exp` parameter in `currencies.json`, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- ****kwargs** (dict) – Arbitrary keyword arguments.

class *telegram.SuccessfulPayment* (*currency*, *total_amount*, *invoice_payload*, *telegram_payment_charge_id*, *provider_payment_charge_id*, *shipping_option_id=None*, *order_info=None*, ***kwargs*)

Bases: *telegram.base.TelegramObject*

This object contains basic information about a successful payment.

currency

Three-letter ISO 4217 currency code.

Type *str*

total_amount

Total price in the smallest units of the currency.

Type *int*

invoice_payload

Bot specified invoice payload.

Type *str*

shipping_option_id

Optional. Identifier of the shipping option chosen by the user.

Type `str`

order_info

Optional. Order info provided by the user.

Type `telegram.OrderInfo`

telegram_payment_charge_id

Telegram payment identifier.

Type `str`

provider_payment_charge_id

Provider payment identifier.

Type `str`

Parameters

- **currency** (`str`) – Three-letter ISO 4217 currency code.
- **total_amount** (`int`) – Total price in the smallest units of the currency (integer, not float/double). For example, for a price of US\$ 1.45 pass amount = 145. See the exp parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- **invoice_payload** (`str`) – Bot specified invoice payload.
- **shipping_option_id** (`str`, optional) – Identifier of the shipping option chosen by the user.
- **order_info** (`telegram.OrderInfo`, optional) – Order info provided by the user
- **telegram_payment_charge_id** (`str`) – Telegram payment identifier.
- **provider_payment_charge_id** (`str`) – Provider payment identifier.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

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

class `telegram.ShippingOption` (`id`, `title`, `prices`, ****kwargs**)

Bases: `telegram.base.TelegramObject`

This object represents one shipping option.

id

Shipping option identifier.

Type `str`

title

Option title.

Type `str`

prices

List of price portions.

Type `List[telegram.LabeledPrice]`

Parameters

- **id** (`str`) – Shipping option identifier.
- **title** (`str`) – Option title.
- **prices** (`List[telegram.LabeledPrice]`) – List of price portions.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

to_dict ()


```
class telegram.ShippingAddress(country_code, state, city, street_line1, street_line2,
                               post_code, **kwargs)
```

Bases: telegram.base.TelegramObject

This object represents a Telegram ShippingAddress.

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

Parameters

- **country_code** (str) – ISO 3166-1 alpha-2 country code.
- **state** (str) – State, if applicable.
- **city** (str) – City.
- **street_line1** (str) – First line for the address.
- **street_line2** (str) – Second line for the address.
- **post_code** (str) – Address post code.
- ****kwargs** (dict) – Arbitrary keyword arguments.

classmethod de_json(*data*, *bot*)

```
class telegram.PreCheckoutQuery(id, from_user, currency, total_amount, invoice_payload,
                                shipping_option_id=None, order_info=None, bot=None,
                                **kwargs)
```

Bases: telegram.base.TelegramObject

This object contains information about an incoming pre-checkout query.

Note:

- In Python *from* is a reserved word, use *from_user* instead.
-

id

Unique query identifier.

Type str

from_user

User who sent the query.

Type `telegram.User`

currency

Three-letter ISO 4217 currency code.

Type `str`

total_amount

Total price in the smallest units of the currency.

Type `int`

invoice_payload

Bot specified invoice payload.

Type `str`

shipping_option_id

Optional. Identifier of the shipping option chosen by the user.

Type `str`

order_info

Optional. Order info provided by the user.

Type `telegram.OrderInfo`

bot

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

Parameters

- **id** (`str`) – Unique query identifier.
- **from_user** (`telegram.User`) – User who sent the query.
- **currency** (`str`) – Three-letter ISO 4217 currency code
- **total_amount** (`int`) – Total price in the smallest units of the currency (integer, not float/double). For example, for a price of US\$ 1.45 pass amount = 145. See the exp parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- **invoice_payload** (`str`) – Bot specified invoice payload.
- **shipping_option_id** (`str`, optional) – Identifier of the shipping option chosen by the user.
- **order_info** (`telegram.OrderInfo`, optional) – Order info provided by the user.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

answer (**args*, ***kwargs*)

Shortcut for:

```
bot.answer_pre_checkout_query(update.pre_checkout_query.id, *args, ↳
↳ **kwargs)
```

Parameters

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

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

class telegram.**OrderInfo** (*name=None*, *phone_number=None*, *email=None*, *shipping_address=None*, ****kwargs**)

Bases: telegram.base.TelegramObject

This object represents information about an order.

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*

Parameters

- **name** (*str*, optional) – User name.
- **phone_number** (*str*, optional) – User’s phone number.
- **email** (*str*, optional) – User email.
- **shipping_address** (*telegram.ShippingAddress*, optional) – User shipping address.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

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

class telegram.**Invoice** (*title*, *description*, *start_parameter*, *currency*, *total_amount*, ****kwargs**)

Bases: telegram.base.TelegramObject

This object contains basic information about an invoice.

title

Product name.

Type *str*

description

Product description.

Type *str*

start_parameter

Unique bot deep-linking parameter.

Type *str*

- **shipping_address** (*telegram.ShippingAddress*) – User specified shipping address.
- **bot** (*telegram.Bot*, optional) – The Bot to use for instance methods.
- ****kwargs** (dict) – Arbitrary keyword arguments.

answer (*args, **kwargs)

Shortcut for:

```
bot.answer_shipping_query(update.shipping_query.id, *args, **kwargs)
```

Parameters

- **ok** (bool) – Specify True if delivery to the specified address is possible and False if there are any problems (for example, if delivery to the specified address is not possible).
- **shipping_options** (List[*telegram.ShippingOption*], optional) – Required if ok is True. A JSON-serialized array of available shipping options.
- **error_message** (str, optional) – Required if ok is False. Error message in human readable form that explains why it is impossible to complete the order (e.g. “Sorry, delivery to your desired address is unavailable”). Telegram will display this message to the user.

classmethod de_json (data, bot)

class telegram.**ChatPhoto** (small_file_id, big_file_id, bot=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents a chat photo.

small_file_id

Unique file identifier of small (160x160) chat photo.

Type str

big_file_id

Unique file identifier of big (640x640) chat photo.

Type str

Parameters

- **small_file_id** (str) – Unique file identifier of small (160x160) chat photo. This file_id can be used only for photo download.
- **big_file_id** (str) – Unique file identifier of big (640x640) chat photo. This file_id can be used only for photo download.
- **bot** (*telegram.Bot*, optional) – The Bot to use for instance methods
- ****kwargs** (dict) – Arbitrary keyword arguments.

classmethod de_json (data, bot)

class telegram.**StickerSet** (name, title, contains_masks, stickers, bot=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents a sticker set.

name

Sticker set name.

Type str

title

Sticker set title.

Type `str`

contains_masks

True, if the sticker set contains masks.

Type `bool`

stickers

List of all set stickers.

Type `List[telegram.Sticker]`

Parameters

- **name** (`str`) – Sticker set name.
- **title** (`str`) – Sticker set title.
- **contains_masks** (`bool`) – True, if the sticker set contains masks.
- **stickers** (`List[telegram.Sticker]`) – List of all set stickers.

static `de_json` (`data`, `bot`)

`to_dict` ()

class `telegram.MaskPosition` (`point`, `x_shift`, `y_shift`, `scale`, `**kwargs`)

Bases: `telegram.base.TelegramObject`

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

point

The part of the face relative to which the mask should be placed.

Type `str`

x_shift

Shift by X-axis measured in widths of the mask scaled to the face size, from left to right.

Type `float`

y_shift

Shift by Y-axis measured in heights of the mask scaled to the face size, from top to bottom.

Type `float`

scale

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

Type `float`

Notes

`type` should be one of the following: *forehead*, *eyes*, *mouth* or *chin*. You can use the class constants for those.

Parameters

- **point** (`str`) – The part of the face relative to which the mask should be placed.
- **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.

```
CHIN = 'chin'
    'chin'

    Type str
```

```
EYES = 'eyes'
    'eyes'

    Type str
```

```
FOREHEAD = 'forehead'
    'forehead'

    Type str
```

```
MOUTH = 'mouth'
    'mouth'

    Type str
```

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

```
class telegram.CallbackGame
    Bases: telegram.base.TelegramObject
```

A placeholder, currently holds no information. Use BotFather to set up your game.

```
class telegram.InputMedia
    Bases: telegram.base.TelegramObject

    Base class for Telegram InputMedia Objects.
```

See [telegram.InputMediaAnimation](#), [telegram.InputMediaAudio](#), [telegram.InputMediaDocument](#), [telegram.InputMediaPhoto](#) and [telegram.InputMediaVideo](#) for detailed use.

```
class telegram.InputMediaPhoto(media, caption=None, parse_mode=None)
    Bases: telegram.files.inputmedia.InputMedia
```

Represents a photo to be sent.

```
type
    photo.

    Type str
```

```
media
    File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an
    HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing telegram.
    PhotoSize object to send.
```

```
    Type str
```

```
caption
    Optional. Caption of the photo to be sent, 0-1024 characters.
```

```
    Type str
```

```
parse_mode
    Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text
    or inline URLs in the media caption. See the constants in telegram.ParseMode for the available
    modes.
```

```
    Type str
```

Parameters

- **media** (str) – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing [telegram.PhotoSize](#) object to send.

- **caption** (`str`, optional) – Caption of the photo to be sent, 0-1024 characters.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in [telegram.ParseMode](#) for the available modes.

class telegram.**InputMediaVideo** (*media*, *caption=None*, *width=None*, *height=None*, *duration=None*, *supports_streaming=None*, *parse_mode=None*, *thumb=None*)

Bases: telegram.files.inputmedia.InputMedia

Represents a video to be sent.

type

video.

Type `str`

media

File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing [telegram.Video](#) object to send.

Type `str`

caption

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

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in [telegram.ParseMode](#) for the available modes.

Type `str`

width

Optional. Video width.

Type `int`

height

Optional. Video height.

Type `int`

duration

Optional. Video duration.

Type `int`

supports_streaming

Optional. Pass True, if the uploaded video is suitable for streaming.

Type `bool`

thumb

Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or `file_id`.

Type *filelike object*

Parameters

- **media** (`str`) – File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing [telegram.Video](#) object to send.

- **caption** (*str*, optional) – Caption of the video to be sent, 0-1024 characters.
- **parse_mode** (*str*, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in [telegram.ParseMode](#) for the available modes.
- **width** (*int*, optional) – Video width.
- **height** (*int*, optional) – Video height.
- **duration** (*int*, optional) – Video duration.
- **supports_streaming** (*bool*, optional) – Pass True, if the uploaded video is suitable for streaming.
- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not passed as a string or `file_id`.

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.

class telegram.**PassportElementError** (*source, type, message, **kwargs*)

Bases: telegram.base.TelegramObject

Baseclass for the PassportElementError* classes.

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*

Parameters

- **source** (*str*) – Error source.
- **type** (*str*) – The section of the user's Telegram Passport which has the error.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

class telegram.**PassportElementErrorFile** (*type, file_hash, message, **kwargs*)

Bases: telegram.passport.passportelementerrors.PassportElementError

Represents an issue with a document scan. The error is considered resolved when the file with the document scan changes.

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`

Parameters

- **type** (`str`) – The section of the user’s Telegram Passport which has the issue, one of “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.
- **file_hash** (`str`) – Base64-encoded file hash.
- **message** (`str`) – Error message.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

class `telegram.PassportElementErrorReverseSide` (*type, file_hash, message, **kwargs*)

Bases: `telegram.passport.passportelementerrors.PassportElementError`

Represents an issue with the front side of a document. The error is considered resolved when the file with the reverse side of the document changes.

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 reverse side of the document.

Type `str`

message

Error message.

Type `str`

Parameters

- **type** (`str`) – The section of the user’s Telegram Passport which has the issue, one of “driver_license”, “identity_card”.
- **file_hash** (`str`) – Base64-encoded hash of the file with the reverse side of the document.
- **message** (`str`) – Error message.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

class `telegram.PassportElementErrorFrontSide` (*type, file_hash, message, **kwargs*)

Bases: `telegram.passport.passportelementerrors.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.

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`

Parameters

- **type** (`str`) – The section of the user’s Telegram Passport which has the issue, one of “passport”, “driver_license”, “identity_card”, “internal_passport”.
- **file_hash** (`str`) – Base64-encoded hash of the file with the front side of the document.
- **message** (`str`) – Error message.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.PassportElementErrorFiles (type, file_hashes, message, **kwargs)
    Bases: telegram.passport.passportelementerrors.PassportElementError
```

Represents an issue with a list of scans. The error is considered resolved when the file with the document scan changes.

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`

Parameters

- **type** (`str`) – The section of the user’s Telegram Passport which has the issue, one of “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.
- **file_hashes** (`List[str]`) – List of base64-encoded file hashes.
- **message** (`str`) – Error message.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

```
class telegram.PassportElementErrorDataField (type, field_name, data_hash, message,
                                              **kwargs)
```

Bases: telegram.passport.passportelementerrors.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.

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`

Parameters

- **type** (`str`) – The section of the user’s Telegram Passport which has the error, one of “personal_details”, “passport”, “driver_license”, “identity_card”, “internal_passport”, “address”.
- **field_name** (`str`) – Name of the data field which has the error.
- **data_hash** (`str`) – Base64-encoded data hash.
- **message** (`str`) – Error message.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

class `telegram.PassportElementErrorFile` (*type, file_hash, message, **kwargs*)

Bases: `telegram.passport.passportelementerrors.PassportElementError`

Represents an issue with a document scan. The error is considered resolved when the file with the document scan changes.

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`

Parameters

- **type** (`str`) – The section of the user’s Telegram Passport which has the issue, one of “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.
- **file_hash** (`str`) – Base64-encoded file hash.
- **message** (`str`) – Error message.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

class `telegram.Credentials` (*secure_data, nonce, bot=None, **kwargs*)

Bases: `telegram.base.TelegramObject`

secure_data

Credentials for encrypted data

Type `telegram.SecureData`

nonce

Bot-specified nonce

Type `str`

classmethod `de_json` (*data, bot*)

```
class telegram.DataCredentials (data_hash, secret, **kwargs)  
    Bases: telegram.passport.credentials._CredentialsBase
```

These credentials can be used to decrypt encrypted data from the data field in EncryptedPassportData.

Parameters

- **data_hash** (*str*) – Checksum of encrypted data
- **secret** (*str*) – Secret of encrypted data

hash
Checksum of encrypted data

Type *str*

secret
Secret of encrypted data

Type *str*

to_dict ()

```
class telegram.SecureData (personal_details=None, passport=None, internal_passport=None,  
                           driver_license=None, identity_card=None, address=None, util-  
                           ity_bill=None, bank_statement=None, rental_agreement=None,  
                           passport_registration=None, temporary_registration=None,  
                           bot=None, **kwargs)
```

Bases: telegram.base.TelegramObject

This object represents the credentials that were used to decrypt the encrypted data. All fields are optional and depend on fields that were requested.

personal_details
Credentials for encrypted personal details.
Type telegram.SecureValue, optional

passport
Credentials for encrypted passport.
Type telegram.SecureValue, optional

internal_passport
Credentials for encrypted internal passport.
Type telegram.SecureValue, optional

driver_license
Credentials for encrypted driver license.
Type telegram.SecureValue, optional

identity_card
Credentials for encrypted ID card
Type telegram.SecureValue, optional

address
Credentials for encrypted residential address.
Type telegram.SecureValue, optional

utility_bill
Credentials for encrypted utility bill.
Type telegram.SecureValue, optional

bank_statement
Credentials for encrypted bank statement.
Type telegram.SecureValue, optional

rental_agreement

Credentials for encrypted rental agreement.

Type telegram.SecureValue, optional

passport_registration

Credentials for encrypted registration from internal passport.

Type telegram.SecureValue, optional

temporary_registration

Credentials for encrypted temporary registration.

Type telegram.SecureValue, optional

classmethod de_json (*data, bot*)

class telegram.**FileCredentials** (*file_hash, secret, **kwargs*)

Bases: telegram.passport.credentials._CredentialsBase

These credentials can be used to decrypt encrypted files from the front_side, reverse_side, selfie and files fields in EncryptedPassportData.

Parameters

- **file_hash** (*str*) – Checksum of encrypted file
- **secret** (*str*) – Secret of encrypted file

hash

Checksum of encrypted file

Type str

secret

Secret of encrypted file

Type str

to_dict ()

class telegram.**IdDocumentData** (*document_no, expiry_date, bot=None, **kwargs*)

Bases: telegram.base.TelegramObject

This object represents the data of an identity document.

document_no

Document number.

Type str

expiry_date

Optional. Date of expiry, in DD.MM.YYYY format.

Type str

classmethod de_json (*data, bot*)

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, bot=None, **kwargs*)

Bases: telegram.base.TelegramObject

This object represents personal details.

first_name

First Name.

Type str

middle_name

Optional. First Name.

Type `str`

last_name

Last Name.

Type `str`

birth_date

Date of birth in DD.MM.YYYY format.

Type `str`

gender

Gender, male or female.

Type `str`

country_code

Citizenship (ISO 3166-1 alpha-2 country code).

Type `str`

residence_country_code

Country of residence (ISO 3166-1 alpha-2 country code).

Type `str`

first_name

First Name in the language of the user's country of residence.

Type `str`

middle_name

Optional. Middle Name in the language of the user's country of residence.

Type `str`

last_name

Last Name in the language of the user's country of residence.

Type `str`

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

class `telegram.ResidentialAddress` (*street_line1*, *street_line2*, *city*, *state*, *country_code*,
post_code, *bot=None*, ***kwargs*)

Bases: `telegram.base.TelegramObject`

This object represents a residential address.

street_line1

First line for the address.

Type `str`

street_line2

Optional. Second line for the address.

Type `str`

city

City.

Type `str`

state

Optional. State.

Type `str`

country_code
ISO 3166-1 alpha-2 country code.

Type `str`

post_code
Address post code.

Type `str`

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

class `telegram.InputMediaVideo` (*media*, *caption=None*, *width=None*, *height=None*, *duration=None*, *supports_streaming=None*, *parse_mode=None*, *thumb=None*)

Bases: `telegram.files.inputmedia.InputMedia`

Represents a video to be sent.

type
video.

Type `str`

media
File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.Video` object to send.

Type `str`

caption
Optional. Caption of the video to be sent, 0-1024 characters.

Type `str`

parse_mode
Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

Type `str`

width
Optional. Video width.

Type `int`

height
Optional. Video height.

Type `int`

duration
Optional. Video duration.

Type `int`

supports_streaming
Optional. Pass True, if the uploaded video is suitable for streaming.

Type `bool`

thumb
Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or `file_id`.

Type *filelike object*

Parameters

- **media** (*str*) – File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.Video` object to send.
- **caption** (*str*, optional) – Caption of the video to be sent, 0-1024 characters.
- **parse_mode** (*str*, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **width** (*int*, optional) – Video width.
- **height** (*int*, optional) – Video height.
- **duration** (*int*, optional) – Video duration.
- **supports_streaming** (*bool*, optional) – Pass True, if the uploaded video is suitable for streaming.
- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not passed as a string or `file_id`.

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.

```
class telegram.InputMediaAnimation (media,          thumb=None,          caption=None,
                                     parse_mode=None, width=None,    height=None,
                                     duration=None)
```

Bases: `telegram.files.inputmedia.InputMedia`

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

type

`animation.`

Type `str`

media

File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.Animation` object to send.

Type `str`

thumb

Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not passed as a string or `file_id`.

Type *filelike object*

caption

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

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

Type `str`

width

Optional. Animation width.

Type `int`

height

Optional. Animation height.

Type `int`

duration

Optional. Animation duration.

Type `int`

Parameters

- **media** (`str`) – File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.Animation` object to send.
- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not passed as a string or `file_id`.
- **caption** (`str`, optional) – Caption of the animation to be sent, 0-1024 characters.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **width** (`int`, optional) – Animation width.
- **height** (`int`, optional) – Animation height.
- **duration** (`int`, optional) – Animation duration.

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.

```
class telegram.InputMediaAudio(media, thumb=None, caption=None, parse_mode=None,  
                               duration=None, performer=None, title=None)
```

Bases: `telegram.files.inputmedia.InputMedia`

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

type

`audio`.

Type `str`

media

File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.Audio` object to send.

Type `str`

caption

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

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

Type `str`

duration

Duration of the audio in seconds.

Type `int`

performer

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

Type `str`

title

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

Type `str`

thumb

Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not passed as a string or `file_id`.

Type *filelike object*

Parameters

- **media** (`str`) – File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing *telegram.Document* object to send.
- **caption** (`str`, optional) – Caption of the audio to be sent, 0-1024 characters.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.
- **duration** (`int`) – Duration of the audio in seconds as defined by sender.
- **performer** (`str`, optional) – Performer of the audio as defined by sender or by audio tags.
- **title** (`str`, optional) – Title of the audio as defined by sender or by audio tags.
- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not passed as a string or `file_id`.

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.

```
class telegram.InputMediaDocument (media, thumb=None, caption=None,
                                     parse_mode=None)
```

Bases: `telegram.files.inputmedia.InputMedia`

Represents a general file to be sent.

type

`document`.

Type `str`

media

File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing *telegram.Document* object to send.

Type `str`

caption

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

Type `str`

parse_mode

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

Type `str`

thumb

Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or `file_id`.

Type *filelike object*

Parameters

- **media** (`str`) – File to send. Pass a `file_id` to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.Document` object to send.
- **caption** (`str`, optional) – Caption of the document to be sent, 0-1024 characters.
- **parse_mode** (`str`, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail's width and height should not exceed 90. Ignored if the file is not is passed as a string or `file_id`.

exception `telegram.TelegramDecryptionError` (*message*)

Bases: `telegram.error.TelegramError`

Something went wrong with decryption.

class `telegram.PassportElementErrorSelfie` (*type*, *file_hash*, *message*, ***kwargs*)

Bases: `telegram.passport.passportelementerrors.PassportElementError`

Represents an issue with the selfie with a document. The error is considered resolved when the file with the selfie changes.

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`

Parameters

- **type** (`str`) – The section of the user's Telegram Passport which has the issue, one of "passport", "driver_license", "identity_card", "internal_passport".
- **file_hash** (`str`) – Base64-encoded hash of the file with the selfie.

- **message** (str) – Error message.
- ****kwargs** (dict) – Arbitrary keyword arguments.

[illegible]

Bases: telegram.passport.passportelementerrors.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.

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

Parameters

- **type** (str) – Type of element of the user’s Telegram Passport which has the issue, one of “passport”, “driver_license”, “identity_card”, “internal_passport”, “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.
- **file_hash** (str) – Base64-encoded hash of the file.
- **message** (str) – Error message.
- ****kwargs** (dict) – Arbitrary keyword arguments.

[illegible]

Bases: telegram.passport.passportelementerrors.PassportElementError

Represents an issue with the translated version of a document. The error is considered resolved when a file with the document translation change.

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 file hash.

Type `str`

message

Error message.

Type `str`

Parameters

- **type** (str) – Type of element of the user’s Telegram Passport which has the issue, one of “passport”, “driver_license”, “identity_card”, “internal_passport”, “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”
- **file_hashes** (List[str]) – List of base64-encoded file hashes.
- **message** (str) – Error message.
- ****kwargs** (dict) – Arbitrary keyword arguments.

[illegible]

Bases: telegram.passport.passportelementerrors.PassportElementError

Represents an issue in an unspecified place. The error is considered resolved when new data is added.

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

Parameters

- **type** (str) – Type of element of the user’s Telegram Passport which has the issue.
- **element_hash** (str) – Base64-encoded element hash.
- **message** (str) – Error message.
- ****kwargs** (dict) – Arbitrary keyword arguments.

CHAPTER 2

Changelog

2.1 Changes

Released 2019-02-13

First beta release ever. It has been so long since last release that we would like to test the impact before a final release.

We do NOT recommend using this beta release in production.

Major 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

See the wiki page at <https://git.io/fxJuV> for a detailed guide on how to migrate from version 11 to version 12.

3.1 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.

3.2 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

3.3 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 “!”).

3.4 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!

3.5 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)

3.6 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)

3.7 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.

Pre-2019 (up and including to version 11.1.0)

2018-09-01 *Released 11.1.0*

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

2018-08-29 *Released 11.0.0*

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 forum_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)

2018-05-02 Released 10.1.0

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)

2018-04-17 Released 10.0.2

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.

2018-03-05 *Released 10.0.1*

Fixes:

- Fix conversationhandler timeout (PR #1032)
- Add missing docs utils (PR #912)

2018-03-02 *Released 10.0.0*

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)

2017-12-08 Released 9.0.0

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.

2017-10-15 Released 8.1.1

- Fix Commandhandler crashing on single character messages (PR #873).

2017-10-14 Released 8.1.0

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.

2017-09-01 Released 8.0.0

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.

2017-07-28 Released 7.0.1

- Fix TypeError exception in RegexHandler (PR #751).

- Small documentation fix (PR #749).

2017-07-25 *Released 7.0.0*

- 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).
- `ConvesationHandler` - check if a user exist before using it (PR #699).
- Removed deprecated `telegram.Emoji`.
- Removed deprecated `Botan` import from `utils` (`Botan` is still available through `contrib`).
- Removed deprecated `ReplyKeyboardHide`.
- Removed deprecated `edit_message` argument of `bot.set_game_score`.
- Internal restructure of files.
- Improved documentation.
- Improved unitests.

2017-06-18

Released 6.1.0

- Fully support Bot API 3.0
- Add more fine-grained filters for status updates
- Bug fixes and other improvements

2017-05-29

Released 6.0.3

- Faulty PyPI release

2017-05-29

Released 6.0.2

- Avoid confusion with user's `urllib3` by renaming vendored `urllib3` to `ptb_urllib3`

2017-05-19

Released 6.0.1

- Add support for `User.language_code`
- Fix `Message.text_html` and `Message.text_markdown` for messages with emoji

2017-05-19

Released 6.0.0

- Add support for Bot API 2.3.1
- Add support for `deleteMessage` API method
- New, simpler API for `JobQueue` - <https://github.com/python-telegram-bot/python-telegram-bot/pull/484>
- Download files into file-like objects - <https://github.com/python-telegram-bot/python-telegram-bot/pull/459>

- Use vendor `urllib3` to address issues with timeouts - The default timeout for messages is now 5 seconds. For sending media, the default timeout is now 20 seconds.
- String attributes that are not set are now `None` by default, instead of empty strings
- Add `text_markdown` and `text_html` properties to `Message` - <https://github.com/python-telegram-bot/python-telegram-bot/pull/507>
- Add support for Socks5 proxy - <https://github.com/python-telegram-bot/python-telegram-bot/pull/518>
- Add support for filters in `CommandHandler` - <https://github.com/python-telegram-bot/python-telegram-bot/pull/536>
- Add the ability to invert (not) filters - <https://github.com/python-telegram-bot/python-telegram-bot/pull/552>
- Add `Filters.group` and `Filters.private`
- Compatibility with GAE via `urllib3.contrib` package - <https://github.com/python-telegram-bot/python-telegram-bot/pull/583>
- Add equality rich comparison operators to telegram objects - <https://github.com/python-telegram-bot/python-telegram-bot/pull/604>
- Several bugfixes and other improvements
- Remove some deprecated code

2017-04-17

Released 5.3.1

- Hotfix release due to bug introduced by `urllib3` version 1.21

2016-12-11

Released 5.3

- Implement API changes of November 21st (Bot API 2.3)
- `JobQueue` now supports `datetime.timedelta` in addition to seconds
- `JobQueue` now supports running jobs only on certain days
- New `Filters.reply` filter
- Bugfix for `Message.edit_reply_markup`
- Other bugfixes

2016-10-25

Released 5.2

- Implement API changes of October 3rd (games update)
- Add `Message.edit_*` methods
- Filters for the `MessageHandler` can now be combined using bitwise operators (`&` and `|`)
- Add a way to save user- and chat-related data temporarily
- Other bugfixes and improvements

2016-09-24

Released 5.1

- Drop Python 2.6 support
- Deprecate `telegram.Emoji`
- Use `ujson` if available
- Add instance methods to `Message`, `Chat`, `User`, `InlineQuery` and `CallbackQuery`
- RegEx filtering for `CallbackQueryHandler` and `InlineQueryHandler`

- New MessageHandler filters: forwarded and entity
- Add Message.get_entity to correctly handle UTF-16 codepoints and MessageEntity offsets
- Fix bug in ConversationHandler when first handler ends the conversation
- Allow multiple Dispatcher instances
- Add ChatMigrated Exception
- Properly split and handle arguments in CommandHandler

2016-07-15

Released 5.0

- Rework JobQueue
- Introduce ConversationHandler
- Introduce telegram.constants - <https://github.com/python-telegram-bot/python-telegram-bot/pull/342>

2016-07-12

Released 4.3.4

- Fix proxy support with urllib3 when proxy requires auth

2016-07-08

Released 4.3.3

- Fix proxy support with urllib3

2016-07-04

Released 4.3.2

- Fix: Use timeout parameter in all API methods

2016-06-29

Released 4.3.1

- Update wrong requirement: urllib3>=1.10

2016-06-28

Released 4.3

- Use urllib3.PoolManager for connection re-use
- Rewrite run_async decorator to re-use threads
- New requirements: urllib3 and certifi

2016-06-10

Released 4.2.1

- Fix CallbackQuery.to_dict() bug (thanks to @jlmadurga)
- Fix editMessageText exception when receiving a CallbackQuery

2016-05-28

Released 4.2

- Implement Bot API 2.1
- Move botan module to telegram.contrib
- New exception type: BadRequest

2016-05-22

Released 4.1.2

- Fix `MessageEntity` decoding with Bot API 2.1 changes

2016-05-16

Released 4.1.1

- Fix deprecation warning in `Dispatcher`

2016-05-15

Released 4.1

- Implement API changes from May 6, 2016
- Fix bug when `start_polling` with `clean=True`
- Methods now have `snake_case` equivalent, for example `telegram.Bot.send_message` is the same as `telegram.Bot.sendMessage`

2016-05-01

Released 4.0.3

- Add missing attribute `location` to `InlineQuery`

2016-04-29

Released 4.0.2

- Bugfixes
- `KeyboardReplyMarkup` now accepts `str` again

2016-04-27

Released 4.0.1

- Implement Bot API 2.0
- Almost complete recode of `Dispatcher`
- Please read the [Transition Guide to 4.0](#)
- **Changes from 4.0rc1**
 - The syntax of filters for `MessageHandler` (upper/lower cases)
 - Handler groups are now identified by `int` only, and ordered
- **Note:** v4.0 has been skipped due to a PyPI accident

2016-04-22

Released 4.0rc1

- Implement Bot API 2.0
- Almost complete recode of `Dispatcher`
- Please read the [Transistion Guide to 4.0](#)

2016-03-22

Released 3.4

- Move `Updater`, `Dispatcher` and `JobQueue` to new `telegram.ext` submodule (thanks to @rahiel)
- Add `disable_notification` parameter (thanks to @aidarbiktimirov)
- Fix bug where commands sent by Telegram Web would not be recognized (thanks to @shelomentsevd)
- Add option to skip old updates on bot startup

- Send files from `BufferedReader`

2016-02-28

Released 3.3

- Inline bots
- Send any file by URL
- Specialized exceptions: `Unauthorized`, `InvalidToken`, `NetworkError` and `TimedOut`
- Integration for botan.io (thanks to @ollmer)
- HTML Parsemode (thanks to @jlmadurga)
- Bugfixes and under-the-hood improvements

Very special thanks to Noam Meltzer (@tsnoam) for all of his work!

2016-01-09

Released 3.3b1

- Implement inline bots (beta)

2016-01-05

Released 3.2.0

- Introducing `JobQueue` (original author: @franciscod)
- Streamlining all exceptions to `TelegramError` (Special thanks to @tsnoam)
- Proper locking of `Updater` and `Dispatcher` start and stop methods
- Small bugfixes

2015-12-29

Released 3.1.2

- Fix custom path for file downloads
- Don't stop the dispatcher thread on uncaught errors in handlers

2015-12-21

Released 3.1.1

- Fix a bug where asynchronous handlers could not have additional arguments
- Add `groups` and `groupdict` as additional arguments for regex-based handlers

2015-12-16

Released 3.1.0

- The `chat`-field in `Message` is now of type `Chat`. (API update Oct 8 2015)
- `Message` now contains the optional fields `supergroup_chat_created`, `migrate_to_chat_id`, `migrate_from_chat_id` and `channel_chat_created`. (API update Nov 2015)

2015-12-08

Released 3.0.0

- Introducing the `Updater` and `Dispatcher` classes

2015-11-11

Released 2.9.2

- Error handling on request timeouts has been improved

2015-11-10

Released 2.9.1

- Add parameter `network_delay` to `Bot.getUpdates` for slow connections

2015-11-10

Released 2.9

- Emoji class now uses `bytes_to_native_str` from future 3rd party lib
- Make `user_from` optional to work with channels
- Raise exception if Telegram times out on long-polling

Special thanks to @jh0ker for all hard work

2015-10-08

Released 2.8.7

- Type as optional for `GroupChat` class

2015-10-08

Released 2.8.6

- Adds type to `User` and `GroupChat` classes (pre-release Telegram feature)

2015-09-24

Released 2.8.5

- Handles HTTP Bad Gateway (503) errors on request
- Fixes regression on `Audio` and `Document` for unicode fields

2015-09-20

Released 2.8.4

- `getFile` and `File.download` is now fully supported

2015-09-10

Released 2.8.3

- Moved `Bot._requestURL` to its own class (`telegram.utils.request`)
- Much better, such wow, Telegram Objects tests
- Add consistency for `str` properties on Telegram Objects
- Better design to test if `chat_id` is invalid
- Add ability to set custom filename on `Bot.sendDocument(..., filename='')`
- Fix Sticker as `InputFile`
- Send JSON requests over urlencoded post data
- Markdown support for `Bot.sendMessage(..., parse_mode=ParseMode.MARKDOWN)`
- Refactor of `TelegramError` class (no more handling `IOError` or `URLError`)

2015-09-05

Released 2.8.2

- Fix regression on Telegram `ReplyMarkup`
- Add certificate to `is_inputfile` method

2015-09-05

Released 2.8.1

- Fix regression on Telegram objects with thumb properties

2015-09-04

Released 2.8

- TelegramError when chat_id is empty for send* methods
- setWebhook now supports sending self-signed certificate
- Huge redesign of existing Telegram classes
- Added support for PyPy
- Added docstring for existing classes

2015-08-19

Released 2.7.1

- Fixed JSON serialization for message

2015-08-17

Released 2.7

- Added support for Voice object and sendVoice method
- Due backward compatibility performer or/and title will be required for sendAudio
- Fixed JSON serialization when forwarded message

2015-08-15

Released 2.6.1

- Fixed parsing image header issue on < Python 2.7.3

2015-08-14

Released 2.6.0

- Depreciation of require_authentication and clearCredentials methods
- Giving AUTHORS the proper credits for their contribution for this project
- Message.date and Message.forward_date are now datetime objects

2015-08-12

Released 2.5.3

- telegram.Bot now supports to be unpickled

2015-08-11

Released 2.5.2

- New changes from Telegram Bot API have been applied
- telegram.Bot now supports to be pickled
- Return empty str instead None when message.text is empty

2015-08-10

Released 2.5.1

- Moved from GPLv2 to LGPLv3

2015-08-09

Released 2.5

- Fixes logging calls in API

2015-08-08

Released 2.4

- Fixes `Emoji` class for Python 3
- PEP8 improvements

2015-08-08

Released 2.3

- Fixes `ForceReply` class
- Remove `logging.basicConfig` from library

2015-07-25

Released 2.2

- Allows `debug=True` when initializing `telegram.Bot`

2015-07-20

Released 2.1

- Fix `to_dict` for `Document` and `Video`

2015-07-19

Released 2.0

- Fixes bugs
- Improves `__str__` over `to_json()`
- Creates abstract class `TelegramObject`

2015-07-15

Released 1.9

- Python 3 officially supported
- PEP8 improvements

2015-07-12

Released 1.8

- Fixes crash when replying an unicode text message (special thanks to JRoot3D)

2015-07-11

Released 1.7

- Fixes crash when `username` is not defined on `chat` (special thanks to JRoot3D)

2015-07-10

Released 1.6

- Improvements for GAE support

2015-07-10

Released 1.5

- Fixes randomly unicode issues when using `InputFile`

2015-07-10

Released 1.4

- `requests` lib is no longer required
- Google App Engine (GAE) is supported

2015-07-10

Released 1.3

- Added support to `setWebhook` (special thanks to macrojames)

2015-07-09

Released 1.2

- `CustomKeyboard` classes now available
- Emojis available
- PEP8 improvements

2015-07-08

Released 1.1

- PyPi package now available

2015-07-08

Released 1.0

- Initial checkin of python-telegram-bot

Indices and tables

- `genindex`
- `modindex`
- `search`

t

`telegram.constants`, [104](#)
`telegram.error`, [106](#)
`telegram.ext.filters`, [8](#)
`telegram.utils.helpers`, [52](#)

Symbols

`__call__()` (*telegram.ext.DelayQueue* method), 20
`__call__()` (*telegram.ext.MessageQueue* method), 18
`__init__()` (*telegram.ext.DelayQueue* method), 20
`__init__()` (*telegram.ext.MessageQueue* method), 19
`__weakref__` (*telegram.ext.MessageQueue* attribute), 19
`_queue` (*telegram.ext.JobQueue* attribute), 16

A

`add_error_handler()` (*telegram.ext.Dispatcher* method), 6
`add_handler()` (*telegram.ext.Dispatcher* method), 7
`add_sticker_to_set()` (*telegram.Bot* method), 57
`address` (*telegram.InlineQueryResultVenue* attribute), 173, 275
`address` (*telegram.InputVenueMessageContent* attribute), 179, 282
`address` (*telegram.SecureData* attribute), 194, 335
`address` (*telegram.Venue* attribute), 143, 314
`addStickerToSet()` (*telegram.Bot* method), 57
`ADMINISTRATOR` (*telegram.ChatMember* attribute), 103
`all` (*telegram.ext.filters.Filters* attribute), 9
`all_members_are_administrators` (*telegram.Chat* attribute), 96, 238
`ALL_TYPES` (*telegram.MessageEntity* attribute), 132
`allow_edited` (*telegram.ext.CommandHandler* attribute), 30
`allow_reentry` (*telegram.ext.ConversationHandler* attribute), 28
`allowed_updates` (*telegram.WebhookInfo* attribute), 147, 317
`amount` (*telegram.LabeledPrice* attribute), 181, 321
`Animation` (class in *telegram*), 55
`animation` (*telegram.ext.filters.Filters* attribute), 9
`animation` (*telegram.Game* attribute), 188, 318
`animation` (*telegram.Message* attribute), 120, 290
`answer()` (*telegram.CallbackQuery* method), 94

`answer()` (*telegram.InlineQuery* method), 151
`answer()` (*telegram.PreCheckoutQuery* method), 187
`answer()` (*telegram.ShippingQuery* method), 185
`answer_callback_query()` (*telegram.Bot* method), 58
`answer_inline_query()` (*telegram.Bot* method), 59
`answer_pre_checkout_query()` (*telegram.Bot* method), 59
`answer_shipping_query()` (*telegram.Bot* method), 60
`answerCallbackQuery()` (*telegram.Bot* method), 58
`answerInlineQuery()` (*telegram.Bot* method), 58
`answerPreCheckoutQuery()` (*telegram.Bot* method), 58
`answerShippingQuery()` (*telegram.Bot* method), 58
`args` (*telegram.ext.CallbackContext* attribute), 21
`args` (*telegram.utils.promise.Promise* attribute), 53
`attach` (*telegram.InputFile* attribute), 111, 281
`Audio` (class in *telegram*), 56
`audio` (*telegram.ext.filters.Filters* attribute), 10
`audio` (*telegram.Message* attribute), 119, 290
`audio_duration` (*telegram.InlineQueryResultAudio* attribute), 154, 257
`audio_file_id` (*telegram.InlineQueryResultCachedAudio* attribute), 155, 258
`audio_url` (*telegram.InlineQueryResultAudio* attribute), 153, 257
`author_signature` (*telegram.Message* attribute), 121, 292

B

`BadRequest`, 106
`bank_statement` (*telegram.SecureData* attribute), 194, 335
`BaseFilter` (class in *telegram.ext.filters*), 8
`BasePersistence` (class in *telegram.ext*), 47
`big_file_id` (*telegram.ChatPhoto* attribute), 103, 327

`birth_date` (*telegram.PersonalDetails* attribute), 195, 337
`BOLD` (*telegram.MessageEntity* attribute), 132
`Bot` (class in *telegram*), 57
`bot` (*telegram.Audio* attribute), 56, 201
`bot` (*telegram.Document* attribute), 106, 249
`bot` (*telegram.EncryptedPassportElement* attribute), 199, 287
`bot` (*telegram.ext.CallbackContext* attribute), 21
`bot` (*telegram.ext.Dispatcher* attribute), 6
`bot` (*telegram.ext.JobQueue* attribute), 16
`bot` (*telegram.ext.Updater* attribute), 3
`bot` (*telegram.Message* attribute), 121, 292
`bot` (*telegram.PassportData* attribute), 196, 288
`bot` (*telegram.PassportFile* attribute), 197, 285
`bot` (*telegram.PhotoSize* attribute), 133, 304
`bot` (*telegram.PreCheckoutQuery* attribute), 186, 324
`bot` (*telegram.ShippingQuery* attribute), 185, 326
`bot` (*telegram.Sticker* attribute), 148, 308
`bot` (*telegram.User* attribute), 140, 311
`bot` (*telegram.Video* attribute), 144, 315
`bot` (*telegram.VideoNote* attribute), 145, 320
`bot` (*telegram.Voice* attribute), 146, 316
`BOT_COMMAND` (*telegram.MessageEntity* attribute), 132
`burst_limit` (*telegram.ext.DelayQueue* attribute), 19

C

`callback` (*telegram.ext.CallbackQueryHandler* attribute), 24
`callback` (*telegram.ext.ChosenInlineResultHandler* attribute), 26
`callback` (*telegram.ext.CommandHandler* attribute), 30
`callback` (*telegram.ext.Handler* attribute), 22
`callback` (*telegram.ext.InlineQueryHandler* attribute), 32
`callback` (*telegram.ext.Job* attribute), 14
`callback` (*telegram.ext.MessageHandler* attribute), 34
`callback` (*telegram.ext.PreCheckoutQueryHandler* attribute), 36
`callback` (*telegram.ext.PrefixHandler* attribute), 38
`callback` (*telegram.ext.RegexHandler* attribute), 40
`callback` (*telegram.ext.ShippingQueryHandler* attribute), 41
`callback` (*telegram.ext.StringCommandHandler* attribute), 43
`callback` (*telegram.ext.StringRegexHandler* attribute), 44
`callback` (*telegram.ext.TypeHandler* attribute), 46
`callback_data` (*telegram.InlineKeyboardButton* attribute), 109, 252
`callback_game` (*telegram.InlineKeyboardButton* attribute), 109, 252
`callback_query` (*telegram.Update* attribute), 138, 309

`CallbackContext` (class in *telegram.ext*), 20
`CallbackGame` (class in *telegram*), 189
`CallbackQuery` (class in *telegram*), 93
`CallbackQueryHandler` (class in *telegram.ext*), 24
`can_add_web_page_previews` (*telegram.ChatMember* attribute), 102, 243
`can_be_edited` (*telegram.ChatMember* attribute), 101, 242
`can_change_info` (*telegram.ChatMember* attribute), 101, 242
`can_delete_messages` (*telegram.ChatMember* attribute), 101, 243
`can_edit_messages` (*telegram.ChatMember* attribute), 101, 242
`can_invite_users` (*telegram.ChatMember* attribute), 101, 243
`can_pin_messages` (*telegram.ChatMember* attribute), 101, 243
`can_post_messages` (*telegram.ChatMember* attribute), 101, 242
`can_promote_members` (*telegram.ChatMember* attribute), 101, 243
`can_restrict_members` (*telegram.ChatMember* attribute), 101, 243
`can_send_media_messages` (*telegram.ChatMember* attribute), 102, 243
`can_send_messages` (*telegram.ChatMember* attribute), 102, 243
`can_send_other_messages` (*telegram.ChatMember* attribute), 102, 243
`can_set_sticker_set` (*telegram.Chat* attribute), 96, 238
`caption` (*telegram.InlineQueryResultAudio* attribute), 154, 257
`caption` (*telegram.InlineQueryResultCachedAudio* attribute), 155, 258
`caption` (*telegram.InlineQueryResultCachedDocument* attribute), 156, 260
`caption` (*telegram.InlineQueryResultCachedGif* attribute), 157, 261
`caption` (*telegram.InlineQueryResultCachedMpeg4Gif* attribute), 158, 262
`caption` (*telegram.InlineQueryResultCachedPhoto* attribute), 159, 263
`caption` (*telegram.InlineQueryResultCachedVideo* attribute), 161, 265
`caption` (*telegram.InlineQueryResultCachedVoice* attribute), 163, 266
`caption` (*telegram.InlineQueryResultDocument* attribute), 165, 268
`caption` (*telegram.InlineQueryResultGif* attribute), 167, 270
`caption` (*telegram.InlineQueryResultMpeg4Gif* attribute), 170, 273
`caption` (*telegram.InlineQueryResultPhoto* attribute), 172, 274
`caption` (*telegram.InlineQueryResultVideo* at-

- tribute*), 175, 277
- caption* (*telegram.InlineQueryResultVoice attribute*), 176, 279
- caption* (*telegram.InputMediaAnimation attribute*), 112, 339
- caption* (*telegram.InputMediaAudio attribute*), 113, 340
- caption* (*telegram.InputMediaDocument attribute*), 114, 341
- caption* (*telegram.InputMediaPhoto attribute*), 115, 329
- caption* (*telegram.InputMediaVideo attribute*), 116, 330, 338
- caption* (*telegram.Message attribute*), 120, 291
- caption_entities* (*telegram.Message attribute*), 119, 290
- caption_html* (*telegram.Message attribute*), 124
- caption_html_urled* (*telegram.Message attribute*), 124
- caption_markdown* (*telegram.Message attribute*), 124
- caption_markdown_urled* (*telegram.Message attribute*), 124
- CASHTAG (*telegram.MessageEntity attribute*), 132
- CHANNEL (*telegram.Chat attribute*), 97
- channel_chat_created* (*telegram.Message attribute*), 121, 291
- channel_post* (*telegram.ext.filters.Filters attribute*), 13
- channel_post* (*telegram.Update attribute*), 138, 309
- channel_post_updates* (*telegram.ext.MessageHandler attribute*), 35
- channel_posts* (*telegram.ext.filters.Filters attribute*), 13
- Chat (class in telegram), 95
- chat* (*telegram.Message attribute*), 119, 289
- chat_created* (*telegram.ext.filters.Filters attribute*), 12
- chat_data* (*telegram.ext.CallbackContext attribute*), 21
- chat_data* (*telegram.ext.DictPersistence attribute*), 50
- chat_data* (*telegram.ext.Dispatcher attribute*), 6
- chat_data_json* (*telegram.ext.DictPersistence attribute*), 50
- chat_id* (*telegram.Message attribute*), 124
- chat_instance* (*telegram.CallbackQuery attribute*), 93, 247
- ChatAction (class in telegram), 100
- ChatMember (class in telegram), 101
- ChatMigrated, 106
- ChatPhoto (class in telegram), 103
- check_update()* (*telegram.ext.CallbackQueryHandler method*), 25
- check_update()* (*telegram.ext.ChosenInlineResultHandler method*), 27
- check_update()* (*telegram.ext.CommandHandler method*), 31
- check_update()* (*telegram.ext.ConversationHandler method*), 29
- check_update()* (*telegram.ext.Handler method*), 23
- check_update()* (*telegram.ext.InlineQueryHandler method*), 33
- check_update()* (*telegram.ext.MessageHandler method*), 36
- check_update()* (*telegram.ext.PreCheckoutQueryHandler method*), 37
- check_update()* (*telegram.ext.PrefixHandler method*), 39
- check_update()* (*telegram.ext.ShippingQueryHandler method*), 42
- check_update()* (*telegram.ext.StringCommandHandler method*), 43
- check_update()* (*telegram.ext.StringRegexHandler method*), 45
- check_update()* (*telegram.ext.TypeHandler method*), 47
- CHIN (*telegram.MaskPosition attribute*), 150
- chosen_inline_result* (*telegram.Update attribute*), 138, 309
- ChosenInlineResult (class in telegram), 180
- ChosenInlineResultHandler (class in telegram.ext), 26
- city* (*telegram.ResidentialAddress attribute*), 196, 337
- city* (*telegram.ShippingAddress attribute*), 182, 323
- CODE (*telegram.MessageEntity attribute*), 132
- collect_additional_context()* (*telegram.ext.CallbackQueryHandler method*), 25
- collect_additional_context()* (*telegram.ext.CommandHandler method*), 31
- collect_additional_context()* (*telegram.ext.Handler method*), 23
- collect_additional_context()* (*telegram.ext.InlineQueryHandler method*), 34
- collect_additional_context()* (*telegram.ext.MessageHandler method*), 36
- collect_additional_context()* (*telegram.ext.PrefixHandler method*), 39
- collect_additional_context()* (*telegram.ext.StringCommandHandler method*), 44
- collect_additional_context()* (*telegram.ext.StringRegexHandler method*), 45

- 45
- `collect_optional_args()` (*telegram.ext.CallbackQueryHandler* method), 25
- `collect_optional_args()` (*telegram.ext.CommandHandler* method), 32
- `collect_optional_args()` (*telegram.ext.Handler* method), 23
- `collect_optional_args()` (*telegram.ext.InlineQueryHandler* method), 34
- `collect_optional_args()` (*telegram.ext.RegexHandler* method), 41
- `collect_optional_args()` (*telegram.ext.StringCommandHandler* method), 44
- `collect_optional_args()` (*telegram.ext.StringRegexHandler* method), 45
- `command` (*telegram.ext.CommandHandler* attribute), 30
- `command` (*telegram.ext.filters.Filters* attribute), 10
- `command` (*telegram.ext.PrefixHandler* attribute), 38
- `command` (*telegram.ext.StringCommandHandler* attribute), 43
- `CommandHandler` (class in *telegram.ext*), 30
- `con_pool_size` (*telegram.utils.request.Request* attribute), 54
- `Conflict`, 106
- `connected_website` (*telegram.Message* attribute), 121, 292
- `Contact` (class in *telegram*), 104
- `contact` (*telegram.ext.filters.Filters* attribute), 10
- `contact` (*telegram.Message* attribute), 120, 291
- `contains_masks` (*telegram.StickerSet* attribute), 149, 328
- `context` (*telegram.ext.Job* attribute), 15
- `conversation_timeout` (*telegram.ext.ConversationHandler* attribute), 28
- `ConversationHandler` (class in *telegram.ext*), 27
- `conversations` (*telegram.ext.DictPersistence* attribute), 50
- `conversations_json` (*telegram.ext.DictPersistence* attribute), 51
- `country_code` (*telegram.PersonalDetails* attribute), 195, 337
- `country_code` (*telegram.ResidentialAddress* attribute), 196, 337
- `country_code` (*telegram.ShippingAddress* attribute), 182, 323
- `create_new_sticker_set()` (*telegram.Bot* method), 60
- `createNewStickerSet()` (*telegram.Bot* method), 60
- `CREATOR` (*telegram.ChatMember* attribute), 103
- `Credentials` (class in *telegram*), 193
- `credentials` (*telegram.PassportData* attribute), 196, 288
- `currency` (*telegram.Invoice* attribute), 181, 325
- `currency` (*telegram.PreCheckoutQuery* attribute), 186, 324
- `currency` (*telegram.SuccessfulPayment* attribute), 184, 321
- D**
- `data` (*telegram.CallbackQuery* attribute), 93, 247
- `data` (*telegram.EncryptedCredentials* attribute), 200, 284
- `data` (*telegram.EncryptedPassportElement* attribute), 198, 286
- `data` (*telegram.PassportData* attribute), 196, 288
- `data_filter` (*telegram.ext.filters.BaseFilter* attribute), 9
- `data_hash` (*telegram.PassportElementErrorDataField* attribute), 192, 333
- `DataCredentials` (class in *telegram*), 193
- `date` (*telegram.Message* attribute), 118, 289
- `days` (*telegram.ext.Job* attribute), 15
- `de_json()` (*telegram.Update* class method), 139
- `de_json()` (*telegram.User* class method), 140
- `de_list()` (*telegram.User* class method), 140
- `decode_conversations_from_json()` (in module *telegram.utils.helpers*), 52
- `decode_user_chat_data_from_json()` (in module *telegram.utils.helpers*), 52
- `decrypted_credentials` (*telegram.PassportData* attribute), 197
- `decrypted_data` (*telegram.EncryptedCredentials* attribute), 200
- `decrypted_data` (*telegram.PassportData* attribute), 197
- `decrypted_secret` (*telegram.EncryptedCredentials* attribute), 201
- `DelayQueue` (class in *telegram.ext*), 19
- `delete()` (*telegram.Message* method), 124
- `delete_chat_photo` (*telegram.ext.filters.Filters* attribute), 12
- `delete_chat_photo` (*telegram.Message* attribute), 121, 291
- `delete_chat_photo()` (*telegram.Bot* method), 61
- `delete_chat_sticker_set()` (*telegram.Bot* method), 62
- `delete_message()` (*telegram.Bot* method), 62
- `delete_sticker_from_set()` (*telegram.Bot* method), 62
- `delete_webhook()` (*telegram.Bot* method), 63
- `deleteChatPhoto()` (*telegram.Bot* method), 61
- `deleteChatStickerSet()` (*telegram.Bot* method), 61
- `deleteMessage()` (*telegram.Bot* method), 61
- `deleteStickerFromSet()` (*telegram.Bot* method), 61
- `deleteWebhook()` (*telegram.Bot* method), 61
- `description` (*telegram.Chat* attribute), 96, 238

description (*telegram.Game* attribute), 187, 318
 description (*telegram.InlineQueryResultArticle* attribute), 152, 256
 description (*telegram.InlineQueryResultCachedDocument* attribute), 156, 259
 description (*telegram.InlineQueryResultCachedPhoto* attribute), 159, 263
 description (*telegram.InlineQueryResultCachedVideo* attribute), 161, 265
 description (*telegram.InlineQueryResultDocument* attribute), 165, 268
 description (*telegram.InlineQueryResultPhoto* attribute), 172, 274
 description (*telegram.InlineQueryResultVideo* attribute), 175, 277
 description (*telegram.Invoice* attribute), 181, 325
 DictPersistence (class in *telegram.ext*), 50
 disable_web_page_preview (*telegram.InputTextMessageContent* attribute), 178, 282
 dispatch_error() (*telegram.ext.Dispatcher* method), 7
 Dispatcher (class in *telegram.ext*), 6
 dispatcher (*telegram.ext.Updater* attribute), 3
 Document (class in *telegram*), 105
 document (*telegram.ext.filters.Filters* attribute), 10
 document (*telegram.Message* attribute), 119, 290
 document_file_id (*telegram.InlineQueryResultCachedDocument* attribute), 156, 259
 document_no (*telegram.IdDocumentData* attribute), 194, 336
 document_url (*telegram.InlineQueryResultDocument* attribute), 165, 268
 done (*telegram.utils.promise.Promise* attribute), 53
 download() (*telegram.File* method), 107
 download() (*telegram.utils.request.Request* method), 54
 download_as_bytearray() (*telegram.File* method), 108
 driver_license (*telegram.SecureData* attribute), 193, 335
 duration (*telegram.Animation* attribute), 55, 317
 duration (*telegram.Audio* attribute), 56, 201
 duration (*telegram.InputMediaAnimation* attribute), 112, 340
 duration (*telegram.InputMediaAudio* attribute), 113, 341
 duration (*telegram.InputMediaVideo* attribute), 116, 330, 338
 duration (*telegram.Video* attribute), 143, 314
 duration (*telegram.VideoNote* attribute), 145, 320
 duration (*telegram.Voice* attribute), 145, 315

E

edit_caption() (*telegram.Message* method), 125
 edit_date (*telegram.Message* attribute), 119, 290
 edit_media() (*telegram.Message* method), 125
 edit_message_caption() (*telegram.Bot* method), 63
 edit_message_caption() (*telegram.CallbackQuery* method), 94
 edit_message_live_location() (*telegram.Bot* method), 64
 edit_message_media() (*telegram.Bot* method), 64
 edit_message_reply_markup() (*telegram.Bot* method), 65
 edit_message_reply_markup() (*telegram.CallbackQuery* method), 94
 edit_message_text() (*telegram.Bot* method), 65
 edit_message_text() (*telegram.CallbackQuery* method), 95
 edit_reply_markup() (*telegram.Message* method), 125
 edit_text() (*telegram.Message* method), 125
 edited_channel_post (*telegram.ext.filters.Filters* attribute), 13
 edited_channel_post (*telegram.Update* attribute), 138, 309
 edited_message (*telegram.ext.filters.Filters* attribute), 13
 edited_message (*telegram.Update* attribute), 138, 309
 edited_updates (*telegram.ext.MessageHandler* attribute), 35
 editMessageCaption() (*telegram.Bot* method), 63
 editMessageLiveLocation() (*telegram.Bot* method), 63
 editMessageMedia() (*telegram.Bot* method), 63
 editMessageReplyMarkup() (*telegram.Bot* method), 63
 editMessageText() (*telegram.Bot* method), 63
 effective_attachment (*telegram.Message* attribute), 126
 effective_chat (*telegram.Update* attribute), 139
 effective_message (*telegram.Update* attribute), 139
 effective_message_type() (in module *telegram.utils.helpers*), 52
 effective_user (*telegram.Update* attribute), 139
 element_hash (*telegram.PassportElementErrorUnspecified* attribute), 344
 email (*telegram.EncryptedPassportElement* attribute), 198, 286
 EMAIL (*telegram.MessageEntity* attribute), 132
 email (*telegram.OrderInfo* attribute), 183, 325
 emoji (*telegram.Sticker* attribute), 148, 307
 enabled (*telegram.ext.Job* attribute), 15
 EncryptedCredentials (class in *telegram*), 200

EncryptedPassportElement (class in telegram), 198
END (telegram.ext.ConversationHandler attribute), 29
encode_conversations_to_json() (in module telegram.utils.helpers), 52
entities (telegram.Message attribute), 119, 290
entry_points (telegram.ext.ConversationHandler attribute), 27
error (telegram.ext.CallbackContext attribute), 21
error_handlers (telegram.ext.Dispatcher attribute), 7
escape_markdown() (in module telegram.utils.helpers), 52
exc_route (telegram.ext.DelayQueue attribute), 19
exception (telegram.utils.promise.Promise attribute), 53
expiry_date (telegram.IdDocumentData attribute), 195, 336
export_chat_invite_link() (telegram.Bot method), 66
exportChatInviteLink() (telegram.Bot method), 66
EYES (telegram.MaskPosition attribute), 150

F

fallbacks (telegram.ext.ConversationHandler attribute), 28
field_name (telegram.PassportElementErrorDataField attribute), 192, 333
File (class in telegram), 107
file_date (telegram.PassportFile attribute), 197, 285
file_hash (telegram.PassportElementErrorFile attribute), 190, 331, 334
file_hash (telegram.PassportElementErrorFiles attribute), 192, 333
file_hash (telegram.PassportElementErrorFrontSide attribute), 191, 332
file_hash (telegram.PassportElementErrorReverseSide attribute), 190, 332
file_hash (telegram.PassportElementErrorSelfie attribute), 342
file_hash (telegram.PassportElementErrorTranslationFile attribute), 343
file_hash (telegram.PassportElementErrorTranslationFiles attribute), 343
file_id (telegram.Animation attribute), 55, 317
file_id (telegram.Audio attribute), 56, 201
file_id (telegram.Document attribute), 105, 249
file_id (telegram.File attribute), 107, 250
file_id (telegram.PassportFile attribute), 197, 285
file_id (telegram.PhotoSize attribute), 133, 303
file_id (telegram.Sticker attribute), 147, 307
file_id (telegram.Video attribute), 143, 314
file_id (telegram.VideoNote attribute), 144, 320
file_id (telegram.Voice attribute), 145, 315
file_name (telegram.Animation attribute), 55, 317
file_name (telegram.Document attribute), 105, 249
file_path (telegram.File attribute), 107, 250
file_size (telegram.Animation attribute), 55, 318
file_size (telegram.Audio attribute), 56, 201
file_size (telegram.Document attribute), 105, 249
file_size (telegram.File attribute), 107, 250
file_size (telegram.PassportFile attribute), 197, 285
file_size (telegram.PhotoSize attribute), 133, 304
file_size (telegram.Sticker attribute), 148, 308
file_size (telegram.Video attribute), 144, 315
file_size (telegram.VideoNote attribute), 145, 320
file_size (telegram.Voice attribute), 146, 316
FileCredentials (class in telegram), 194
filename (telegram.ext.PicklePersistence attribute), 48
filename (telegram.InputFile attribute), 111, 281
files (telegram.EncryptedPassportElement attribute), 198, 286
filter() (telegram.ext.filters.BaseFilter method), 9
filter() (telegram.ext.filters.InvertedFilter method), 14
filter() (telegram.ext.filters.MergedFilter method), 14
Filters (class in telegram.ext.filters), 9
filters (telegram.ext.CommandHandler attribute), 30
filters (telegram.ext.MessageHandler attribute), 34
filters (telegram.ext.PrefixHandler attribute), 38
Filters.caption_entity (class in telegram.ext.filters), 10
Filters.chat (class in telegram.ext.filters), 10
Filters.entity (class in telegram.ext.filters), 10
Filters.language (class in telegram.ext.filters), 11
Filters.regex (class in telegram.ext.filters), 11
Filters.user (class in telegram.ext.filters), 13
FIND_LOCATION (telegram.ChatAction attribute), 100
first_name (telegram.Bot attribute), 66
first_name (telegram.Chat attribute), 95, 238
first_name (telegram.Contact attribute), 105, 249
first_name (telegram.InlineQueryResultContact attribute), 164, 267
first_name (telegram.InputContactMessageContent attribute), 179, 280
first_name (telegram.PersonalDetails attribute), 195, 336, 337
first_name (telegram.User attribute), 139, 311
flush() (telegram.ext.BasePersistence method), 47
flush() (telegram.ext.PicklePersistence method), 49
force_reply (telegram.ForceReply attribute), 108, 251
ForceReply (class in telegram), 108
FOREHEAD (telegram.MaskPosition attribute), 150
forward() (telegram.Message method), 126
forward_date (telegram.Message attribute), 119, 289
forward_from (telegram.Message attribute), 119,

- 289
- `forward_from_chat` (*telegram.Message* attribute), 119, 289
- `forward_from_message_id` (*telegram.Message* attribute), 119, 289
- `forward_message()` (*telegram.Bot* method), 66
- `forward_signature` (*telegram.Message* attribute), 121, 292
- `forwarded` (*telegram.ext.filters.Filters* attribute), 10
- `forwardMessage()` (*telegram.Bot* method), 66
- `foursquare_id` (*telegram.InlineQueryResultVenue* attribute), 173, 276
- `foursquare_id` (*telegram.InputVenueMessageContent* attribute), 179, 282
- `foursquare_id` (*telegram.Venue* attribute), 143, 314
- `foursquare_type` (*telegram.InlineQueryResultVenue* attribute), 173, 276
- `foursquare_type` (*telegram.InputVenueMessageContent* attribute), 179, 282
- `foursquare_type` (*telegram.Venue* attribute), 143, 314
- `from_button()` (*telegram.InlineKeyboardMarkup* class method), 110
- `from_button()` (*telegram.ReplyKeyboardMarkup* class method), 135
- `from_column()` (*telegram.InlineKeyboardMarkup* class method), 110
- `from_column()` (*telegram.ReplyKeyboardMarkup* class method), 136
- `from_row()` (*telegram.InlineKeyboardMarkup* class method), 110
- `from_row()` (*telegram.ReplyKeyboardMarkup* class method), 136
- `from_timestamp()` (in module *telegram.utils.helpers*), 52
- `from_user` (*telegram.CallbackQuery* attribute), 93, 246
- `from_user` (*telegram.ChosenInlineResult* attribute), 180, 245
- `from_user` (*telegram.InlineQuery* attribute), 150, 254
- `from_user` (*telegram.Message* attribute), 118, 289
- `from_user` (*telegram.PreCheckoutQuery* attribute), 186, 323
- `from_user` (*telegram.ShippingQuery* attribute), 185, 326
- `front_side` (*telegram.EncryptedPassportElement* attribute), 198, 286
- `full_name` (*telegram.User* attribute), 140
- G**
- `Game` (class in *telegram*), 187
- `game` (*telegram.ext.filters.Filters* attribute), 11
- `game` (*telegram.Message* attribute), 120, 290
- `game_short_name` (*telegram.CallbackQuery* attribute), 93, 247
- `game_short_name` (*telegram.InlineQueryResultGame* attribute), 166, 280
- `GameHighScore` (class in *telegram*), 189
- `gender` (*telegram.PersonalDetails* attribute), 195, 337
- `get()` (*telegram.utils.request.Request* method), 54
- `get_administrators()` (*telegram.Chat* method), 97
- `get_chat()` (*telegram.Bot* method), 67
- `get_chat_administrators()` (*telegram.Bot* method), 68
- `get_chat_data()` (*telegram.ext.BasePersistence* method), 47
- `get_chat_data()` (*telegram.ext.DictPersistence* method), 51
- `get_chat_data()` (*telegram.ext.PicklePersistence* method), 49
- `get_chat_member()` (*telegram.Bot* method), 68
- `get_chat_members_count()` (*telegram.Bot* method), 68
- `get_conversations()` (*telegram.ext.BasePersistence* method), 48
- `get_conversations()` (*telegram.ext.DictPersistence* method), 51
- `get_conversations()` (*telegram.ext.PicklePersistence* method), 49
- `get_file()` (*telegram.Audio* method), 57
- `get_file()` (*telegram.Bot* method), 69
- `get_file()` (*telegram.Document* method), 106
- `get_file()` (*telegram.PassportFile* method), 198
- `get_file()` (*telegram.PhotoSize* method), 134
- `get_file()` (*telegram.Sticker* method), 148
- `get_file()` (*telegram.Video* method), 144
- `get_file()` (*telegram.VideoNote* method), 145
- `get_file()` (*telegram.Voice* method), 146
- `get_game_high_scores()` (*telegram.Bot* method), 69
- `get_instance()` (*telegram.ext.Dispatcher* class method), 7
- `get_jobs_by_name()` (*telegram.ext.JobQueue* method), 16
- `get_me()` (*telegram.Bot* method), 69
- `get_member()` (*telegram.Chat* method), 97
- `get_members_count()` (*telegram.Chat* method), 97
- `get_profile_photos()` (*telegram.User* method), 140
- `get_signal_name()` (in module *telegram.utils.helpers*), 52
- `get_sticker_set()` (*telegram.Bot* method), 69
- `get_updates()` (*telegram.Bot* method), 70
- `get_user_data()` (*telegram.ext.BasePersistence* method), 48
- `get_user_data()` (*telegram.ext.DictPersistence* method), 51
- `get_user_data()` (*telegram.ext.PicklePersistence*

method), 49
get_user_profile_photos() (telegram.Bot method), 70
get_webhook_info() (telegram.Bot method), 71
getChat() (telegram.Bot method), 67
getChatAdministrators() (telegram.Bot method), 67
getChatMember() (telegram.Bot method), 67
getChatMembersCount() (telegram.Bot method), 67
getFile() (telegram.Bot method), 67
getGameHighScores() (telegram.Bot method), 67
getMe() (telegram.Bot method), 67
getStickerSet() (telegram.Bot method), 67
getUpdates() (telegram.Bot method), 67
getUserProfilePhotos() (telegram.Bot method), 67
getWebhookInfo() (telegram.Bot method), 67
gif_duration (telegram.InlineQueryResultGif attribute), 167, 270
gif_file_id (telegram.InlineQueryResultCachedGif attribute), 157, 260
gif_height (telegram.InlineQueryResultGif attribute), 167, 270
gif_url (telegram.InlineQueryResultGif attribute), 167, 270
gif_width (telegram.InlineQueryResultGif attribute), 167, 270
GROUP (telegram.Chat attribute), 97
group (telegram.ext.filters.Filters attribute), 11
group_chat_created (telegram.Message attribute), 121, 291
groups (telegram.ext.Dispatcher attribute), 7

H

handle_update() (telegram.ext.ConversationHandler method), 29
handle_update() (telegram.ext.Handler method), 23
Handler (class in telegram.ext), 22
handlers (telegram.ext.Dispatcher attribute), 7
has_custom_certificate (telegram.WebhookInfo attribute), 146, 316
hash (telegram.DataCredentials attribute), 193, 335
hash (telegram.EncryptedCredentials attribute), 200, 284
hash (telegram.EncryptedPassportElement attribute), 199, 286
hash (telegram.FileCredentials attribute), 194, 336
HASHTAG (telegram.MessageEntity attribute), 132
height (telegram.Animation attribute), 55, 317
height (telegram.InputMediaAnimation attribute), 112, 340
height (telegram.InputMediaVideo attribute), 116, 330, 338
height (telegram.PhotoSize attribute), 133, 303

height (telegram.Sticker attribute), 147, 307
height (telegram.Video attribute), 143, 314
hide_url (telegram.InlineQueryResultArticle attribute), 152, 256
HTML (telegram.ParseMode attribute), 133

I

id (telegram.Bot attribute), 71
id (telegram.CallbackQuery attribute), 93, 246
id (telegram.Chat attribute), 95, 237
id (telegram.InlineQuery attribute), 150, 254
id (telegram.InlineQueryResult attribute), 152, 255
id (telegram.InlineQueryResultArticle attribute), 152, 256
id (telegram.InlineQueryResultAudio attribute), 153, 257
id (telegram.InlineQueryResultCachedAudio attribute), 155, 258
id (telegram.InlineQueryResultCachedDocument attribute), 156, 259
id (telegram.InlineQueryResultCachedGif attribute), 157, 260
id (telegram.InlineQueryResultCachedMpeg4Gif attribute), 158, 261
id (telegram.InlineQueryResultCachedPhoto attribute), 159, 263
id (telegram.InlineQueryResultCachedSticker attribute), 160, 264
id (telegram.InlineQueryResultCachedVideo attribute), 161, 264
id (telegram.InlineQueryResultCachedVoice attribute), 162, 266
id (telegram.InlineQueryResultContact attribute), 163, 267
id (telegram.InlineQueryResultDocument attribute), 165, 268
id (telegram.InlineQueryResultGame attribute), 166, 279
id (telegram.InlineQueryResultGif attribute), 167, 270
id (telegram.InlineQueryResultLocation attribute), 168, 271
id (telegram.InlineQueryResultMpeg4Gif attribute), 170, 272
id (telegram.InlineQueryResultPhoto attribute), 171, 274
id (telegram.InlineQueryResultVenue attribute), 173, 275
id (telegram.InlineQueryResultVideo attribute), 174, 277
id (telegram.InlineQueryResultVoice attribute), 176, 278
id (telegram.PreCheckoutQuery attribute), 186, 323
id (telegram.ShippingOption attribute), 183, 322
id (telegram.ShippingQuery attribute), 185, 326
id (telegram.User attribute), 139, 310
IdDocumentData (class in telegram), 194
identity_card (telegram.SecureData attribute), 194, 335

[idle\(\)](#) (*telegram.ext.Updater* method), 4
[inline_keyboard](#) (*telegram.InlineKeyboardMarkup* attribute), 110, 253
[inline_message_id](#) (*telegram.CallbackQuery* attribute), 93, 246
[inline_message_id](#) (*telegram.ChosenInlineResult* attribute), 180, 246
[inline_query](#) (*telegram.Update* attribute), 138, 309
[InlineKeyboardButton](#) (class in *telegram*), 109
[InlineKeyboardMarkup](#) (class in *telegram*), 110
[InlineQuery](#) (class in *telegram*), 150
[InlineQueryHandler](#) (class in *telegram.ext*), 32
[InlineQueryResult](#) (class in *telegram*), 151
[InlineQueryResultArticle](#) (class in *telegram*), 152
[InlineQueryResultAudio](#) (class in *telegram*), 153
[InlineQueryResultCachedAudio](#) (class in *telegram*), 154
[InlineQueryResultCachedDocument](#) (class in *telegram*), 156
[InlineQueryResultCachedGif](#) (class in *telegram*), 157
[InlineQueryResultCachedMpeg4Gif](#) (class in *telegram*), 158
[InlineQueryResultCachedPhoto](#) (class in *telegram*), 159
[InlineQueryResultCachedSticker](#) (class in *telegram*), 160
[InlineQueryResultCachedVideo](#) (class in *telegram*), 161
[InlineQueryResultCachedVoice](#) (class in *telegram*), 162
[InlineQueryResultContact](#) (class in *telegram*), 163
[InlineQueryResultDocument](#) (class in *telegram*), 165
[InlineQueryResultGame](#) (class in *telegram*), 166
[InlineQueryResultGif](#) (class in *telegram*), 167
[InlineQueryResultLocation](#) (class in *telegram*), 168
[InlineQueryResultMpeg4Gif](#) (class in *telegram*), 170
[InlineQueryResultPhoto](#) (class in *telegram*), 171
[InlineQueryResultVenue](#) (class in *telegram*), 173
[InlineQueryResultVideo](#) (class in *telegram*), 174
[InlineQueryResultVoice](#) (class in *telegram*), 176
[input_file_content](#) (*telegram.InputFile* attribute), 111, 280
[input_message_content](#) (*telegram.InlineQueryResultArticle* attribute), 152, 256
[input_message_content](#) (*telegram.InlineQueryResultAudio* attribute), 154, 258
[input_message_content](#) (*telegram.InlineQueryResultCachedAudio* attribute), 155, 259
[input_message_content](#) (*telegram.InlineQueryResultCachedDocument* attribute), 156, 260
[input_message_content](#) (*telegram.InlineQueryResultCachedGif* attribute), 157, 261
[input_message_content](#) (*telegram.InlineQueryResultCachedMpeg4Gif* attribute), 159, 262
[input_message_content](#) (*telegram.InlineQueryResultCachedPhoto* attribute), 160, 263
[input_message_content](#) (*telegram.InlineQueryResultCachedSticker* attribute), 161, 264
[input_message_content](#) (*telegram.InlineQueryResultCachedVideo* attribute), 162, 265
[input_message_content](#) (*telegram.InlineQueryResultCachedVoice* attribute), 163, 266
[input_message_content](#) (*telegram.InlineQueryResultContact* attribute), 164, 267
[input_message_content](#) (*telegram.InlineQueryResultDocument* attribute), 165, 269
[input_message_content](#) (*telegram.InlineQueryResultGif* attribute), 168, 270
[input_message_content](#) (*telegram.InlineQueryResultLocation* attribute), 169, 271
[input_message_content](#) (*telegram.InlineQueryResultMpeg4Gif* attribute), 171, 273
[input_message_content](#) (*telegram.InlineQueryResultPhoto* attribute), 172, 274
[input_message_content](#) (*telegram.InlineQueryResultVenue* attribute), 173, 276
[input_message_content](#) (*telegram.InlineQueryResultVideo* attribute), 175, 278
[input_message_content](#) (*telegram.InlineQueryResultVoice* attribute), 177, 279
[InputContactMessageContent](#) (class in *telegram*), 179
[InputFile](#) (class in *telegram*), 111

- `InputLocationMessageContent` (class in `telegram`), 178
- `InputMedia` (class in `telegram`), 111
- `InputMediaAnimation` (class in `telegram`), 112
- `InputMediaAudio` (class in `telegram`), 113
- `InputMediaDocument` (class in `telegram`), 114
- `InputMediaPhoto` (class in `telegram`), 115
- `InputMediaVideo` (class in `telegram`), 116
- `InputMessageContent` (class in `telegram`), 177
- `InputTextMessageContent` (class in `telegram`), 177
- `InputVenueMessageContent` (class in `telegram`), 178
- `internal_passport` (`telegram.SecureData` attribute), 193, 335
- `interval` (`telegram.ext.Job` attribute), 15
- `interval_seconds` (`telegram.ext.Job` attribute), 15
- `InvalidToken`, 106
- `InvertedFilter` (class in `telegram.ext.filters`), 14
- `invite_link` (`telegram.Chat` attribute), 96, 238
- `Invoice` (class in `telegram`), 181
- `invoice` (`telegram.ext.filters.Filters` attribute), 11
- `invoice` (`telegram.Message` attribute), 121, 292
- `invoice_payload` (`telegram.PreCheckoutQuery` attribute), 186, 324
- `invoice_payload` (`telegram.ShippingQuery` attribute), 185, 326
- `invoice_payload` (`telegram.SuccessfulPayment` attribute), 184, 321
- `is_bot` (`telegram.User` attribute), 139, 310
- `is_image()` (`telegram.InputFile` static method), 111
- `ITALIC` (`telegram.MessageEntity` attribute), 132
- ## J
- `Job` (class in `telegram.ext`), 14
- `job` (`telegram.ext.CallbackContext` attribute), 21
- `job_queue` (`telegram.ext.CallbackContext` attribute), 21
- `job_queue` (`telegram.ext.Dispatcher` attribute), 6
- `job_queue` (`telegram.ext.Job` attribute), 15
- `job_queue` (`telegram.ext.Updater` attribute), 3
- `JobQueue` (class in `telegram.ext`), 16
- `jobs()` (`telegram.ext.JobQueue` method), 16
- ## K
- `keyboard` (`telegram.ReplyKeyboardMarkup` attribute), 135, 305
- `KeyboardButton` (class in `telegram`), 117
- `kick_chat_member()` (`telegram.Bot` method), 71
- `kick_member()` (`telegram.Chat` method), 97
- `kickChatMember()` (`telegram.Bot` method), 71
- `KICKED` (`telegram.ChatMember` attribute), 103
- `kwargs` (`telegram.utils.promise.Promise` attribute), 53
- ## L
- `label` (`telegram.LabeledPrice` attribute), 181, 321
- `LabeledPrice` (class in `telegram`), 181
- `language_code` (`telegram.User` attribute), 140, 311
- `last_error_date` (`telegram.WebhookInfo` attribute), 147, 316
- `last_error_message` (`telegram.WebhookInfo` attribute), 147, 317
- `last_name` (`telegram.Bot` attribute), 71
- `last_name` (`telegram.Chat` attribute), 95, 238
- `last_name` (`telegram.Contact` attribute), 105, 249
- `last_name` (`telegram.InlineQueryResultContact` attribute), 164, 267
- `last_name` (`telegram.InputContactMessageContent` attribute), 179, 280
- `last_name` (`telegram.PersonalDetails` attribute), 195, 337
- `last_name` (`telegram.User` attribute), 139, 311
- `latitude` (`telegram.InlineQueryResultLocation` attribute), 169, 271
- `latitude` (`telegram.InlineQueryResultVenue` attribute), 173, 275
- `latitude` (`telegram.InputLocationMessageContent` attribute), 178, 281
- `latitude` (`telegram.InputVenueMessageContent` attribute), 178, 282
- `latitude` (`telegram.Location` attribute), 118, 284
- `leave()` (`telegram.Chat` method), 98
- `leave_chat()` (`telegram.Bot` method), 72
- `leaveChat()` (`telegram.Bot` method), 71
- `LEFT` (`telegram.ChatMember` attribute), 103
- `left_chat_member` (`telegram.ext.filters.Filters` attribute), 12
- `left_chat_member` (`telegram.Message` attribute), 120, 291
- `length` (`telegram.MessageEntity` attribute), 131, 302
- `length` (`telegram.VideoNote` attribute), 144, 320
- `link` (`telegram.Chat` attribute), 98
- `link` (`telegram.Message` attribute), 126
- `link` (`telegram.User` attribute), 140
- `live_period` (`telegram.InlineQueryResultLocation` attribute), 169, 271
- `Location` (class in `telegram`), 118
- `location` (`telegram.ChosenInlineResult` attribute), 180, 246
- `location` (`telegram.ext.filters.Filters` attribute), 11
- `location` (`telegram.InlineQuery` attribute), 150, 254
- `location` (`telegram.Message` attribute), 120, 291
- `location` (`telegram.Venue` attribute), 142, 314
- `longitude` (`telegram.InlineQueryResultLocation` attribute), 169, 271
- `longitude` (`telegram.InlineQueryResultVenue` attribute), 173, 275
- `longitude` (`telegram.InputLocationMessageContent` attribute), 178, 281
- `longitude` (`telegram.InputVenueMessageContent` attribute), 178, 282
- `longitude` (`telegram.Location` attribute), 118, 284
- ## M
- `MARKDOWN` (`telegram.ParseMode` attribute), 133

- `mask_position` (*telegram.Sticker* attribute), 148, 308
 - `MaskPosition` (class in *telegram*), 149
 - `match` (*telegram.ext.CallbackContext* attribute), 21
 - `matches` (*telegram.ext.CallbackContext* attribute), 21
 - `MAX_CAPTION_LENGTH` (in module *telegram.constants*), 104
 - `max_connections` (*telegram.WebhookInfo* attribute), 147, 317
 - `MAX_FILESIZE_DOWNLOAD` (in module *telegram.constants*), 104
 - `MAX_FILESIZE_UPLOAD` (in module *telegram.constants*), 104
 - `MAX_INLINE_QUERY_RESULTS` (in module *telegram.constants*), 104
 - `MAX_MESSAGE_ENTITIES` (in module *telegram.constants*), 104
 - `MAX_MESSAGE_LENGTH` (in module *telegram.constants*), 104
 - `MAX_MESSAGES_PER_MINUTE_PER_GROUP` (in module *telegram.constants*), 104
 - `MAX_MESSAGES_PER_SECOND` (in module *telegram.constants*), 104
 - `MAX_MESSAGES_PER_SECOND_PER_CHAT` (in module *telegram.constants*), 104
 - `media` (*telegram.InputMediaAnimation* attribute), 112, 339
 - `media` (*telegram.InputMediaAudio* attribute), 113, 340
 - `media` (*telegram.InputMediaDocument* attribute), 114, 341
 - `media` (*telegram.InputMediaPhoto* attribute), 115, 329
 - `media` (*telegram.InputMediaVideo* attribute), 116, 330, 338
 - `media_group_id` (*telegram.Message* attribute), 119, 290
 - `MEMBER` (*telegram.ChatMember* attribute), 103
 - `MENTION` (*telegram.MessageEntity* attribute), 132
 - `mention_html()` (in module *telegram.utils.helpers*), 52
 - `mention_html()` (*telegram.User* method), 140
 - `mention_markdown()` (in module *telegram.utils.helpers*), 52
 - `mention_markdown()` (*telegram.User* method), 140
 - `MergedFilter` (class in *telegram.ext.filters*), 14
 - `Message` (class in *telegram*), 118
 - `message` (*telegram.CallbackQuery* attribute), 93, 246
 - `message` (*telegram.ext.filters.Filters* attribute), 13
 - `message` (*telegram.PassportElementError* attribute), 189, 331
 - `message` (*telegram.PassportElementErrorDataField* attribute), 192, 333
 - `message` (*telegram.PassportElementErrorFile* attribute), 190, 331, 334
 - `message` (*telegram.PassportElementErrorFiles* attribute), 192, 333
 - `message` (*telegram.PassportElementErrorFrontSide* attribute), 191, 332
 - `message` (*telegram.PassportElementErrorReverseSide* attribute), 191, 332
 - `message` (*telegram.PassportElementErrorSelfie* attribute), 342
 - `message` (*telegram.PassportElementErrorTranslationFile* attribute), 343
 - `message` (*telegram.PassportElementErrorTranslationFiles* attribute), 343
 - `message` (*telegram.PassportElementErrorUnspecified* attribute), 344
 - `message` (*telegram.Update* attribute), 138, 309
 - `message_id` (*telegram.Message* attribute), 118, 289
 - `message_text` (*telegram.InputTextMessageContent* attribute), 177, 282
 - `message_updates` (*telegram.ext.MessageHandler* attribute), 35
 - `MessageEntity` (class in *telegram*), 131
 - `MessageHandler` (class in *telegram.ext*), 34
 - `MessageQueue` (class in *telegram.ext*), 18
 - `messages` (*telegram.ext.filters.Filters* attribute), 13
 - `middle_name` (*telegram.PersonalDetails* attribute), 195, 336, 337
 - `migrate` (*telegram.ext.filters.Filters* attribute), 12
 - `migrate_from_chat_id` (*telegram.Message* attribute), 121, 291
 - `migrate_to_chat_id` (*telegram.Message* attribute), 121, 291
 - `mime_type` (*telegram.Animation* attribute), 55, 318
 - `mime_type` (*telegram.Audio* attribute), 56, 201
 - `mime_type` (*telegram.Document* attribute), 105, 249
 - `mime_type` (*telegram.InlineQueryResultDocument* attribute), 165, 268
 - `mime_type` (*telegram.InlineQueryResultVideo* attribute), 175, 277
 - `mime_type` (*telegram.Video* attribute), 144, 315
 - `mime_type` (*telegram.Voice* attribute), 146, 315
 - `MOUTH` (*telegram.MaskPosition* attribute), 150
 - `mpeg4_duration` (*telegram.InlineQueryResultMpeg4Gif* attribute), 170, 273
 - `mpeg4_file_id` (*telegram.InlineQueryResultCachedMpeg4Gif* attribute), 158, 262
 - `mpeg4_height` (*telegram.InlineQueryResultMpeg4Gif* attribute), 170, 272
 - `mpeg4_url` (*telegram.InlineQueryResultMpeg4Gif* attribute), 170, 272
 - `mpeg4_width` (*telegram.InlineQueryResultMpeg4Gif* attribute), 170, 272
- ## N
- `name` (*telegram.Bot* attribute), 72
 - `name` (*telegram.ext.ConversationHandler* attribute), 28
 - `name` (*telegram.ext.DelayQueue* attribute), 19

`name` (*telegram.ext.filters.BaseFilter* attribute), 9
`name` (*telegram.ext.Job* attribute), 15
`name` (*telegram.OrderInfo* attribute), 183, 325
`name` (*telegram.StickerSet* attribute), 149, 327
`name` (*telegram.User* attribute), 140
`NetworkError`, 106
`new_chat_members` (*telegram.ext.filters.Filters* attribute), 12
`new_chat_members` (*telegram.Message* attribute), 120, 291
`new_chat_photo` (*telegram.ext.filters.Filters* attribute), 12
`new_chat_photo` (*telegram.Message* attribute), 120, 291
`new_chat_title` (*telegram.ext.filters.Filters* attribute), 12
`new_chat_title` (*telegram.Message* attribute), 120, 291
`nonce` (*telegram.Credentials* attribute), 193, 334

O

`offset` (*telegram.InlineQuery* attribute), 151, 254
`offset` (*telegram.MessageEntity* attribute), 131, 302
`on_flush` (*telegram.ext.PicklePersistence* attribute), 49
`one_time_keyboard` (*telegram.ReplyKeyboardMarkup* attribute), 135, 305
`order_info` (*telegram.PreCheckoutQuery* attribute), 186, 324
`order_info` (*telegram.SuccessfulPayment* attribute), 184, 322
`OrderInfo` (class in *telegram*), 183

P

`parse_caption_entities()` (*telegram.Message* method), 126
`parse_caption_entity()` (*telegram.Message* method), 126
`parse_entities()` (*telegram.Message* method), 127
`parse_entity()` (*telegram.Message* method), 127
`parse_mode` (*telegram.InlineQueryResultAudio* attribute), 154, 257
`parse_mode` (*telegram.InlineQueryResultCachedAudio* attribute), 155, 259
`parse_mode` (*telegram.InlineQueryResultCachedDocument* attribute), 156, 260
`parse_mode` (*telegram.InlineQueryResultCachedGif* attribute), 157, 261
`parse_mode` (*telegram.InlineQueryResultCachedMpeg4Gif* attribute), 158, 262
`parse_mode` (*telegram.InlineQueryResultCachedPhoto* attribute), 160, 263
`parse_mode` (*telegram.InlineQueryResultCachedVideo* attribute), 162, 265
`parse_mode` (*telegram.InlineQueryResultCachedVoice* attribute), 163, 266

`parse_mode` (*telegram.InlineQueryResultDocument* attribute), 165, 268
`parse_mode` (*telegram.InlineQueryResultGif* attribute), 168, 270
`parse_mode` (*telegram.InlineQueryResultMpeg4Gif* attribute), 170, 273
`parse_mode` (*telegram.InlineQueryResultPhoto* attribute), 172, 274
`parse_mode` (*telegram.InlineQueryResultVideo* attribute), 175, 277
`parse_mode` (*telegram.InlineQueryResultVoice* attribute), 176, 279
`parse_mode` (*telegram.InputMediaAnimation* attribute), 112, 339
`parse_mode` (*telegram.InputMediaAudio* attribute), 113, 340
`parse_mode` (*telegram.InputMediaDocument* attribute), 114, 342
`parse_mode` (*telegram.InputMediaPhoto* attribute), 115, 329
`parse_mode` (*telegram.InputMediaVideo* attribute), 116, 330, 338
`parse_mode` (*telegram.InputTextMessageContent* attribute), 177, 282
`parse_text_entities()` (*telegram.Game* method), 188
`parse_text_entity()` (*telegram.Game* method), 188
`ParseMode` (class in *telegram*), 133
`pass_args` (*telegram.ext.CommandHandler* attribute), 30
`pass_args` (*telegram.ext.PrefixHandler* attribute), 38
`pass_args` (*telegram.ext.StringCommandHandler* attribute), 43
`pass_chat_data` (*telegram.ext.CallbackQueryHandler* attribute), 24
`pass_chat_data` (*telegram.ext.ChosenInlineResultHandler* attribute), 26
`pass_chat_data` (*telegram.ext.CommandHandler* attribute), 30
`pass_chat_data` (*telegram.ext.Handler* attribute), 22
`pass_chat_data` (*telegram.ext.InlineQueryHandler* attribute), 33
`pass_chat_data` (*telegram.ext.MessageHandler* attribute), 34
`pass_chat_data` (*telegram.ext.PreCheckoutQueryHandler* attribute), 36
`pass_chat_data` (*telegram.ext.PrefixHandler* attribute), 38
`pass_chat_data` (*telegram.ext.RegexHandler* attribute), 40
`pass_chat_data` (*telegram.ext.RegexHandler* attribute), 40

<code>gram.ext.ShippingQueryHandler</code> attribute), 42	<code>gram.ext.ChosenInlineResultHandler</code> attribute), 26
<code>pass_groupdict</code> (<code>telegram.ext.CallbackQueryHandler</code> attribute), 24	<code>pass_update_queue</code> (<code>telegram.ext.CommandHandler</code> attribute), 30
<code>pass_groupdict</code> (<code>telegram.ext.InlineQueryHandler</code> attribute), 32	<code>pass_update_queue</code> (<code>telegram.ext.Handler</code> attribute), 22
<code>pass_groupdict</code> (<code>telegram.ext.RegexHandler</code> attribute), 40	<code>pass_update_queue</code> (<code>telegram.ext.InlineQueryHandler</code> attribute), 32
<code>pass_groupdict</code> (<code>telegram.ext.StringRegexHandler</code> attribute), 44	<code>pass_update_queue</code> (<code>telegram.ext.MessageHandler</code> attribute), 34
<code>pass_groups</code> (<code>telegram.ext.CallbackQueryHandler</code> attribute), 24	<code>pass_update_queue</code> (<code>telegram.ext.PreCheckoutQueryHandler</code> attribute), 36
<code>pass_groups</code> (<code>telegram.ext.InlineQueryHandler</code> attribute), 32	<code>pass_update_queue</code> (<code>telegram.ext.PrefixHandler</code> attribute), 38
<code>pass_groups</code> (<code>telegram.ext.RegexHandler</code> attribute), 40	<code>pass_update_queue</code> (<code>telegram.ext.RegexHandler</code> attribute), 40
<code>pass_groups</code> (<code>telegram.ext.StringRegexHandler</code> attribute), 44	<code>pass_update_queue</code> (<code>telegram.ext.ShippingQueryHandler</code> attribute), 42
<code>pass_job_queue</code> (<code>telegram.ext.CallbackQueryHandler</code> attribute), 24	<code>pass_update_queue</code> (<code>telegram.ext.StringCommandHandler</code> attribute), 43
<code>pass_job_queue</code> (<code>telegram.ext.ChosenInlineResultHandler</code> attribute), 26	<code>pass_update_queue</code> (<code>telegram.ext.StringRegexHandler</code> attribute), 45
<code>pass_job_queue</code> (<code>telegram.ext.CommandHandler</code> attribute), 30	<code>pass_update_queue</code> (<code>telegram.ext.TypeHandler</code> attribute), 46
<code>pass_job_queue</code> (<code>telegram.ext.Handler</code> attribute), 22	<code>pass_user_data</code> (<code>telegram.ext.CallbackQueryHandler</code> attribute), 24
<code>pass_job_queue</code> (<code>telegram.ext.InlineQueryHandler</code> attribute), 32	<code>pass_user_data</code> (<code>telegram.ext.ChosenInlineResultHandler</code> attribute), 26
<code>pass_job_queue</code> (<code>telegram.ext.MessageHandler</code> attribute), 34	<code>pass_user_data</code> (<code>telegram.ext.CommandHandler</code> attribute), 30
<code>pass_job_queue</code> (<code>telegram.ext.PreCheckoutQueryHandler</code> attribute), 36	<code>pass_user_data</code> (<code>telegram.ext.Handler</code> attribute), 22
<code>pass_job_queue</code> (<code>telegram.ext.PrefixHandler</code> attribute), 38	<code>pass_user_data</code> (<code>telegram.ext.InlineQueryHandler</code> attribute), 32
<code>pass_job_queue</code> (<code>telegram.ext.RegexHandler</code> attribute), 40	<code>pass_user_data</code> (<code>telegram.ext.MessageHandler</code> attribute), 34
<code>pass_job_queue</code> (<code>telegram.ext.ShippingQueryHandler</code> attribute), 42	<code>pass_user_data</code> (<code>telegram.ext.PreCheckoutQueryHandler</code> attribute), 36
<code>pass_job_queue</code> (<code>telegram.ext.StringCommandHandler</code> attribute), 43	<code>pass_user_data</code> (<code>telegram.ext.PrefixHandler</code> attribute), 38
<code>pass_job_queue</code> (<code>telegram.ext.StringRegexHandler</code> attribute), 45	<code>pass_user_data</code> (<code>telegram.ext.RegexHandler</code> attribute), 40
<code>pass_job_queue</code> (<code>telegram.ext.TypeHandler</code> attribute), 46	<code>pass_user_data</code> (<code>telegram.ext.ShippingQueryHandler</code> attribute), 42
<code>pass_update_queue</code> (<code>telegram.ext.CallbackQueryHandler</code> attribute), 24	<code>passport</code> (<code>telegram.SecureData</code> attribute), 193, 335
<code>pass_update_queue</code> (<code>telegram.ext.ChosenInlineResultHandler</code> attribute), 26	<code>passport_data</code> (<code>telegram.ext.filters.Filters</code> attribute), 11

- `passport_data` (*telegram.Message* attribute), 121, 292
- `passport_registration` (*telegram.SecureData* attribute), 194, 336
- `PassportData` (class in *telegram*), 196
- `PassportElementError` (class in *telegram*), 189
- `PassportElementErrorDataField` (class in *telegram*), 192
- `PassportElementErrorFile` (class in *telegram*), 190
- `PassportElementErrorFiles` (class in *telegram*), 191
- `PassportElementErrorFrontSide` (class in *telegram*), 191
- `PassportElementErrorReverseSide` (class in *telegram*), 190
- `PassportFile` (class in *telegram*), 197
- `pattern` (*telegram.ext.CallbackQueryHandler* attribute), 24
- `pattern` (*telegram.ext.InlineQueryHandler* attribute), 32
- `pattern` (*telegram.ext.RegexHandler* attribute), 40
- `pattern` (*telegram.ext.StringRegexHandler* attribute), 44
- `pay` (*telegram.InlineKeyboardButton* attribute), 109, 252
- `pending_update_count` (*telegram.WebhookInfo* attribute), 147, 316
- `per_chat` (*telegram.ext.ConversationHandler* attribute), 28
- `per_message` (*telegram.ext.ConversationHandler* attribute), 28
- `per_user` (*telegram.ext.ConversationHandler* attribute), 28
- `performer` (*telegram.Audio* attribute), 56, 201
- `performer` (*telegram.InlineQueryResultAudio* attribute), 154, 257
- `performer` (*telegram.InputMediaAudio* attribute), 113, 341
- `persistence` (*telegram.ext.ConversationHandler* attribute), 29
- `persistence` (*telegram.ext.Dispatcher* attribute), 6
- `persistence` (*telegram.ext.Updater* attribute), 4
- `persistent` (*telegram.ext.ConversationHandler* attribute), 28
- `personal_details` (*telegram.SecureData* attribute), 193, 335
- `PersonalDetails` (class in *telegram*), 195
- `phone_number` (*telegram.Contact* attribute), 104, 248
- `phone_number` (*telegram.EncryptedPassportElement* attribute), 198, 286
- `phone_number` (*telegram.InlineQueryResultContact* attribute), 164, 267
- `phone_number` (*telegram.InputContactMessageContent* attribute), 179, 280
- `PHONE_NUMBER` (*telegram.MessageEntity* attribute), 132
- `phone_number` (*telegram.OrderInfo* attribute), 183, 325
- `photo` (*telegram.Chat* attribute), 96, 238
- `photo` (*telegram.ext.filters.Filters* attribute), 11
- `photo` (*telegram.Game* attribute), 187, 318
- `photo` (*telegram.Message* attribute), 120, 290
- `photo_file_id` (*telegram.InlineQueryResultCachedPhoto* attribute), 159, 263
- `photo_height` (*telegram.InlineQueryResultPhoto* attribute), 172, 274
- `photo_url` (*telegram.InlineQueryResultPhoto* attribute), 171, 274
- `photo_width` (*telegram.InlineQueryResultPhoto* attribute), 172, 274
- `photos` (*telegram.UserProfilePhotos* attribute), 142, 313
- `PhotoSize` (class in *telegram*), 133
- `PicklePersistence` (class in *telegram.ext*), 48
- `pin_chat_message()` (*telegram.Bot* method), 72
- `pinChatMessage()` (*telegram.Bot* method), 72
- `pinned_message` (*telegram.Chat* attribute), 96, 238
- `pinned_message` (*telegram.ext.filters.Filters* attribute), 12
- `pinned_message` (*telegram.Message* attribute), 121, 291
- `point` (*telegram.MaskPosition* attribute), 149, 328
- `pooled_function` (*telegram.utils.promise.Promise* attribute), 53
- `position` (*telegram.GameHighScore* attribute), 189, 319
- `post()` (*telegram.utils.request.Request* method), 54
- `post_code` (*telegram.ResidentialAddress* attribute), 196, 338
- `post_code` (*telegram.ShippingAddress* attribute), 182, 323
- `PRE` (*telegram.MessageEntity* attribute), 132
- `pre_checkout_query` (*telegram.Update* attribute), 138, 309
- `PreCheckoutQuery` (class in *telegram*), 186
- `PreCheckoutQueryHandler` (class in *telegram.ext*), 36
- `prefix` (*telegram.ext.PrefixHandler* attribute), 38
- `PrefixHandler` (class in *telegram.ext*), 37
- `prices` (*telegram.ShippingOption* attribute), 183, 322
- `PRIVATE` (*telegram.Chat* attribute), 97
- `private` (*telegram.ext.filters.Filters* attribute), 11
- `process_update()` (*telegram.ext.Dispatcher* method), 7
- `Promise` (class in *telegram.utils.promise*), 53
- `promote_chat_member()` (*telegram.Bot* method), 72
- `promoteChatMember()` (*telegram.Bot* method), 72
- `provider_payment_charge_id` (*telegram.SuccessfulPayment* attribute), 184, 322

Q

query (*telegram.ChosenInlineResult* attribute), 180, 246

query (*telegram.InlineQuery* attribute), 151, 254

R

RECORD_AUDIO (*telegram.ChatAction* attribute), 100

RECORD_VIDEO (*telegram.ChatAction* attribute), 100

RECORD_VIDEO_NOTE (*telegram.ChatAction* attribute), 100

RegexHandler (class in *telegram.ext*), 40

remove_error_handler() (*telegram.ext.Dispatcher* method), 7

remove_handler() (*telegram.ext.Dispatcher* method), 8

remove_keyboard (*telegram.ReplyKeyboardRemove* attribute), 134, 304

removed (*telegram.ext.Job* attribute), 15

rental_agreement (*telegram.SecureData* attribute), 194, 335

repeat (*telegram.ext.Job* attribute), 15

reply (*telegram.ext.filters.Filters* attribute), 12

reply_animation() (*telegram.Message* method), 127

reply_audio() (*telegram.Message* method), 128

reply_contact() (*telegram.Message* method), 128

reply_document() (*telegram.Message* method), 128

reply_html() (*telegram.Message* method), 128

reply_location() (*telegram.Message* method), 128

reply_markdown() (*telegram.Message* method), 129

reply_markup (*telegram.InlineQueryResultArticle* attribute), 152, 256

reply_markup (*telegram.InlineQueryResultAudio* attribute), 154, 258

reply_markup (*telegram.InlineQueryResultCachedAudio* attribute), 155, 259

reply_markup (*telegram.InlineQueryResultCachedDocument* attribute), 156, 260

reply_markup (*telegram.InlineQueryResultCachedGif* attribute), 157, 261

reply_markup (*telegram.InlineQueryResultCachedMpeg4Gif* attribute), 158, 262

reply_markup (*telegram.InlineQueryResultCachedPhoto* attribute), 160, 263

reply_markup (*telegram.InlineQueryResultCachedSticker* attribute), 161, 264

reply_markup (*telegram.InlineQueryResultCachedVideo* attribute), 162, 265

reply_markup (*telegram.InlineQueryResultCachedVoice* attribute), 163, 266

reply_markup (*telegram.InlineQueryResultContact* attribute), 164, 267

reply_markup (*telegram.InlineQueryResultDocument* attribute), 165, 268

reply_markup (*telegram.InlineQueryResultGame* attribute), 167, 280

reply_markup (*telegram.InlineQueryResultGif* attribute), 168, 270

reply_markup (*telegram.InlineQueryResultLocation* attribute), 169, 271

reply_markup (*telegram.InlineQueryResultMpeg4Gif* attribute), 171, 273

reply_markup (*telegram.InlineQueryResultPhoto* attribute), 172, 274

reply_markup (*telegram.InlineQueryResultVenue* attribute), 173, 276

reply_markup (*telegram.InlineQueryResultVideo* attribute), 175, 278

reply_markup (*telegram.InlineQueryResultVoice* attribute), 177, 279

reply_media_group() (*telegram.Message* method), 129

reply_photo() (*telegram.Message* method), 129

reply_sticker() (*telegram.Message* method), 129

reply_text() (*telegram.Message* method), 129

reply_to_message (*telegram.Message* attribute), 119, 289

reply_venue() (*telegram.Message* method), 130

reply_video() (*telegram.Message* method), 130

reply_video_note() (*telegram.Message* method), 130

reply_voice() (*telegram.Message* method), 130

ReplyKeyboardMarkup (class in *telegram*), 135

ReplyKeyboardRemove (class in *telegram*), 134

ReplyMarkup (class in *telegram*), 137

Request (class in *telegram.utils.request*), 54

request_contact (*telegram.KeyboardButton* attribute), 117, 283

request_location (*telegram.KeyboardButton* attribute), 117, 283

residence_country_code (*telegram.PersonalDetails* attribute), 195, 337

ResidentialAddress (class in *telegram*), 196

resize_keyboard (*telegram.ReplyKeyboardMarkup* attribute), 135, 305

restrict_chat_member() (*telegram.Bot*

method), 73
restrictChatMember() (*telegram.Bot method*), 73
RESTRICTED (*telegram.ChatMember attribute*), 103
result() (*telegram.utils.promise.Promise method*), 53
result_id (*telegram.ChosenInlineResult attribute*), 180, 245
retrieve() (*telegram.utils.request.Request method*), 54
RetryAfter, 106
reverse_side (*telegram.EncryptedPassportElement attribute*), 199, 286
run() (*telegram.ext.DelayQueue method*), 20
run() (*telegram.ext.Job method*), 16
run() (*telegram.utils.promise.Promise method*), 53
run_async() (*telegram.ext.Dispatcher method*), 8
run_daily() (*telegram.ext.JobQueue method*), 16
run_once() (*telegram.ext.JobQueue method*), 16
run_repeating() (*telegram.ext.JobQueue method*), 17
running (*telegram.ext.Dispatcher attribute*), 8
running (*telegram.ext.Updater attribute*), 4

S

scale (*telegram.MaskPosition attribute*), 149, 328
schedule_removal() (*telegram.ext.Job method*), 16
score (*telegram.GameHighScore attribute*), 189, 320
secret (*telegram.DataCredentials attribute*), 193, 335
secret (*telegram.EncryptedCredentials attribute*), 200, 284
secret (*telegram.FileCredentials attribute*), 194, 336
secure_data (*telegram.Credentials attribute*), 193, 334
SecureData (*class in telegram*), 193
selective (*telegram.ForceReply attribute*), 108, 251
selective (*telegram.ReplyKeyboardMarkup attribute*), 135, 305
selective (*telegram.ReplyKeyboardRemove attribute*), 134, 304
selfie (*telegram.EncryptedPassportElement attribute*), 199, 286
send_action() (*telegram.Chat method*), 98
send_animation() (*telegram.Bot method*), 75
send_animation() (*telegram.Chat method*), 98
send_animation() (*telegram.User method*), 141
send_audio() (*telegram.Bot method*), 76
send_audio() (*telegram.Chat method*), 98
send_audio() (*telegram.User method*), 141
send_chat_action() (*telegram.Bot method*), 77
send_contact() (*telegram.Bot method*), 77
send_document() (*telegram.Bot method*), 78
send_document() (*telegram.Chat method*), 98
send_document() (*telegram.User method*), 141
send_game() (*telegram.Bot method*), 79

send_invoice() (*telegram.Bot method*), 79
send_location() (*telegram.Bot method*), 80
send_media_group() (*telegram.Bot method*), 81
send_message() (*telegram.Bot method*), 82
send_message() (*telegram.Chat method*), 98
send_message() (*telegram.User method*), 141
send_photo() (*telegram.Bot method*), 82
send_photo() (*telegram.Chat method*), 99
send_photo() (*telegram.User method*), 141
send_sticker() (*telegram.Bot method*), 83
send_sticker() (*telegram.Chat method*), 99
send_sticker() (*telegram.User method*), 141
send_venue() (*telegram.Bot method*), 83
send_video() (*telegram.Bot method*), 84
send_video() (*telegram.Chat method*), 99
send_video() (*telegram.User method*), 141
send_video_note() (*telegram.Bot method*), 85
send_video_note() (*telegram.Chat method*), 99
send_video_note() (*telegram.User method*), 142
send_voice() (*telegram.Bot method*), 86
send_voice() (*telegram.Chat method*), 99
send_voice() (*telegram.User method*), 142
sendAnimation() (*telegram.Bot method*), 74
sendAudio() (*telegram.Bot method*), 74
sendChatAction() (*telegram.Bot method*), 74
sendContact() (*telegram.Bot method*), 74
sendDocument() (*telegram.Bot method*), 74
sendGame() (*telegram.Bot method*), 74
sendInvoice() (*telegram.Bot method*), 74
sendLocation() (*telegram.Bot method*), 74
sendMediaGroup() (*telegram.Bot method*), 75
sendMessage() (*telegram.Bot method*), 75
sendPhoto() (*telegram.Bot method*), 75
sendSticker() (*telegram.Bot method*), 75
sendVenue() (*telegram.Bot method*), 75
sendVideo() (*telegram.Bot method*), 75
sendVideoNote() (*telegram.Bot method*), 75
sendVoice() (*telegram.Bot method*), 75
set_chat_description() (*telegram.Bot method*), 87
set_chat_photo() (*telegram.Bot method*), 87
set_chat_sticker_set() (*telegram.Bot method*), 88
set_chat_title() (*telegram.Bot method*), 88
set_game_score() (*telegram.Bot method*), 89
set_name (*telegram.Sticker attribute*), 148, 308
set_passport_data_errors() (*telegram.Bot method*), 89
set_sticker_position_in_set() (*telegram.Bot method*), 90
set_webhook() (*telegram.Bot method*), 90
setChatDescription() (*telegram.Bot method*), 87
setChatPhoto() (*telegram.Bot method*), 87
setChatStickerSet() (*telegram.Bot method*), 87
setChatTitle() (*telegram.Bot method*), 87
setGameScore() (*telegram.Bot method*), 87

- setPassportDataErrors() (*telegram.Bot method*), 87
 setStickerPositionInSet() (*telegram.Bot method*), 87
 setWebhook() (*telegram.Bot method*), 87
 shipping_address (*telegram.OrderInfo attribute*), 183, 325
 shipping_address (*telegram.ShippingQuery attribute*), 185, 326
 shipping_option_id (*telegram.PreCheckoutQuery attribute*), 186, 324
 shipping_option_id (*telegram.SuccessfulPayment attribute*), 184, 321
 shipping_query (*telegram.Update attribute*), 138, 309
 ShippingAddress (*class in telegram*), 182
 ShippingOption (*class in telegram*), 183
 ShippingQuery (*class in telegram*), 185
 ShippingQueryHandler (*class in telegram.ext*), 41
 single_file (*telegram.ext.PicklePersistence attribute*), 49
 small_file_id (*telegram.ChatPhoto attribute*), 103, 327
 source (*telegram.PassportElementError attribute*), 189, 331
 start() (*telegram.ext.Dispatcher method*), 8
 start() (*telegram.ext.JobQueue method*), 18
 start() (*telegram.ext.MessageQueue method*), 19
 start_parameter (*telegram.Invoice attribute*), 181, 325
 start_polling() (*telegram.ext.Updater method*), 4
 start_webhook() (*telegram.ext.Updater method*), 5
 state (*telegram.ResidentialAddress attribute*), 196, 337
 state (*telegram.ShippingAddress attribute*), 182, 323
 states (*telegram.ext.ConversationHandler attribute*), 27
 status (*telegram.ChatMember attribute*), 101, 242
 status_update (*telegram.ext.filters.Filters attribute*), 12
 Sticker (*class in telegram*), 147
 sticker (*telegram.ext.filters.Filters attribute*), 12
 sticker (*telegram.Message attribute*), 120, 290
 sticker_file_id (*telegram.InlineQueryResultCachedSticker attribute*), 160, 264
 sticker_set_name (*telegram.Chat attribute*), 96, 238
 stickers (*telegram.StickerSet attribute*), 149, 328
 StickerSet (*class in telegram*), 149
 stop() (*telegram.ext.DelayQueue method*), 20
 stop() (*telegram.ext.Dispatcher method*), 8
 stop() (*telegram.ext.JobQueue method*), 18
 stop() (*telegram.ext.MessageQueue method*), 19
 stop() (*telegram.ext.Updater method*), 5
 stop_message_live_location() (*telegram.Bot method*), 91
 stopMessageLiveLocation() (*telegram.Bot method*), 91
 store_chat_data (*telegram.ext.BasePersistence attribute*), 47
 store_chat_data (*telegram.ext.DictPersistence attribute*), 50
 store_chat_data (*telegram.ext.PicklePersistence attribute*), 48
 store_user_data (*telegram.ext.BasePersistence attribute*), 47
 store_user_data (*telegram.ext.DictPersistence attribute*), 50
 store_user_data (*telegram.ext.PicklePersistence attribute*), 48
 street_line1 (*telegram.ResidentialAddress attribute*), 196, 337
 street_line1 (*telegram.ShippingAddress attribute*), 182, 323
 street_line2 (*telegram.ResidentialAddress attribute*), 196, 337
 street_line2 (*telegram.ShippingAddress attribute*), 182, 323
 strict (*telegram.ext.TypeHandler attribute*), 46
 StringCommandHandler (*class in telegram.ext*), 43
 StringRegexHandler (*class in telegram.ext*), 44
 successful_payment (*telegram.ext.filters.Filters attribute*), 12
 successful_payment (*telegram.Message attribute*), 121, 292
 SuccessfulPayment (*class in telegram*), 184
 SUPERGROUP (*telegram.Chat attribute*), 97
 supergroup_chat_created (*telegram.Message attribute*), 121, 291
 SUPPORTED_WEBHOOK_PORTS (*in module telegram.constants*), 104
 supports_streaming (*telegram.InputMediaVideo attribute*), 116, 330, 338
 switch_inline_query (*telegram.InlineKeyboardButton attribute*), 109, 252
 switch_inline_query_current_chat (*telegram.InlineKeyboardButton attribute*), 109, 252
- ## T
- telegram.constants (*module*), 104
 telegram.error (*module*), 106
 telegram.ext.filters (*module*), 8
 telegram.utils.helpers (*module*), 52
 telegram_payment_charge_id (*telegram.SuccessfulPayment attribute*), 184, 322
 TelegramError, 106

TelegramObject (class in telegram), 137

temporary_registration (telegram.SecureData attribute), 194, 336

text (telegram.ext.filters.Filters attribute), 13

text (telegram.Game attribute), 187, 318

text (telegram.InlineKeyboardButton attribute), 109, 252

text (telegram.KeyboardButton attribute), 117, 283

text (telegram.Message attribute), 119, 290

text_entities (telegram.Game attribute), 188, 318

text_html (telegram.Message attribute), 130

text_html_urled (telegram.Message attribute), 131

TEXT_LINK (telegram.MessageEntity attribute), 133

text_markdown (telegram.Message attribute), 131

text_markdown_urled (telegram.Message attribute), 131

TEXT_MENTION (telegram.MessageEntity attribute), 133

thumb (telegram.Animation attribute), 55, 317

thumb (telegram.Audio attribute), 56, 201

thumb (telegram.Document attribute), 105, 249

thumb (telegram.InputMediaAnimation attribute), 112, 339

thumb (telegram.InputMediaAudio attribute), 113, 341

thumb (telegram.InputMediaDocument attribute), 114, 342

thumb (telegram.InputMediaVideo attribute), 116, 330, 338

thumb (telegram.Sticker attribute), 148, 307

thumb (telegram.Video attribute), 143, 315

thumb (telegram.VideoNote attribute), 145, 320

thumb_height (telegram.InlineQueryResultArticle attribute), 153, 256

thumb_height (telegram.InlineQueryResultContact attribute), 164, 267

thumb_height (telegram.InlineQueryResultDocument attribute), 166, 269

thumb_height (telegram.InlineQueryResultLocation attribute), 169, 272

thumb_height (telegram.InlineQueryResultVenue attribute), 174, 276

thumb_url (telegram.InlineQueryResultArticle attribute), 152, 256

thumb_url (telegram.InlineQueryResultContact attribute), 164, 267

thumb_url (telegram.InlineQueryResultDocument attribute), 166, 269

thumb_url (telegram.InlineQueryResultGif attribute), 167, 270

thumb_url (telegram.InlineQueryResultLocation attribute), 169, 271

thumb_url (telegram.InlineQueryResultMpeg4Gif attribute), 170, 273

thumb_url (telegram.InlineQueryResultPhoto attribute), 171, 274

thumb_url (telegram.InlineQueryResultVenue attribute), 173, 276

thumb_url (telegram.InlineQueryResultVideo attribute), 175, 277

thumb_width (telegram.InlineQueryResultArticle attribute), 153, 256

thumb_width (telegram.InlineQueryResultContact attribute), 164, 267

thumb_width (telegram.InlineQueryResultDocument attribute), 166, 269

thumb_width (telegram.InlineQueryResultLocation attribute), 169, 271

thumb_width (telegram.InlineQueryResultVenue attribute), 174, 276

tick() (telegram.ext.JobQueue method), 18

time_limit (telegram.ext.DelayQueue attribute), 19

TimedOut, 107

TIMEOUT (telegram.ext.ConversationHandler attribute), 29

title (telegram.Audio attribute), 56, 201

title (telegram.Chat attribute), 95, 238

title (telegram.Game attribute), 187, 318

title (telegram.InlineQueryResultArticle attribute), 152, 256

title (telegram.InlineQueryResultAudio attribute), 153, 257

title (telegram.InlineQueryResultCachedDocument attribute), 156, 259

title (telegram.InlineQueryResultCachedGif attribute), 157, 261

title (telegram.InlineQueryResultCachedMpeg4Gif attribute), 158, 262

title (telegram.InlineQueryResultCachedPhoto attribute), 159, 263

title (telegram.InlineQueryResultCachedVideo attribute), 161, 265

title (telegram.InlineQueryResultCachedVoice attribute), 163, 266

title (telegram.InlineQueryResultDocument attribute), 165, 268

title (telegram.InlineQueryResultGif attribute), 167, 270

title (telegram.InlineQueryResultLocation attribute), 169, 271

title (telegram.InlineQueryResultMpeg4Gif attribute), 170, 273

title (telegram.InlineQueryResultPhoto attribute), 172, 274

title (telegram.InlineQueryResultVenue attribute), 173, 275

title (telegram.InlineQueryResultVideo attribute), 175, 277

title (telegram.InlineQueryResultVoice attribute), 176, 279

title (telegram.InputMediaAudio attribute), 113,

- 341
- `title` (*telegram.InputVenueMessageContent* attribute), 178, 282
- `title` (*telegram.Invoice* attribute), 181, 325
- `title` (*telegram.ShippingOption* attribute), 183, 322
- `title` (*telegram.StickerSet* attribute), 149, 327
- `title` (*telegram.Venue* attribute), 143, 314
- `to_json()` (*telegram.TelegramObject* method), 137
- `to_timestamp()` (in module *telegram.utils.helpers*), 53
- `total_amount` (*telegram.Invoice* attribute), 181, 326
- `total_amount` (*telegram.PreCheckoutQuery* attribute), 186, 324
- `total_amount` (*telegram.SuccessfulPayment* attribute), 184, 321
- `total_count` (*telegram.UserProfilePhotos* attribute), 142, 313
- `translation` (*telegram.EncryptedPassportElement* attribute), 199, 286
- `type` (*telegram.Chat* attribute), 95, 238
- `type` (*telegram.EncryptedPassportElement* attribute), 198, 286
- `type` (*telegram.ext.TypeHandler* attribute), 46
- `type` (*telegram.InlineQueryResult* attribute), 151, 255
- `type` (*telegram.InlineQueryResultArticle* attribute), 152, 256
- `type` (*telegram.InlineQueryResultAudio* attribute), 153, 257
- `type` (*telegram.InlineQueryResultCachedAudio* attribute), 155, 258
- `type` (*telegram.InlineQueryResultCachedDocument* attribute), 156, 259
- `type` (*telegram.InlineQueryResultCachedGif* attribute), 157, 260
- `type` (*telegram.InlineQueryResultCachedMpeg4Gif* attribute), 158, 261
- `type` (*telegram.InlineQueryResultCachedPhoto* attribute), 159, 262
- `type` (*telegram.InlineQueryResultCachedSticker* attribute), 160, 264
- `type` (*telegram.InlineQueryResultCachedVideo* attribute), 161, 264
- `type` (*telegram.InlineQueryResultCachedVoice* attribute), 162, 265
- `type` (*telegram.InlineQueryResultContact* attribute), 163, 267
- `type` (*telegram.InlineQueryResultDocument* attribute), 165, 268
- `type` (*telegram.InlineQueryResultGame* attribute), 166, 279
- `type` (*telegram.InlineQueryResultGif* attribute), 167, 269
- `type` (*telegram.InlineQueryResultLocation* attribute), 168, 271
- `type` (*telegram.InlineQueryResultMpeg4Gif* attribute), 170, 272
- `type` (*telegram.InlineQueryResultPhoto* attribute), 171, 274
- `type` (*telegram.InlineQueryResultVenue* attribute), 173, 275
- `type` (*telegram.InlineQueryResultVideo* attribute), 174, 277
- `type` (*telegram.InlineQueryResultVoice* attribute), 176, 278
- `type` (*telegram.InputMediaAnimation* attribute), 112, 339
- `type` (*telegram.InputMediaAudio* attribute), 113, 340
- `type` (*telegram.InputMediaDocument* attribute), 114, 341
- `type` (*telegram.InputMediaPhoto* attribute), 115, 329
- `type` (*telegram.InputMediaVideo* attribute), 116, 330, 338
- `type` (*telegram.MessageEntity* attribute), 131, 301
- `type` (*telegram.PassportElementError* attribute), 189, 331
- `type` (*telegram.PassportElementErrorDataField* attribute), 192, 333
- `type` (*telegram.PassportElementErrorFile* attribute), 190, 331, 334
- `type` (*telegram.PassportElementErrorFiles* attribute), 191, 333
- `type` (*telegram.PassportElementErrorFrontSide* attribute), 191, 332
- `type` (*telegram.PassportElementErrorReverseSide* attribute), 190, 332
- `type` (*telegram.PassportElementErrorSelfie* attribute), 342
- `type` (*telegram.PassportElementErrorTranslationFile* attribute), 343
- `type` (*telegram.PassportElementErrorTranslationFiles* attribute), 343
- `type` (*telegram.PassportElementErrorUnspecified* attribute), 344
- `TypeHandler` (class in *telegram.ext*), 46
- `TYPING` (*telegram.ChatAction* attribute), 100
- ## U
- `Unauthorized`, 107
- `unban_chat_member()` (*telegram.Bot* method), 91
- `unban_member()` (*telegram.Chat* method), 99
- `unbanChatMember()` (*telegram.Bot* method), 91
- `unpin_chat_message()` (*telegram.Bot* method), 92
- `unpinChatMessage()` (*telegram.Bot* method), 92
- `until_date` (*telegram.ChatMember* attribute), 101, 242
- `Update` (class in *telegram*), 137
- `update` (*telegram.ext.filters.Filters* attribute), 13
- `update_chat_data()` (*telegram.ext.BasePersistence* method), 48
- `update_chat_data()` (*telegram.ext.DictPersistence* method), 51
- `update_chat_data()` (*telegram.ext.PicklePersistence* method), 49

`update_conversation()` (*telegram.ext.BasePersistence method*), 48
`update_conversation()` (*telegram.ext.DictPersistence method*), 51
`update_conversation()` (*telegram.ext.PicklePersistence method*), 49
`update_filter` (*telegram.ext.filters.BaseFilter attribute*), 9
`update_id` (*telegram.Update attribute*), 138, 309
`update_persistence()` (*telegram.ext.Dispatcher method*), 8
`update_queue` (*telegram.ext.CallbackContext attribute*), 21
`update_queue` (*telegram.ext.Dispatcher attribute*), 6
`update_queue` (*telegram.ext.Updater attribute*), 3
`update_user_data()` (*telegram.ext.BasePersistence method*), 48
`update_user_data()` (*telegram.ext.DictPersistence method*), 51
`update_user_data()` (*telegram.ext.PicklePersistence method*), 50
`Updater` (*class in telegram.ext*), 3
`UPLOAD_AUDIO` (*telegram.ChatAction attribute*), 100
`UPLOAD_DOCUMENT` (*telegram.ChatAction attribute*), 100
`UPLOAD_PHOTO` (*telegram.ChatAction attribute*), 100
`upload_sticker_file()` (*telegram.Bot method*), 92
`UPLOAD_VIDEO` (*telegram.ChatAction attribute*), 100
`UPLOAD_VIDEO_NOTE` (*telegram.ChatAction attribute*), 100
`uploadStickerFile()` (*telegram.Bot method*), 92
`url` (*telegram.InlineKeyboardButton attribute*), 109, 252
`url` (*telegram.InlineQueryResultArticle attribute*), 152, 256
`URL` (*telegram.MessageEntity attribute*), 133
`url` (*telegram.MessageEntity attribute*), 131, 302
`url` (*telegram.WebhookInfo attribute*), 146, 316
`use_context` (*telegram.ext.Updater attribute*), 4
`User` (*class in telegram*), 139
`user` (*telegram.ChatMember attribute*), 101, 242
`user` (*telegram.GameHighScore attribute*), 189, 320
`user` (*telegram.MessageEntity attribute*), 131, 302
`user_data` (*telegram.ext.CallbackContext attribute*), 21
`user_data` (*telegram.ext.DictPersistence attribute*), 51
`user_data` (*telegram.ext.Dispatcher attribute*), 6, 8
`user_data_json` (*telegram.ext.DictPersistence attribute*), 51
`user_id` (*telegram.Contact attribute*), 105, 249
`user_sig_handler` (*telegram.ext.Updater attribute*), 3
`username` (*telegram.Bot attribute*), 92
`username` (*telegram.Chat attribute*), 95, 238
`username` (*telegram.User attribute*), 140, 311

`UserProfilePhotos` (*class in telegram*), 142
`utility_bill` (*telegram.SecureData attribute*), 194, 335

V

`vcard` (*telegram.Contact attribute*), 105, 249
`vcard` (*telegram.InlineQueryResultContact attribute*), 164, 267
`vcard` (*telegram.InputContactMessageContent attribute*), 179, 280
`Venue` (*class in telegram*), 142
`venue` (*telegram.ext.filters.Filters attribute*), 13
`venue` (*telegram.Message attribute*), 120, 291
`Video` (*class in telegram*), 143
`video` (*telegram.ext.filters.Filters attribute*), 13
`video` (*telegram.Message attribute*), 120, 290
`video_duration` (*telegram.InlineQueryResultVideo attribute*), 175, 277
`video_file_id` (*telegram.InlineQueryResultCachedVideo attribute*), 161, 264
`video_height` (*telegram.InlineQueryResultVideo attribute*), 175, 277
`video_note` (*telegram.ext.filters.Filters attribute*), 14
`video_note` (*telegram.Message attribute*), 120, 290
`video_url` (*telegram.InlineQueryResultVideo attribute*), 175, 277
`video_width` (*telegram.InlineQueryResultVideo attribute*), 175, 277
`VideoNote` (*class in telegram*), 144
`Voice` (*class in telegram*), 145
`voice` (*telegram.ext.filters.Filters attribute*), 14
`voice` (*telegram.Message attribute*), 120, 290
`voice_duration` (*telegram.InlineQueryResultVoice attribute*), 176, 279
`voice_file_id` (*telegram.InlineQueryResultCachedVoice attribute*), 162, 266
`voice_url` (*telegram.InlineQueryResultVoice attribute*), 176, 278

W

`WAITING` (*telegram.ext.ConversationHandler attribute*), 29
`WebhookInfo` (*class in telegram*), 146
`width` (*telegram.Animation attribute*), 55, 317
`width` (*telegram.InputMediaAnimation attribute*), 112, 339
`width` (*telegram.InputMediaVideo attribute*), 116, 330, 338
`width` (*telegram.PhotoSize attribute*), 133, 303
`width` (*telegram.Sticker attribute*), 147, 307
`width` (*telegram.Video attribute*), 143, 314
`workers` (*telegram.ext.Dispatcher attribute*), 6

X

`x_shift` (*telegram.MaskPosition attribute*), [149](#), [328](#)

Y

`y_shift` (*telegram.MaskPosition attribute*), [149](#), [328](#)