



Python Telegram Bot Documentation

Release 9.0.0

Leandro Toledo

Feb 14, 2018

1	telegram package	3
1.1	telegram.utils.helpers Module	3
1.2	telegram.ext package	4
1.2.1	telegram.ext.Updater	4
1.2.2	telegram.ext.Dispatcher	6
1.2.3	telegram.ext.filters Module	8
1.2.4	telegram.ext.Job	11
1.2.5	telegram.ext.JobQueue	13
1.2.6	telegram.ext.MessageQueue	15
1.2.7	telegram.ext.DelayQueue	16
1.2.8	Handlers	17
1.3	telegram.Audio	34
1.4	telegram.Bot	35
1.5	telegram.CallbackQuery	64
1.6	telegram.Chat	66
1.7	telegram.ChatAction	68
1.8	telegram.ChatMember	69
1.9	telegram.ChatPhoto	71
1.10	telegram.constants Module	71
1.11	telegram.Contact	72
1.12	telegram.Document	73
1.13	telegram.error module	73
1.14	telegram.File	74
1.15	telegram.ForceReply	75
1.16	telegram.InlineKeyboardButton	75
1.17	telegram.InlineKeyboardMarkup	76
1.18	telegram.InputFile	76
1.19	telegram.InputMedia	77
1.20	telegram.InputMediaPhoto	77
1.21	telegram.InputMediaVideo	78
1.22	telegram.KeyboardButton	78
1.23	telegram.Location	79
1.24	telegram.Message	80
1.25	telegram.MessageEntity	90
1.26	telegram.ParseMode	91
1.27	telegram.PhotoSize	92
1.28	telegram.ReplyKeyboardRemove	92
1.29	telegram.ReplyKeyboardMarkup	93
1.30	telegram.ReplyMarkup	93
1.31	telegram.TelegramObject	94

1.32	telegram.Update	94
1.33	telegram.User	95
1.34	telegram.UserProfilePhotos	96
1.35	telegram.Venue	97
1.36	telegram.Video	97
1.37	telegram.VideoNote	98
1.38	telegram.Voice	98
1.39	telegram.WebhookInfo	99
1.40	Stickers	100
1.40.1	telegram.Sticker	100
1.40.2	telegram.StickerSet	101
1.40.3	telegram.MaskPosition	101
1.41	Inline Mode	102
1.41.1	telegram.InlineQuery	102
1.41.2	telegram.InlineQueryResult	103
1.41.3	telegram.InlineQueryResultArticle	103
1.41.4	telegram.InlineQueryResultAudio	104
1.41.5	telegram.InlineQueryResultCachedAudio	105
1.41.6	telegram.InlineQueryResultCachedDocument	106
1.41.7	telegram.InlineQueryResultCachedGif	107
1.41.8	telegram.InlineQueryResultCachedMpeg4Gif	108
1.41.9	telegram.InlineQueryResultCachedPhoto	108
1.41.10	telegram.InlineQueryResultCachedSticker	109
1.41.11	telegram.InlineQueryResultCachedVideo	110
1.41.12	telegram.InlineQueryResultCachedVoice	111
1.41.13	telegram.InlineQueryResultContact	112
1.41.14	telegram.InlineQueryResultDocument	113
1.41.15	telegram.InlineQueryResultGame	114
1.41.16	telegram.InlineQueryResultGif	114
1.41.17	telegram.InlineQueryResultLocation	115
1.41.18	telegram.InlineQueryResultMpeg4Gif	116
1.41.19	telegram.InlineQueryResultPhoto	117
1.41.20	telegram.InlineQueryResultVenue	119
1.41.21	telegram.InlineQueryResultVideo	120
1.41.22	telegram.InlineQueryResultVoice	121
1.41.23	telegram.InputMessageContent	122
1.41.24	telegram.InputTextMessageContent	122
1.41.25	telegram.InputLocationMessageContent	122
1.41.26	telegram.InputVenueMessageContent	123
1.41.27	telegram.InputContactMessageContent	123
1.41.28	telegram.ChosenInlineResult	124
1.42	Payments	124
1.42.1	telegram.LabeledPrice	124
1.42.2	telegram.Invoice	125
1.42.3	telegram.ShippingAddress	125
1.42.4	telegram.OrderInfo	126
1.42.5	telegram.ShippingOption	126
1.42.6	telegram.SuccessfulPayment	127
1.42.7	telegram.ShippingQuery	128
1.42.8	telegram.PreCheckoutQuery	129
1.43	Games	130
1.43.1	telegram.Game	130
1.43.2	telegram.Animation	131
1.43.3	telegram.Callbackgame	132
1.43.4	telegram.GameHighScore	132
1.44	Module contents	132

3 Indices and tables	227
Python Module Index	229

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.utils.helpers Module

This module contains helper functions.

`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 telegram.ext package

1.2.1 telegram.ext.Updater

```
class telegram.ext.Updater (token=None, base_url=None, workers=4, bot=None,  
                             user_sig_handler=None, request_kwargs=None)
```

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*telegram.Bot* – The bot used with this Updater.**user_sig_handler**

signal – signals the updater will respond to.

update_queue

Queue – Queue for the updates.

job_queue*telegram.ext.JobQueue* – Jobqueue for the updater.**dispatcher***telegram.ext.Dispatcher* – Dispatcher that handles the updates and dispatches them to the handlers.**running**

bool – Indicates if the updater is running.

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.
- **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) setable 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.

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.SIGIOT: 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, *network_delay*=None, *clean*=False, *bootstrap_retries*=0, *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
 - 0 - no retries (default)
 - > 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).
- **network_delay** – Deprecated. Will be honoured as *read_latency* for a while but will be removed in the future.

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
 - 0 - no retries (default)
 - > 0 - retry up to X times
- **webhook_url** (str, optional) – Explicitly specify the webhook url. Useful behind NAT, reverse proxy, etc. Default is derived from *listen*, *port* & *url_path*.
- **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.

1.2.2 telegram.ext.Dispatcher

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

Bases: object

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

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

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.

add_error_handler (*callback*)

Registers an error handler in the Dispatcher.

Parameters **callback** (callable) – A function that takes Bot, Update, TelegramError as arguments.

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.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.

chat_data = None

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

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

List[callable] – A list of errorHandlers.

classmethod **get_instance** ()

Get the singleton instance of this class.

Returns `telegram.ext.Dispatcher`

Raises `RuntimeError`

groups = None

List[int] – A list with all groups.

handlers = None

Dict[int, List[`telegram.ext.Handler`]] – Holds the handlers per group.

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/kwags) to be run asynchronously.

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

bool – Indicates if this dispatcher is running.

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.

user_data = None

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

1.2.3 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

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

filter (*message*)

This method must be overwritten.

Parameters `message` (*telegram.Message*) – The message that is tested.

Returns 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`

Filter – All Messages.

audio = `Filters.audio`

Filter – Messages that contain *telegram.Audio*.

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`

Filter – Messages starting with /.

contact = `Filters.contact`

Filter – Messages that contain *telegram.Contact*.

document = `Filters.document`

Filter – Messages that contain *telegram.Document*.

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`

Filter – Messages that are forwarded.

game = `Filters.game`

Filter – Messages that contain *telegram.Game*.

group = Filters.group

Filter – Messages sent in a group chat.

invoice = Filters.invoice

Filter – Messages that contain *telegram.Invoice*.

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

Filter – Messages that contain *telegram.Location*.

photo = Filters.photo

Filter – Messages that contain *telegram.PhotoSize*.

private = Filters.private

Filter – Messages sent in a private chat.

reply = Filters.reply

Filter – Messages that are a reply to another message.

status_update = Filters.status_update

Subset for messages containing a status update.

Examples

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

chat_created

Filter – Messages that contain *telegram.Message.group_chat_created*, *telegram.Message.supergroup_chat_created* or *telegram.Message.channel_chat_created*.

delete_chat_photo

Filter – Messages that contain *telegram.Message.delete_chat_photo*.

left_chat_member

Filter – Messages that contain *telegram.Message.left_chat_member*.

migrate

Filter – Messages that contain *telegram.Message.migrate_from_chat_id* or *attr: telegram.Message.migrate_from_chat_id*.

new_chat_members

Filter – Messages that contain *telegram.Message.new_chat_member*.

new_chat_photo

Filter – Messages that contain *telegram.Message.new_chat_photo*.

new_chat_title

Filter – Messages that contain *telegram.Message.new_chat_title*.

pinned_message

Filter – Messages that contain *telegram.Message.pinned_message*.

sticker = Filters.sticker

Filter – Messages that contain *telegram.Sticker*.

successful_payment = Filters.successful_payment

Filter – Messages that confirm a *telegram.SuccessfulPayment*.

text = Filters.text

Filter – Text Messages.

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

Filter – Messages that contain *telegram.Venue*.

video = Filters.video

Filter – Messages that contain *telegram.Video*.

voice = Filters.voice

Filter – Messages that contain *telegram.Voice*.

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

Bases: *telegram.ext.filters.BaseFilter*

Represents a filter that has been inverted.

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

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.

1.2.4 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

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

context

object – Optional. Additional data needed for the callback function.

name

str – Optional. The name of the new job.

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

Tuple[int] – Optional. Defines on which days of the week the job should run.

enabled

bool – Whether this job is enabled.

interval

int | float | datetime.timedelta – Optional. The interval in which the job will run.

interval_seconds

int – The interval for this job in seconds.

job_queue

`telegram.ext.JobQueue` – Optional. The `JobQueue` this job belongs to.

removed

bool – Whether this job is due to be removed.

repeat

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

run (bot)

Executes the callback function.

schedule_removal ()

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

1.2.5 telegram.ext.JobQueue

class telegram.ext.JobQueue (bot, prevent_autostart=None)

Bases: object

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

_queue

PriorityQueue – The queue that holds the Jobs.

bot

telegram.Bot – Bot that's send to the handlers.

Parameters **bot** (telegram.Bot) – The bot instance that should be passed to the jobs.

Deprecated: prevent_autostart (bool, optional): Thread does not start during initialisation. Use start method instead.

jobs ()

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

put (job, next_t=None)

Queue a new job.

Note: This method is deprecated. Please use: `run_once`, `run_daily` or `run_repeating` instead.

Parameters

- **job** (telegram.ext.Job) – The Job instance representing the new job.
- **next_t** (int | float | datetime.timedelta | datetime.datetime | datetime.time, optional) – Time in or at which the job should run for the first time. 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 at which the job should run. This could be either today or, if the time has already passed, tomorrow.

run_daily (callback, time, days=(0, 1, 2, 3, 4, 5, 6), 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 it's 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 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.
- **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.

1.2.6 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 True.

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

__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.

1.2.7 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

int – Number of maximum callbacks to process per time-window.

time_limit

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

exc_route

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

name

str – Thread's name.

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*.

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.

1.2.8 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

callable – The callback function for this handler.

pass_update_queue

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

pass_job_queue

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

pass_user_data

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

pass_chat_data

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

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.

Parameters

- **callback** (callable) – A function that takes `bot`, `update` as positional arguments. It will be called when the `check_update` has determined that an update should be processed by this handler.
- **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`.

`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 bool

`collect_optional_args(dispatcher, update=None)`

Prepares the optional arguments that are the same for all types of handlers.

Parameters `dispatcher` (`telegram.ext.Dispatcher`) – The dispatcher.

`handle_update(update, dispatcher)`

This method is called if it was determined that an update should indeed be handled by this instance. It should also be overridden, but in most cases call `self.callback(dispatcher.bot, update)`, possibly along with optional arguments. To work with the `ConversationHandler`, this method should return the value returned from `self.callback`

Parameters

- **update** (str | `telegram.Update`) – The update to be handled.
- **dispatcher** (`telegram.ext.Dispatcher`) – The dispatcher to collect optional args.

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

callable – The callback function for this handler.

pass_update_queue

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

pass_job_queue

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

pattern

str | *Pattern* – Optional. Regex pattern to test `telegram.CallbackQuery.data` against.

pass_groups

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

pass_groupdict

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

pass_user_data

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

pass_chat_data

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

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.

Parameters

- **callback** (callable) – A function that takes `bot`, `update` as positional arguments. It will be called when the `check_update` has determined that an update should be processed by this handler.
- **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`.
- **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`.
- **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_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`.

check_update (*update*)

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

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

Returns bool

handle_update (*update*, *dispatcher*)

Send the update to the `callback`.

Parameters

- **update** (*telegram.Update*) – Incoming telegram update.
- **dispatcher** (*telegram.ext.Dispatcher*) – Dispatcher that originated the 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

callable – The callback function for this handler.

pass_update_queue

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

pass_job_queue

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

pass_user_data

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

pass_chat_data

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

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.

Parameters

- **callback** (callable) – A function that takes *bot*, *update* as positional arguments. It will be called when the *check_update* has determined that an update should be processed by this handler.
- **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*.

check_update (*update*)

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

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

Returns bool

handle_update (*update*, *dispatcher*)
 Send the update to the *callback*.

Parameters

- **update** (*telegram.Update*) – Incoming telegram update.
- **dispatcher** (*telegram.ext.Dispatcher*) – Dispatcher that originated the Update.

telegram.ext.ConversationHandler

```
class telegram.ext.ConversationHandler(entry_points, states, fallbacks, allow_reentry=False, run_async_timeout=None,
                                       timed_out_behavior=None, per_chat=True,
                                       per_user=True, per_message=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. You will probably use mostly *telegram.ext.MessageHandler* and *telegram.ext.RegexHandler* here.

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. To end the conversation, the callback function must return `:attr'END'` or `-1`.

entry_points

List[*telegram.ext.Handler*] – A list of `Handler` objects that can trigger the start of the conversation.

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.

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*.

allow_reentry

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

run_async_timeout

float – Optional. The time-out for `run_async` decorated Handlers.

timed_out_behavior

List[*telegram.ext.Handler*] – Optional. A list of handlers that might be used if the wait for `run_async` timed out.

per_chat

bool – Optional. If the conversationkey should contain the Chat's ID.

per_user

bool – Optional. If the conversationkey should contain the User's ID.

per_message

bool – Optional. If the conversationkey should contain the Message's ID.

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.
- **run_async_timeout** (float, optional) – If the previous handler for this user was running asynchronously using the [run_async](#) decorator, it might not be finished when the next message arrives. This timeout defines how long the conversation handler should wait for the next state to be computed. The default is None which means it will wait indefinitely.
- **timed_out_behavior** (List[[telegram.ext.Handler](#)], optional) – A list of handlers that might be used if the wait for [run_async](#) timed out. The first handler which [check_update](#) method returns True will be used. If all return False, the update is not handled.
- **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.

Raises ValueError

END = -1

int – Used as a constant to return when a conversation is ended.

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*)

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

Parameters

- **update** ([telegram.Update](#)) – Incoming telegram update.

- **dispatcher** (*telegram.ext.Dispatcher*) – Dispatcher that originated the Update.

telegram.ext.CommandHandler

```
class telegram.ext.CommandHandler(command, callback, filters=None, allow_edited=False,  
                                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.

command

str | List[str] – The command or list of commands this handler should listen for.

callback

callable – The callback function for this handler.

filters

telegram.ext.BaseFilter – Optional. Only allow updates with these Filters.

allow_edited

bool – Optional. Determines Whether the handler should also accept edited messages.

pass_args

bool – Optional. Determines whether the handler should be passed args.

pass_update_queue

bool – Optional. Determines whether update_queue will be passed to the callback function.

pass_job_queue

bool – Optional. Determines whether job_queue will be passed to the callback function.

pass_user_data

bool – Optional. Determines whether user_data will be passed to the callback function.

pass_chat_data

bool – Optional. Determines whether chat_data will be passed to the callback function.

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.

Parameters

- **command** (str | List[str]) – The command or list of commands this handler should listen for.
- **callback** (callable) – A function that takes bot, update as positional arguments. It will be called when the *check_update* has determined that an update should be processed by this handler.
- **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.

- **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`
- **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`.

check_update (*update*)

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

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

Returns bool

handle_update (*update, dispatcher*)

Send the update to the *callback*.

Parameters

- **update** (*telegram.Update*) – Incoming telegram update.
- **dispatcher** (*telegram.ext.Dispatcher*) – Dispatcher that originated the 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

callable – The callback function for this handler.

pass_update_queue

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

pass_job_queue

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

pattern

str | Pattern – Optional. Regex pattern to test `telegram.InlineQuery.query` against.

pass_groups

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

pass_groupdict

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

pass_user_data

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

pass_chat_data

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

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`.

Parameters

- **callback** (callable) – A function that takes `bot`, `update` as positional arguments. It will be called when the `check_update` has determined that an update should be processed by this handler.
- **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`.
- **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`
- **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_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`.

check_update (*update*)

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

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

Returns bool

handle_update (*update*, *dispatcher*)

Send the update to the `callback`.

Parameters

- **update** (`telegram.Update`) – Incoming telegram update.
- **dispatcher** (`telegram.ext.Dispatcher`) – Dispatcher that originated the Update.

telegram.ext.MessageHandler

```
class telegram.ext.MessageHandler (filters,          callback,          allow_edited=False,
                                     pass_update_queue=False, pass_job_queue=False,
                                     pass_user_data=False, pass_chat_data=False, mes-
                                     sage_updates=True,      channel_post_updates=True,
                                     edited_updates=False)
```

Bases: telegram.ext.handler.Handler

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

filters

Filter – Only allow updates with these Filters. See [telegram.ext.filters](#) for a full list of all available filters.

callback

callable – The callback function for this handler.

pass_update_queue

bool – Optional. Determines whether update_queue will be passed to the callback function.

pass_job_queue

bool – Optional. Determines whether job_queue will be passed to the callback function.

pass_user_data

bool – Optional. Determines whether user_data will be passed to the callback function.

pass_chat_data

bool – Optional. Determines whether chat_data will be passed to the callback function.

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.

allow_edited

bool – Optional. If the handler should also accept edited messages. Default is False - Deprecated. use edited_updates instead.

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.

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).
- **callback** (callable) – A function that takes bot, update as positional arguments. It will be called when the [check_update](#) has determined that an update should be processed by this handler.
- **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.
- **allow_edited** (bool, optional) – If the handler should also accept edited messages. Default is False - Deprecated. use `edited_updates` instead.

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

handle_update (*update*, *dispatcher*)

Send the update to the *callback*.

Parameters

- **update** (`telegram.Update`) – Incoming telegram update.
- **dispatcher** (`telegram.ext.Dispatcher`) – Dispatcher that originated the 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

callable – The callback function for this handler.

pass_update_queue

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

pass_job_queue

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

pass_user_data

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

pass_chat_data

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

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.

Parameters

- **callback** (callable) – A function that takes `bot`, `update` as positional arguments. It will be called when the `check_update` has determined that an update should be processed by this handler.
- **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`.

`check_update(update)`

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

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

Returns bool

`handle_update(update, dispatcher)`

Send the update to the `callback`.

Parameters

- **update** (`telegram.Update`) – Incoming telegram update.
- **dispatcher** (`telegram.ext.Dispatcher`) – Dispatcher that originated the 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.handler.Handler`

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

str | Pattern – The regex pattern.

callback

callable – The callback function for this handler.

pass_groups

bool – Optional. Determines whether groups will be passed to the callback function.

pass_groupdict

bool – Optional. Determines whether groupdict. will be passed to the callback function.

pass_update_queue

bool – Optional. Determines whether update_queue will be passed to the callback function.

pass_job_queue

bool – Optional. Determines whether job_queue will be passed to the callback function.

pass_user_data

bool – Optional. Determines whether user_data will be passed to the callback function.

pass_chat_data

bool – Optional. Determines whether chat_data will be passed to the callback function.

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.

Parameters

- **pattern** (str | Pattern) – The regex pattern.
- **callback** (callable) – A function that takes bot, update as positional arguments. It will be called when the `check_update` has determined that an update should be processed 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
- **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.
- **allow_edited** (bool, optional) – If the handler should also accept edited messages. Default is False - Deprecated. use edited_updates instead.

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`

handle_update (*update*, *dispatcher*)

Send the update to the *callback*.

Parameters

- **update** (*telegram.Update*) – Incoming telegram update.
- **dispatcher** (*telegram.ext.Dispatcher*) – Dispatcher that originated the 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

`callable` – The callback function for this handler.

pass_update_queue

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

pass_job_queue

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

pass_user_data

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

pass_chat_data

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

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`.

Parameters

- **callback** (`callable`) – A function that takes `bot`, `update` as positional arguments. It will be called when the *check_update* has determined that an update should be processed by this handler.
- **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.

check_update (*update*)

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

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

Returns bool

handle_update (*update, dispatcher*)

Send the update to the *callback*.

Parameters

- **update** (*telegram.Update*) – Incoming telegram update.
- **dispatcher** (*telegram.ext.Dispatcher*) – Dispatcher that originated the Update.

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

str – The command this handler should listen for.

callback

callable – The callback function for this handler.

pass_args

bool – Optional. Determines whether the handler should be passed args.

pass_update_queue

bool – Optional. Determines whether update_queue will be passed to the callback function.

pass_job_queue

bool – Optional. Determines whether job_queue will be passed to the callback function.

Parameters

- **command** (str) – The command this handler should listen for.
- **callback** (callable) – A function that takes bot, update as positional arguments. It will be called when the *check_update* has determined that a command should be processed by this handler.
- **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

- **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 class: `telegram.ext.JobQueue` instance created by the `telegram.ext.Updater` which can be used to schedule new jobs. Default is False.

check_update (*update*)

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

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

Returns bool

handle_update (*update*, *dispatcher*)

Send the update to the *callback*.

Parameters

- **update** (*str*) – An incoming command.
- **dispatcher** (`telegram.ext.Dispatcher`) – Dispatcher that originated the command.

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

str | `Pattern` – The regex pattern.

callback

callable – The callback function for this handler.

pass_groups

bool – Optional. Determines whether groups will be passed to the callback function.

pass_groupdict

bool – Optional. Determines whether groupdict. will be passed to the callback function.

pass_update_queue

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

pass_job_queue

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

Parameters

- **pattern** (*str* | `Pattern`) – The regex pattern.

- **callback** (callable) – A function that takes `bot`, `update` as positional arguments. It will be called when the `check_update` has determined that an update should be processed 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`
- **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`.

check_update (*update*)

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

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

Returns bool

handle_update (*update*, *dispatcher*)

Send the update to the *callback*.

Parameters

- **update** (str) – An incoming command.
- **dispatcher** (`telegram.ext.Dispatcher`) – Dispatcher that originated the command.

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

type – The type of updates this handler should process.

callback

callable – The callback function for this handler.

strict

bool – Optional. Use *type* instead of `isinstance`. Default is `False`

pass_update_queue

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

pass_job_queue

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

Parameters

- **type** (*type*) – The type of updates this handler should process, as determined by `isinstance`

- **callback** (callable) – A function that takes `bot`, `update` as positional arguments. It will be called when the `check_update` has determined that an update should be processed by this handler.
- **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`.
- **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`.

check_update (*update*)

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

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

Returns bool

handle_update (*update*, *dispatcher*)

Send the update to the `callback`.

Parameters

- **update** (`telegram.Update`) – Incoming telegram update.
- **dispatcher** (`telegram.ext.Dispatcher`) – Dispatcher that originated the Update.

1.3 telegram.Audio

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

Bases: `telegram.base.TelegramObject`

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

file_id

str – Unique identifier for this file.

duration

int – Duration of the audio in seconds.

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.

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

1.4 telegram.Bot

class telegram.**Bot** (*token*, *base_url=None*, *base_file_url=None*, *request=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.

add_sticker_to_set (*user_id*, *name*, *png_sticker*, *emojis*, *mask_position=None*, *timeout=None*, ***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

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*

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

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

edit_message_caption (*chat_id=None*, *message_id=None*, *inline_message_id=None*, *caption=None*, *reply_markup=None*, *timeout=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.
- **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*, ***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_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.
- **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`

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

str – Bot's first name.

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*

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 `getFile` again.

Parameters

- **file_id** (`str`) – File identifier to get info 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, network_delay=None, 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

int – Unique identifier for this bot.

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

str – Optional. Bot’s last name.

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

str – Bot’s @username.

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

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

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`

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`, ****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, 0-200 characters.
- **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.
- **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*, ***kwargs*)

Use this method to send phone contacts.

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

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.
- **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*, ***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), 0-200 characters.
- **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_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*, ***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), 0-200 characters.
- **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=None*, ***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`, ****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`.

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.
- **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*, ***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*), 0-200 characters.
- **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_video_note(chat_id, video_note, duration=None, length=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, **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.
- **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, **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, 0-200 characters.
- **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*

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=None*, ***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** (*telegram.InputFile*) – 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_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`

stop_message_live_location (`chat_id=None`, `message_id=None`, `inline_message_id=None`, `reply_markup=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`

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`

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`

upload_sticker_file (*user_id*, *png_sticker*, *timeout=None*, ***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

`str` – Bot’s username.

1.5 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

`str` – Unique identifier for this query.

from_user

`telegram.User` – Sender.

message

`telegram.Message` – Optional. Message with the callback button that originated the query.

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.

data

`str` – Optional. Data associated with the callback button.

game_short_name

`str` – Optional. Short name of a Game to be returned.

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

Shortcut for either:

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

or:

```
bot.edit_message_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 (**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,
                             *args, **kwargs)
```

or:

```
bot.edit_message_reply_markup(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_text (*args, **kwargs)

Shortcut for either:

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

or:

```
bot.edit_message_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*

1.6 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

int – Unique identifier for this chat.

type

str – Type of chat.

title

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

username

str – Optional. Username.

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.

photo

telegram.ChatPhoto – Optional. Chat photo.

description

str – Optional. Description, for supergroups and channel chats.

invite_link

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

pinned_message

telegram.Message – Optional. Pinned message, for supergroups. Returned only in get_chat.

sticker_set_name

str – Optional. For supergroups, name of Group sticker set.

can_set_sticker_set

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

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’
`str` – ‘channel’

GROUP = ‘group’
`str` – ‘group’

PRIVATE = ‘private’
`str` – ‘private’

SUPERGROUP = ‘supergroup’
`str` – ‘supergroup’

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.

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`

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`

1.7 telegram.ChatAction

class `telegram.ChatAction`

Bases: `object`

Helper class to provide constants for different chat actions.

```
FIND_LOCATION = 'find_location'
    str - 'find_location'

RECORD_AUDIO = 'record_audio'
    str - 'record_audio'

RECORD_VIDEO = 'record_video'
    str - 'record_video'

RECORD_VIDEO_NOTE = 'record_video_note'
    str - 'record_video_note'

TYPING = 'typing'
    str - 'typing'

UPLOAD_AUDIO = 'upload_audio'
    str - 'upload_audio'

UPLOAD_DOCUMENT = 'upload_document'
    str - 'upload_document'

UPLOAD_PHOTO = 'upload_photo'
    str - 'upload_photo'

UPLOAD_VIDEO = 'upload_video'
    str - 'upload_video'

UPLOAD_VIDEO_NOTE = 'upload_video_note'
    str - 'upload_video_note'
```

1.8 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

telegram.User – Information about the user.

status

str – The member's status in the chat.

until_date

datetime.datetime – Optional. Date when restrictions will be lifted for this user.

can_be_edited

bool – Optional. If the bot is allowed to edit administrator privileges of that user.

can_change_info

bool – Optional. If the administrator can change the chat title, photo and other settings.

can_post_messages

bool – Optional. If the administrator can post in the channel.

can_edit_messages

bool – Optional. If the administrator can edit messages of other users.

can_delete_messages

bool – Optional. If the administrator can delete messages of other users.

can_invite_users

bool – Optional. If the administrator can invite new users to the chat.

can_restrict_members

bool – Optional. If the administrator can restrict, ban or unban chat members.

can_pin_messages

bool – Optional. If the administrator can pin messages.

can_promote_members

bool – Optional. If the administrator can add new administrators.

can_send_messages

bool – Optional. If the user can send text messages, contacts, locations and venues.

can_send_media_messages

bool – Optional. If the user can send media messages, implies can_send_messages.

can_send_other_messages

bool – Optional. If the user can send animations, games, stickers and use inline bots, implies can_send_media_messages.

can_add_web_page_previews

bool – Optional. If user may add web page previews to his messages, implies can_send_media_messages

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'  
str – 'administrator'
```

```
CREATOR = 'creator'  
str – 'creator'
```

```
KICKED = 'kicked'  
str – 'kicked'
```

```
LEFT = 'left'  
str – 'left'
```

```
MEMBER = 'member'  
str – 'member'
```

```
RESTRICTED = 'restricted'  
str – 'restricted'
```

1.9 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  
str – Unique file identifier of small (160x160) chat photo.
```

```
big_file_id  
str – Unique file identifier of big (640x640) chat photo.
```

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.

1.10 telegram.constants Module

Constants in the Telegram network.

The following constants were extracted from the [Telegram Bots FAQ](#).

```
telegram.constants.MAX_MESSAGE_LENGTH
    int – 4096

telegram.constants.MAX_CAPTION_LENGTH
    int – 200

telegram.constants.SUPPORTED_WEBHOOK_PORTS
    List[int] – [443, 80, 88, 8443]

telegram.constants.MAX_FILESIZE_DOWNLOAD
    int – In bytes (20MB)

telegram.constants.MAX_FILESIZE_UPLOAD
    int – In bytes (50MB)

telegram.constants.MAX_MESSAGES_PER_SECOND_PER_CHAT
    int – 1. Telegram may allow short bursts that go over this limit, but eventually you'll begin receiving 429
    errors.

telegram.constants.MAX_MESSAGES_PER_SECOND
    int – 30

telegram.constants.MAX_MESSAGES_PER_MINUTE_PER_GROUP
    int – 20

telegram.constants.MAX_INLINE_QUERY_RESULTS
    int – 50
```

The following constant have been found by experimentation:

```
telegram.constants.MAX_MESSAGE_ENTITIES
    int – 100 (Beyond this cap telegram will simply ignore further formatting styles)
```

1.11 telegram.Contact

```
class telegram.Contact(phone_number, first_name, last_name=None, user_id=None,
                       **kwargs)
```

Bases: telegram.base.TelegramObject

This object represents a phone contact.

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.

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

1.12 telegram.Document

```
class telegram.Document(file_id, thumb=None, file_name=None, mime_type=None,  
                        file_size=None, **kwargs)  
    Bases: telegram.base.TelegramObject
```

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

```
file_id  
    str – Unique file identifier.  
  
thumb  
    telegram.PhotoSize – Optional. Document thumbnail.  
  
file_name  
    str – Original filename.  
  
mime_type  
    str – Optional. MIME type of the file.  
  
file_size  
    int – Optional. File size.
```

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

1.13 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.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.Timeout  
    Bases: telegram.error.NetworkError
```

exception telegram.error.Unauthorized(*message*)
Bases: telegram.error.TelegramError

1.14 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
str – Unique identifier for this file.

file_size
str – Optional. File size.

file_path
str – Optional. File path. Use `download` to get the file.

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.

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** (*object*, optional) – A file-like object. Must be opened 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).

Raises `ValueError` – If both `custom_path` and `out` are passed.

1.15 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

True – Shows reply interface to the user.

selective

bool – Optional. Force reply from specific users only.

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.

1.16 telegram.InlineKeyboardButton

class telegram.InlineKeyboardButton (*text, url=None, callback_data=None, switch_inline_query=None, switch_inline_query_current_chat=None, callback_game=None, pay=None, **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

str – Label text on the button.

url

str – Optional. 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. 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.

switch_inline_query_current_chat

str – Optional. Will insert the bot's username and the specified inline query in the current chat's input field.

callback_game

`telegram.CallbackGame` – Optional. Description of the game that will be launched when the user presses the button.

pay

`bool` – Optional. Specify `True`, to send a Pay button.

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.

1.17 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

`List[List[telegram.InlineKeyboardButton]]` – Array of button rows, each represented by an Array of `InlineKeyboardButton` objects.

Parameters

- **inline_keyboard** (`List[List[telegram.InlineKeyboardButton]]`) – Array of button rows, each represented by an Array of `InlineKeyboardButton` objects.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.18 telegram.InputFile

class `telegram.InputFile` (`data`)

Bases: `object`

This object represents a Telegram `InputFile`.

data

`dict` – Data containing an inputfile.

Parameters `data` (`dict`) – Data containing an inputfile.

Raises `TelegramError`

content_type

`str` – Content type

headers

`dict` – Headers.

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`

static `is_inputfile` (`data`)

Check if the request is a file request.

Parameters `data` (`Dict[str, str]`) – A dict of (str, str) key/value pairs.

Returns `bool`

to_form ()

Transform the inputfile to multipart/form data.

Returns `str`

1.19 telegram.InputMedia

class `telegram.InputMedia`

Bases: `telegram.base.TelegramObject`

Base class for Telegram InputMedia Objects.

See `telegram.InputMediaPhoto` and `telegram.InputMediaVideo` for detailed use.

1.20 telegram.InputMediaPhoto

class `telegram.InputMediaPhoto` (`media`, `caption=None`)

Bases: `telegram.files.inputmedia.InputMedia`

Represents a photo to be sent.

type

`str` – photo.

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, 0-200 characters.

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, 0-200 characters.

Note: At the moment using a new file is not yet supported.

1.21 telegram.InputMediaVideo

class telegram.**InputMediaVideo** (*media*, *caption=None*, *width=None*, *height=None*, *duration=None*)

Bases: telegram.files.inputmedia.InputMedia

Represents a video to be sent.

type

`str` – video.

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, 0-200 characters.

width

`int` – Optional. Video width.

height

`int` – Optional. Video height.

duration

`int` – Optional. Video duration.

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, 0-200 characters.
- **width** (`int`, optional) – Video width.
- **height** (`int`, optional) – Video height.
- **duration** (`int`, optional) – Video duration.

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. At the moment using a new file is not yet supported.

1.22 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

`str` – Text of the button.

request_contact

`bool` – Optional. If the user’s phone number will be sent.

request_location

`bool` – Optional. If the user’s current location will be sent.

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.

1.23 telegram.Location

class `telegram.Location` (*longitude, latitude, **kwargs*)

Bases: `telegram.base.TelegramObject`

This object represents a point on the map.

longitude

`float` – Longitude as defined by sender.

latitude

`float` – Latitude as defined by sender.

Parameters

- **longitude** (`float`) – Longitude as defined by sender.
- **latitude** (`float`) – Latitude as defined by sender.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.24 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, new_chat_member=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, 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

int – Unique message identifier inside this chat.

from_user

telegram.User – Optional. Sender.

date

datetime.datetime – Date the message was sent.

chat

telegram.Chat – Conversation the message belongs to.

forward_from

telegram.User – Optional. Sender of the original message.

forward_from_chat

telegram.Chat – Optional. Information about the original channel.

forward_from_message_id

int – Optional. Identifier of the original message in the channel.

forward_date

datetime.datetime – Optional. Date the original message was sent.

reply_to_message

telegram.Message – Optional. The original message.

edit_date

datetime.datetime – Optional. Date the message was last edited.

media_group_id

str – Optional. The unique identifier of a media message group this message belongs to.

text

str – Optional. The actual UTF-8 text of the message.

entities

List[*telegram.MessageEntity*] – Optional. Special entities like usernames, URLs, bot com-

mands, etc. that appear in the text. See `Message.parse_entity` and `parse_entities` methods for how to use properly.

caption_entities

List[`telegram.MessageEntity`] – 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.

audio

`telegram.Audio` – Optional. Information about the file.

document

`telegram.Document` – Optional. Information about the file.

game

`telegram.Game` – Optional. Information about the game.

photo

List[`telegram.PhotoSize`] – Optional. Available sizes of the photo.

sticker

`telegram.Sticker` – Optional. Information about the sticker.

video

`telegram.Video` – Optional. Information about the video.

voice

`telegram.Voice` – Optional. Information about the file.

video_note

`telegram.VideoNote` – Optional. Information about the video message.

new_chat_members

List[`telegram.User`] – Optional. Information about new members to the chat. (the bot itself may be one of these members).

caption

str – Optional. Caption for the document, photo or video, 0-200 characters.

contact

`telegram.Contact` – Optional. Information about the contact.

location

`telegram.Location` – Optional. Information about the location.

venue

`telegram.Venue` – Optional. Information about the venue.

left_chat_member

`telegram.User` – Optional. Information about the user that left the group. (this member may be the bot itself).

new_chat_title

str – Optional. A chat title was changed to this value.

new_chat_photo

List[`telegram.PhotoSize`] – Optional. A chat photo was changed to this value.

delete_chat_photo

bool – Optional. The chat photo was deleted.

group_chat_created

bool – Optional. The group has been created.

supergroup_chat_created

bool – Optional. The supergroup has been created.

channel_chat_created

bool – Optional. The channel has been created.

migrate_to_chat_id

`int` – Optional. The group has been migrated to a supergroup with the specified identifier.

migrate_from_chat_id

`int` – Optional. The supergroup has been migrated from a group with the specified identifier.

pinned_message

`telegram.message` – Optional. Specified message was pinned.

invoice

`telegram.Invoice` – Optional. Information about the invoice.

successful_payment

`telegram.SuccessfulPayment` – Optional. Information about the payment.

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.

bot

`telegram.Bot` – Optional. The Bot to use for instance methods.

Deprecated: 6.0 `new_chat_member` (`telegram.User`): Replaced with `new_chat_members`

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.
- **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, 0-200 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 [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.
- **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.

chat_id

`int` – Shortcut for `telegram.Chat.id` for `chat`.

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_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.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`

new_chat_member

Deprecated

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 code-point 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_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`

1.25 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

`str` – Type of the entity.

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. Url that will be opened after user taps on the text.

user

`telegram.User` – Optional. The mentioned 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', 'bot_command', 'url', 'email', 'bold', 'italic',
List[*str*] – List of all the types.

BOLD = 'bold'
str – ‘bold’

BOT_COMMAND = 'bot_command'
str – ‘bot_command’

CODE = 'code'
str – ‘code’

EMAIL = 'email'
str – ‘email’

HASHTAG = 'hashtag'
str – ‘hashtag’

ITALIC = 'italic'
str – ‘italic’

MENTION = 'mention'
str – ‘mention’

PRE = 'pre'
str – ‘pre’

TEXT_LINK = 'text_link'
str – ‘text_link’

TEXT_MENTION = 'text_mention'
str – ‘text_mention’

URL = 'url'
str – ‘url’

1.26 telegram.ParseMode

class telegram.ParseMode

Bases: object

This object represents a Telegram Message Parse Modes.

HTML = 'HTML'
str – ‘HTML’

MARKDOWN = 'Markdown'
str – ‘Markdown’

1.27 telegram.PhotoSize

class telegram.**PhotoSize** (*file_id*, *width*, *height*, *file_size=None*, ***kwargs*)

Bases: telegram.base.TelegramObject

This object represents one size of a photo or a file/sticker thumbnail.

file_id

str – Unique identifier for this file.

width

int – Photo width.

height

int – Photo height.

file_size

int – Optional. File size.

Parameters

- **file_id** (str) – Unique identifier for this file.
- **width** (int) – Photo width.
- **height** (int) – Photo height.
- **file_size** (int, optional) – File size.
- ****kwargs** (dict) – Arbitrary keyword arguments.

1.28 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

True – Requests clients to remove the custom keyboard.

selective

bool – Optional. Use this parameter if you want to remove the keyboard for specific users only.

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.

1.29 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

List[List[[telegram.KeyboardButton](#) | str]] – Array of button rows.

resize_keyboard

bool – Optional. Requests clients to resize the keyboard.

one_time_keyboard

bool – Optional. Requests clients to hide the keyboard as soon as it's been used.

selective

bool – Optional. Show the keyboard to specific users only.

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.

1.30 telegram.ReplyMarkup

```
class telegram.ReplyMarkup
```

Bases: telegram.base.TelegramObject

Base class for Telegram ReplyMarkup Objects.

See [telegram.ReplyKeyboardMarkup](#) and [telegram.InlineKeyboardMarkup](#) for detailed use.

1.31 telegram.TelegramObject

class telegram.TelegramObject

Bases: object

Base class for most telegram objects.

to_json()

Returns str

1.32 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

int – The update’s unique identifier.

message

telegram.Message – Optional. New incoming message.

edited_message

telegram.Message – Optional. New version of a message.

channel_post

telegram.Message – Optional. New incoming channel post.

edited_channel_post

telegram.Message – Optional. New version of a channel post.

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.

callback_query

telegram.CallbackQuery – Optional. New incoming callback query.

shipping_query

telegram.ShippingQuery – Optional. New incoming shipping query.

pre_checkout_query

telegram.PreCheckoutQuery – Optional. New incoming pre-checkout query.

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

telegram.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*.

effective_message

telegram.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*.

effective_user

telegram.User – The user that sent this update, no matter what kind of update this is. Will be None for *channel_post*.

1.33 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

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.

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

`str` – Convenience property. The user’s `first_name`, followed by (if available) `last_name`.

get_profile_photos (`*args`, `**kwargs`)

Shortcut for:

`bot.get_user_profile_photos(update.message.from_user.id, *args, **kwargs)`

mention_html (`name=None`)

Parameters `name` (`str`) – If provided, will overwrite the user’s name.

Returns The inline mention for the user as HTML.

Return type `str`

mention_markdown (`name=None`)

Parameters `name` (`str`) – If provided, will overwrite the user’s name.

Returns The inline mention for the user as markdown.

Return type `str`

name

`str` – Convenience property. If available, returns the user’s `username` prefixed with “@”. If `username` is not available, returns `full_name`.

1.34 telegram.UserProfilePhotos

class `telegram.UserProfilePhotos` (`total_count`, `photos`, `**kwargs`)

Bases: `telegram.base.TelegramObject`

This object represent a user’s profile pictures.

total_count

`int` – Total number of profile pictures.

photos

List[List[*telegram.PhotoSize*]] – Requested profile pictures.

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).

1.35 telegram.Venue

class telegram.Venue (*location, title, address, foursquare_id=None, **kwargs*)

Bases: telegram.base.TelegramObject

This object represents a venue.

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.

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

1.36 telegram.Video

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

Bases: telegram.base.TelegramObject

This object represents a video file.

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.

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

1.37 telegram.VideoNote

class telegram.**VideoNote** (*file_id, length, duration, thumb=None, file_size=None, **kwargs*)

Bases: telegram.base.TelegramObject

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

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.

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

1.38 telegram.Voice

class telegram.**Voice** (*file_id, duration, mime_type=None, file_size=None, **kwargs*)

Bases: telegram.base.TelegramObject

This object represents a voice note.

file_id

str – Unique identifier for this file.

duration

int – Duration of the audio in seconds as defined by sender.

mime_type

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

file_size

int – Optional. File size.

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

1.39 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

str – Webhook URL.

has_custom_certificate

bool – If a custom certificate was provided for webhook.

pending_update_count

int – Number of updates awaiting delivery.

last_error_date

int – Optional. Unix time for the most recent error that happened.

last_error_message

str – Optional. Error message in human-readable format.

max_connections

int – Optional. Maximum allowed number of simultaneous HTTPS connections.

allowed_updates

List[str] – Optional. A list of update types the bot is subscribed to.

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.

1.40 Stickers

1.40.1 telegram.Sticker

class telegram.**Sticker** (*file_id*, *width*, *height*, *thumb=None*, *emoji=None*, *file_size=None*, *set_name=None*, *mask_position=None*, ***kwargs*)

Bases: telegram.base.TelegramObject

This object represents a sticker.

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.

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.

1.40.2 telegram.StickerSet

class telegram.**StickerSet** (*name, title, contains_masks, stickers, bot=None, **kwargs*)

Bases: telegram.base.TelegramObject

This object represents a sticker set.

name

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.

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.

1.40.3 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

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.

y_shift

float – Shift by Y-axis measured in heights of the mask scaled to the face size, from top to bottom.

scale

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

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'
      str - 'chin'

EYES = 'eyes'
      str - 'eyes'

FOREHEAD = 'forehead'
          str - 'forehead'

MOUTH = 'mouth'
       str - 'mouth'
```

1.41 Inline Mode

1.41.1 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

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.

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.

1.41.2 telegram.InlineQueryResult

class telegram.InlineQueryResult (*type, id, **kwargs*)

Bases: telegram.base.TelegramObject

Baseclass for the InlineQueryResult* classes.

type

str – Type of the result.

id

str – Unique identifier for this result, 1-64 Bytes.

Parameters

- **type** (str) – Type of the result.
- **id** (str) – Unique identifier for this result, 1-64 Bytes.
- ****kwargs** (dict) – Arbitrary keyword arguments.

1.41.3 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

str – 'article'.

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.

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.

1.41.4 telegram.InlineQueryResultAudio

```
class telegram.InlineQueryResultAudio(id, audio_url, title, performer=None,
                                       audio_duration=None, caption=None,
                                       input_message_content=None, reply_markup=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
str – 'audio'.

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.

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.

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.
- **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.

1.41.5 telegram.InlineQueryResultCachedAudio

```
class telegram.InlineQueryResultCachedAudio(id, audio_file_id, caption=None,
                                             reply_markup=None, input_message_content=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
 str – 'audio'.

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

audio_file_id
 str – A valid file identifier for the audio file.

caption
 str – Optional. Caption, 0-200 characters

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.

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, 0-200 characters
- **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.

1.41.6 telegram.InlineQueryResultCachedDocument

```
class telegram.InlineQueryResultCachedDocument (id, title, document_file_id, de-
                                                scription=None, caption=None,
                                                reply_markup=None, input_
                                                message_content=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

str – ‘document’.

id

str – Unique identifier for this result, 1-64 bytes.

title

str – Title for the result.

document_file_id

str – A valid file identifier for the file.

description

str – Optional. Short description of the result.

caption

str – Optional. Caption, 0-200 characters

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.

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, 0-200 characters
- **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.

1.41.7 telegram.InlineQueryResultCachedGif

```
class telegram.InlineQueryResultCachedGif(id, gif_file_id, title=None, caption=None, reply_markup=None, input_message_content=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

str – ‘gif’.

id

str – Unique identifier for this result, 1-64 bytes.

gif_file_id

str – A valid file identifier for the GIF file.

title

str – Optional. Title for the result.

caption

str – Optional. Caption, 0-200 characters

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.

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, 0-200 characters
- **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.

1.41.8 telegram.InlineQueryResultCachedMpeg4Gif

```
class telegram.InlineQueryResultCachedMpeg4Gif (id, mpeg4_file_id, title=None, cap-
                                                tion=None, reply_markup=None,
                                                input_message_content=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

str – 'mpeg4_gif'.

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, 0-200 characters

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.

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, 0-200 characters
- **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.

1.41.9 telegram.InlineQueryResultCachedPhoto

```
class telegram.InlineQueryResultCachedPhoto (id, photo_file_id, title=None, de-
                                                scription=None, caption=None,
                                                reply_markup=None,
                                                input_message_content=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
 str – ‘photo’.

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, 0-200 characters

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.

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, 0-200 characters
- **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.

1.41.10 telegram.InlineQueryResultCachedSticker

```
class telegram.InlineQueryResultCachedSticker(id, sticker_file_id, reply_markup=None, input_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
 str – ‘sticker’.

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

sticker_file_id
 str – A valid file identifier of the sticker.

reply_markup

telegram.InlineKeyboardMarkup – Optional. Inline keyboard attached to the message.

input_message_content

telegram.InputMessageContent – Optional. Content of the message to be sent instead of the sticker.

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.

1.41.11 telegram.InlineQueryResultCachedVideo

```
class telegram.InlineQueryResultCachedVideo (id, video_file_id, title, description=None,
                                              caption=None, reply_markup=None, input_message_content=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

str – 'video'.

id

str – Unique identifier for this result, 1-64 bytes.

video_file_id

str – A valid file identifier for the video file.

title

str – Title for the result.

description

str – Optional. Short description of the result.

caption

str – Optional. Caption, 0-200 characters.

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.

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, 0-200 characters.
- **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.

1.41.12 telegram.InlineQueryResultCachedVoice

```
class telegram.InlineQueryResultCachedVoice(id, voice_file_id, title, caption=None, reply_markup=None, input_message_content=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

str – ‘voice’.

id

str – Unique identifier for this result, 1-64 bytes.

voice_file_id

str – A valid file identifier for the voice message.

title

str – Voice message title.

caption

str – Optional. Caption, 0-200 characters.

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.

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, 0-200 characters.
- **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.

1.41.13 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, **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

str – ‘contact’.

id

str – Unique identifier for this result, 1-64 bytes.

phone_number

str – Contact’s phone number.

first_name

str – Contact’s first name.

last_name

str – Optional. Contact’s last name.

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.

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.
- **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.

1.41.14 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, **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

str – ‘document’.

id

str – Unique identifier for this result, 1-64 bytes.

title

str – Title for the result.

caption

str – Optional. Caption, 0-200 characters

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.

Parameters

- **id** (str) – Unique identifier for this result, 1-64 bytes.
- **title** (str) – Title for the result.
- **caption** (str, optional) – Caption, 0-200 characters
- **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.

1.41.15 telegram.InlineQueryResultGame

```
class telegram.InlineQueryResultGame(id, game_short_name, reply_markup=None,  
                                     **kwargs)  
    Bases: telegram.inline.inlinequeryresult.InlineQueryResult  
    Represents a Game.  
  
    type  
        str – ‘game’.  
  
    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.
```

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.

1.41.16 telegram.InlineQueryResultGif

```
class telegram.InlineQueryResultGif(id, gif_url, thumb_url, gif_width=None,  
                                     gif_height=None, title=None, caption=None, re-  
                                     ply_markup=None, input_message_content=None,  
                                     gif_duration=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  
    str – ‘gif’.  
  
id  
    str – Unique identifier for this result, 1-64 bytes.  
  
gif_url  
    str – A valid URL for the GIF file. File size must not exceed 1MB.  
  
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, 0-200 characters

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.

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, 0-200 characters
- **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.

1.41.17 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
 str – 'location'.

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

latitude
 float – Location latitude in degrees.

longitude
float – Location longitude in degrees.

title
str – Location title.

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.

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.

1.41.18 telegram.InlineQueryResultMpeg4Gif

```
class telegram.InlineQueryResultMpeg4Gif (id,          mpeg4_url,          thumb_url,
                                           mpeg4_width=None, mpeg4_height=None,
                                           title=None,          caption=None,
                                           reply_markup=None,          in-
                                           put_message_content=None,
                                           mpeg4_duration=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
 str – ‘mpeg4_gif’.

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, 0-200 characters

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.

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, 0-200 characters
- **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.

1.41.19 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, **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

`str` – ‘photo’.

id

`str` – Unique identifier for this result, 1-64 bytes.

photo_url

`str` – A valid URL of the photo. Photo must be in jpeg format. Photo size must not exceed 5MB.

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, 0-200 characters

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.

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, 0-200 characters
- **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.

1.41.20 telegram.InlineQueryResultVenue

```
class telegram.InlineQueryResultVenue(id, latitude, longitude, title, address,
                                     foursquare_id=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

str – 'venue'.

id

str – Unique identifier for this result, 1-64 Bytes.

latitude

float – Latitude of the venue location in degrees.

longitude

float – Longitude of the venue location in degrees.

title

str – Title of the venue.

address

str – Address of the venue.

foursquare_id

str – Optional. Foursquare identifier of the venue if known.

reply_markup

`telegram.InlineKeyboardMarkup` – Optional. Inline keyboard attached to the message.

input_message_content

`telegram.InputMessageContent` – Optional. Content of the message to be sent instead of the venue.

thumb_url

str – Optional. Url of the thumbnail for the result.

thumb_width

int – Optional. Thumbnail width.

thumb_height

int – Optional. Thumbnail height.

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.
- **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.

1.41.21 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,  
                                     **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

str – ‘video’.

id

str – Unique identifier for this result, 1-64 bytes.

video_url

str – A valid URL for the embedded video player or video file.

mime_type

str – Mime type of the content of 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, 0-200 characters

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.

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, 0-200 characters.
- **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.

1.41.22 telegram.InlineQueryResultVoice

```
class telegram.InlineQueryResultVoice(id, voice_url, title, voice_duration=None,  
                                       caption=None, reply_markup=None, in-  
                                       put_message_content=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

str – ‘voice’.

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, 0-200 characters.

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.

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, 0-200 characters.
- **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.

1.41.23 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.

1.41.24 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

str – Text of the message to be sent, 1-4096 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 your bot's message.

disable_web_page_preview

bool – Optional. Disables link previews for links in the sent message.

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.

1.41.25 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

float – Latitude of the location in degrees.

longitude

float – Longitude of the location in degrees.

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.

1.41.26 telegram.InputVenueMessageContent

```
class telegram.InputVenueMessageContent (latitude, longitude, title, address,  
                                         foursquare_id=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

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.

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

1.41.27 telegram.InputContactMessageContent

```
class telegram.InputContactMessageContent (phone_number, first_name,  
                                           last_name=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

str – Contact's phone number.

first_name

str – Contact's first name.

last_name

str – Optional. Contact's last name.

Parameters

- **phone_number** (`str`) – Contact’s phone number.
- **first_name** (`str`) – Contact’s first name.
- **last_name** (`str`, optional) – Contact’s last name.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.41.28 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

`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.

inline_message_id

`str` – Optional. Identifier of the sent inline message.

query

`str` – The query that was used to obtain the result.

Parameters

- **result_id** (`str`) – The unique identifier for the result that was chosen.
- **from_user** (`telegram.User`) – The user that chose the result.
- **location** (`telegram.Location`, optional) – Sender location, only for bots that require user location.
- **inline_message_id** (`str`, optional) – Identifier of the sent inline message. Available only if there is an inline keyboard attached to the message. Will be also received in callback queries and can be used to edit the message.
- **query** (`str`) – The query that was used to obtain the result.
- ****kwargs** (`dict`) – Arbitrary keyword arguments.

1.42 Payments

1.42.1 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

`str` – Portion label.

amount

`int` – Price of the product in the smallest units of the currency.

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.

1.42.2 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

`str` – Product name.

description

`str` – Product description.

start_parameter

`str` – Unique bot deep-linking parameter.

currency

`str` – Three-letter ISO 4217 currency code.

total_amount

`int` – Total price in the smallest units of the currency.

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.

1.42.3 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

`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.

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.

1.42.4 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
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.

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.

1.42.5 telegram.ShippingOption

class telegram.**ShippingOption** (id, title, prices, **kwargs)
Bases: telegram.base.TelegramObject

This object represents one shipping option.

id
str – Shipping option identifier.

title
str – Option title.

prices
List[*telegram.LabeledPrice*] – List of price portions.

Parameters

- **id** (str) – Shipping option identifier.
- **title** (str) – Option title.
- **prices** (List[*telegram.LabeledPrice*]) – List of price portions.
- ****kwargs** (dict) – Arbitrary keyword arguments.

1.42.6 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
str – Three-letter ISO 4217 currency code.

total_amount
int – Total price in the smallest units of the currency.

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.

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.

1.42.7 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

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.

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.

1.42.8 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

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.

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.

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.

1.43 Games

1.43.1 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

`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`.

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.

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 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*

1.43.2 telegram.Animation

class *telegram.Animation* (*file_id*, *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

str – Unique file identifier.

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.

Parameters

- **file_id** (*str*) – Unique file identifier.
- **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.

1.43.3 telegram.CallbackGame

class telegram.CallbackGame

Bases: telegram.base.TelegramObject

A placeholder, currently holds no information. Use BotFather to set up your game.

1.43.4 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

int – Position in high score table for the game.

user

telegram.User – User.

score

int – Score.

Parameters

- **position** (*int*) – Position in high score table for the game.
- **user** (*telegram.User*) – User.
- **score** (*int*) – Score.

1.44 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, **kwargs*)

Bases: telegram.base.TelegramObject

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

file_id

str – Unique identifier for this file.

duration

int – Duration of the audio in seconds.

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.

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

classmethod `de_json` (data, bot)

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

add_sticker_to_set (user_id, name, png_sticker, emojis, mask_position=None, timeout=None, **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

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`

create_new_sticker_set (`user_id`, `name`, `title`, `png_sticker`, `emojis`, `contains_masks=None`, `mask_position=None`, `timeout=None`, `**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

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`

edit_message_caption (*chat_id=None*, *message_id=None*, *inline_message_id=None*, *caption=None*, *reply_markup=None*, *timeout=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.
- **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, **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_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.
- **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`

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

str – Bot's first name.

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

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 getFile again.

Parameters

- **file_id** (*str*) – File identifier to get info 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*, *network_delay=None*, *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

int – Unique identifier for this bot.

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

str – Optional. Bot’s last name.

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

str – Bot’s @username.

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

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*, *time-*
out=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

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`

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`, ****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, 0-200 characters.
- **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.
- **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*, ***kwargs*)

Use this method to send phone contacts.

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

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.
- **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*, ***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), 0-200 characters.
- **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_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`, `**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`), 0-200 characters.
- **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=None*, ***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*, ***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*.

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.
- **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*, ***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), 0-200 characters.
- **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_video_note(chat_id, video_note, duration=None, length=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, **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.
- **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, **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, 0-200 characters.
- **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*

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=None*, ***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** (*telegram.InputFile*) – 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_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`

stop_message_live_location (*chat_id=None*, *message_id=None*, *inline_message_id=None*, *reply_markup=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()

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`

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`

upload_sticker_file (*user_id, png_sticker, timeout=None, **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

`str` – Bot’s username.

```
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

`int` – Unique identifier for this chat.

type

`str` – Type of chat.

title

`str` – Optional. Title, for supergroups, channels and group chats.

username

`str` – Optional. Username.

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.

photo

`telegram.ChatPhoto` – Optional. Chat photo.

description

`str` – Optional. Description, for supergroups and channel chats.

invite_link

`str` – Optional. Chat invite link, for supergroups and channel chats.

pinned_message

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

sticker_set_name

`str` – Optional. For supergroups, name of Group sticker set.

can_set_sticker_set

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

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'`

`GROUP = 'group'`

`PRIVATE = 'private'`

`SUPERGROUP = 'supergroup'`

`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.

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

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

telegram.User – Information about the user.

status

str – The member's status in the chat.

until_date

datetime.datetime – Optional. Date when restrictions will be lifted for this user.

can_be_edited

bool – Optional. If the bot is allowed to edit administrator privileges of that user.

can_change_info

bool – Optional. If the administrator can change the chat title, photo and other settings.

can_post_messages

bool – Optional. If the administrator can post in the channel.

can_edit_messages

bool – Optional. If the administrator can edit messages of other users.

can_delete_messages

bool – Optional. If the administrator can delete messages of other users.

can_invite_users

bool – Optional. If the administrator can invite new users to the chat.

can_restrict_members

bool – Optional. If the administrator can restrict, ban or unban chat members.

can_pin_messages

bool – Optional. If the administrator can pin messages.

can_promote_members

bool – Optional. If the administrator can add new administrators.

can_send_messages

bool – Optional. If the user can send text messages, contacts, locations and venues.

can_send_media_messages

bool – Optional. If the user can send media messages, implies can_send_messages.

can_send_other_messages

bool – Optional. If the user can send animations, games, stickers and use inline bots, implies can_send_media_messages.

can_add_web_page_previews

bool – Optional. If user may add web page previews to his messages, implies can_send_media_messages

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'
```

```
CREATOR = 'creator'
```

```
KICKED = 'kicked'
```

```
LEFT = 'left'
```

```
MEMBER = 'member'
```

```
RESTRICTED = 'restricted'
```

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

```
to_dict()
```

```
class telegram.ChatAction
```

```
Bases: object
```

Helper class to provide constants for different chatactions.

```
FIND_LOCATION = 'find_location'
```

```
RECORD_AUDIO = 'record_audio'
```

```
RECORD_VIDEO = 'record_video'
```

```
RECORD_VIDEO_NOTE = 'record_video_note'
```

```
TYPING = 'typing'
```

```
UPLOAD_AUDIO = 'upload_audio'
```

```
UPLOAD_DOCUMENT = 'upload_document'
```

```
UPLOAD_PHOTO = 'upload_photo'
```

```
UPLOAD_VIDEO = 'upload_video'
```

```
UPLOAD_VIDEO_NOTE = 'upload_video_note'
```

```
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
```

```
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.

inline_message_id
str – Optional. Identifier of the sent inline message.

query
str – The query that was used to obtain the result.

Parameters

- **result_id** (*str*) – The unique identifier for the result that was chosen.
- **from_user** (*telegram.User*) – The user that chose the result.
- **location** (*telegram.Location*, optional) – Sender location, only for bots that require user location.
- **inline_message_id** (*str*, optional) – Identifier of the sent inline message. Available only if there is an inline keyboard attached to the message. Will be also received in callback queries and can be used to edit the message.
- **query** (*str*) – The query that was used to obtain the result.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

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

to_dict ()

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
str – Unique identifier for this query.

from_user
telegram.User – Sender.

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

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.

data
str – Optional. Data associated with the callback button.

game_short_name

`str` – Optional. Short name of a Game to be returned.

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** (`*args, **kwargs`)

Shortcut for either:

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

or:

```
bot.edit_message_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 (`*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,
                             *args, **kwargs)
```

or:

```
bot.edit_message_reply_markup(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_text (*args, **kwargs)

Shortcut for either:

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

or:

```
bot.edit_message_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*

to_dict ()

class telegram.Contact (phone_number, first_name, last_name=None, user_id=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents a phone contact.

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.

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.
- ****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, **kwargs)
    Bases: telegram.base.TelegramObject
```

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

file_id
str – Unique file identifier.

thumb
telegram.PhotoSize – Optional. Document thumbnail.

file_name
str – Original filename.

mime_type
str – Optional. MIME type of the file.

file_size
int – Optional. File size.

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

```
classmethod de_json(data, bot)

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
str – Unique identifier for this file.

file_size
str – Optional. File size.

file_path
str – Optional. File path. Use *download* to get the file.

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.

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** (object, optional) – A file-like object. Must be opened 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).

Raises `ValueError` – If both `custom_path` and `out` are passed.

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

`True` – Shows reply interface to the user.

selective

`bool` – Optional. Force reply from specific users only.

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, callback_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

str – Label text on the button.

url

str – Optional. 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. 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.

switch_inline_query_current_chat

str – Optional. Will insert the bot's username and the specified inline query in the current chat's input field.

callback_game

telegram.CallbackGame – Optional. Description of the game that will be launched when the user presses the button.

pay

bool – Optional. Specify True, to send a Pay button.

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

List[List[*telegram.InlineKeyboardButton*]] – Array of button rows, each represented by an Array of `InlineKeyboardButton` objects.

Parameters

- **inline_keyboard** (List[List[*telegram.InlineKeyboardButton*]]) – Array of button rows, each represented by an Array of `InlineKeyboardButton` objects.
- ****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

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.

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*)

to_dict ()

class `telegram.InlineQueryResult` (*type*, *id*, ***kwargs*)

Bases: `telegram.base.TelegramObject`

Baseclass for the `InlineQueryResult*` classes.

type

str – Type of the result.

id

str – Unique identifier for this result, 1-64 Bytes.

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

str – Type of the result.

id

str – Unique identifier for this result, 1-64 Bytes.

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

str – ‘article’.

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.

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, **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
str – 'audio'.

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.

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.

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.
- **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, **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

str – ‘audio’.

id

str – Unique identifier for this result, 1-64 bytes.

audio_file_id

str – A valid file identifier for the audio file.

caption

str – Optional. Caption, 0-200 characters

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.

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, 0-200 characters

- **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,
                                                **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
str – 'document'.

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

title
str – Title for the result.

document_file_id
str – A valid file identifier for the file.

description
str – Optional. Short description of the result.

caption
str – Optional. Caption, 0-200 characters

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.

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, 0-200 characters
- **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, cap-
                                             tion=None, reply_markup=None, in-
                                             put_message_content=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

`str` – ‘gif’.

id

`str` – Unique identifier for this result, 1-64 bytes.

gif_file_id

`str` – A valid file identifier for the GIF file.

title

`str` – Optional. Title for the result.

caption

`str` – Optional. Caption, 0-200 characters

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.

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, 0-200 characters
- **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, **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

`str` – ‘mpeg4_gif’.

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, 0-200 characters

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.

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, 0-200 characters
- **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, description=None, caption=None, reply_markup=None, input_message_content=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

str – ‘photo’.

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, 0-200 characters

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.

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, 0-200 characters
- **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, reply_markup=None, input_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

str – ‘sticker’.

id

str – Unique identifier for this result, 1-64 bytes.

sticker_file_id

str – A valid file identifier of the sticker.

reply_markup

telegram.InlineKeyboardMarkup – Optional. Inline keyboard attached to the message.

input_message_content

telegram.InputMessageContent – Optional. Content of the message to be sent instead of the sticker.

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, **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

str – ‘video’.

id

str – Unique identifier for this result, 1-64 bytes.

video_file_id

str – A valid file identifier for the video file.

title

str – Title for the result.

description

str – Optional. Short description of the result.

caption

str – Optional. Caption, 0-200 characters.

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.

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, 0-200 characters.
- **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, **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

str – 'voice'.

id

str – Unique identifier for this result, 1-64 bytes.

voice_file_id

str – A valid file identifier for the voice message.

title

str – Voice message title.

caption

str – Optional. Caption, 0-200 characters.

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.

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, 0-200 characters.
- **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, **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

str – ‘contact’.

id

str – Unique identifier for this result, 1-64 bytes.

phone_number

str – Contact’s phone number.

first_name

str – Contact’s first name.

last_name

str – Optional. Contact’s last name.

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.

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.
- **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, **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
str – ‘document’.

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

title
str – Title for the result.

caption
str – Optional. Caption, 0-200 characters

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.

Parameters

- **id** (str) – Unique identifier for this result, 1-64 bytes.
- **title** (str) – Title for the result.
- **caption** (str, optional) – Caption, 0-200 characters
- **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, **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

str – ‘gif’.

id

str – Unique identifier for this result, 1-64 bytes.

gif_url

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

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, 0-200 characters

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.

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, 0-200 characters
- **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
str – 'location'.

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

latitude
float – Location latitude in degrees.

longitude
float – Location longitude in degrees.

title
str – Location title.

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.

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, **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

str – ‘mpeg4_gif’.

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, 0-200 characters

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.

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, 0-200 characters
- **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, **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

str – ‘photo’.

id

str – Unique identifier for this result, 1-64 bytes.

photo_url

str – A valid URL of the photo. Photo must be in jpeg format. Photo size must not exceed 5MB.

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, 0-200 characters

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.

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, 0-200 characters
- **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, 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

str – ‘venue’.

id

str – Unique identifier for this result, 1-64 Bytes.

latitude

float – Latitude of the venue location in degrees.

longitude

float – Longitude of the venue location in degrees.

title

str – Title of the venue.

address

str – Address of the venue.

foursquare_id

str – Optional. Foursquare identifier of the venue if known.

reply_markup

telegram.InlineKeyboardMarkup – Optional. Inline keyboard attached to the message.

input_message_content

telegram.InputMessageContent – Optional. Content of the message to be sent instead of the venue.

thumb_url

str – Optional. Url of the thumbnail for the result.

thumb_width

int – Optional. Thumbnail width.

thumb_height

int – Optional. Thumbnail height.

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.
- **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, **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

str – ‘video’.

id

str – Unique identifier for this result, 1-64 bytes.

video_url

str – A valid URL for the embedded video player or video file.

mime_type

str – Mime type of the content of 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, 0-200 characters

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.

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, 0-200 characters.
- **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, **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

str – ‘voice’.

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, 0-200 characters.

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.

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, 0-200 characters.
- **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

str – 'game'.

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.

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, **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

str – Contact's phone number.

first_name

str – Contact's first name.

last_name

str – Optional. Contact's last name.

Parameters

- **phone_number** (*str*) – Contact’s phone number.
- **first_name** (*str*) – Contact’s first name.
- **last_name** (*str*, optional) – Contact’s last name.
- ****kwargs** (*dict*) – Arbitrary keyword arguments.

class telegram.**InputFile** (*data*)

Bases: object

This object represents a Telegram InputFile.

data

dict – Data containing an inputfile.

Parameters **data** (*dict*) – Data containing an inputfile.

Raises TelegramError

content_type

str – Content type

headers

dict – Headers.

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*

static is_inputfile (*data*)

Check if the request is a file request.

Parameters **data** (*Dict[str, str]*) – A dict of (str, str) key/value pairs.

Returns *bool*

to_form ()

Transform the inputfile to multipart/form data.

Returns *str*

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

float – Latitude of the location in degrees.

longitude

float – Longitude of the location in degrees.

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

str – Text of the message to be sent, 1-4096 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 your bot's message.

disable_web_page_preview

bool – Optional. Disables link previews for links in the sent message.

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*, ***kwargs*)

Bases: telegram.inline.inputmessagecontent.InputMessageContent

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

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.

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.

- ****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

str – Text of the button.

request_contact

bool – Optional. If the user’s phone number will be sent.

request_location

bool – Optional. If the user’s current location will be sent.

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

float – Longitude as defined by sender.

latitude

float – Latitude as defined by sender.

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.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, new_chat_member=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, 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

int – Unique message identifier inside this chat.

from_user

[telegram.User](#) – Optional. Sender.

date

[datetime.datetime](#) – Date the message was sent.

chat

[telegram.Chat](#) – Conversation the message belongs to.

forward_from

[telegram.User](#) – Optional. Sender of the original message.

forward_from_chat

[telegram.Chat](#) – Optional. Information about the original channel.

forward_from_message_id

int – Optional. Identifier of the original message in the channel.

forward_date

[datetime.datetime](#) – Optional. Date the original message was sent.

reply_to_message

[telegram.Message](#) – Optional. The original message.

edit_date

[datetime.datetime](#) – Optional. Date the message was last edited.

media_group_id

str – Optional. The unique identifier of a media message group this message belongs to.

text

str – Optional. The actual UTF-8 text of the message.

entities

List[[telegram.MessageEntity](#)] – 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.

caption_entities

List[*telegram.MessageEntity*] – 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.

audio

telegram.Audio – Optional. Information about the file.

document

telegram.Document – Optional. Information about the file.

game

telegram.Game – Optional. Information about the game.

photo

List[*telegram.PhotoSize*] – Optional. Available sizes of the photo.

sticker

telegram.Sticker – Optional. Information about the sticker.

video

telegram.Video – Optional. Information about the video.

voice

telegram.Voice – Optional. Information about the file.

video_note

telegram.VideoNote – Optional. Information about the video message.

new_chat_members

List[*telegram.User*] – Optional. Information about new members to the chat. (the bot itself may be one of these members).

caption

str – Optional. Caption for the document, photo or video, 0-200 characters.

contact

telegram.Contact – Optional. Information about the contact.

location

telegram.Location – Optional. Information about the location.

venue

telegram.Venue – Optional. Information about the venue.

left_chat_member

telegram.User – Optional. Information about the user that left the group. (this member may be the bot itself).

new_chat_title

str – Optional. A chat title was changed to this value.

new_chat_photo

List[*telegram.PhotoSize*] – Optional. A chat photo was changed to this value.

delete_chat_photo

bool – Optional. The chat photo was deleted.

group_chat_created

bool – Optional. The group has been created.

supergroup_chat_created

bool – Optional. The supergroup has been created.

channel_chat_created

bool – Optional. The channel has been created.

migrate_to_chat_id

int – Optional. The group has been migrated to a supergroup with the specified identifier.

migrate_from_chat_id

`int` – Optional. The supergroup has been migrated from a group with the specified identifier.

pinned_message

`telegram.message` – Optional. Specified message was pinned.

invoice

`telegram.Invoice` – Optional. Information about the invoice.

successful_payment

`telegram.SuccessfulPayment` – Optional. Information about the payment.

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.

bot

`telegram.Bot` – Optional. The Bot to use for instance methods.

Deprecated: 6.0 `new_chat_member` (`telegram.User`): Replaced with `new_chat_members`

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.
- **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, 0-200 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 `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.
- **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.

chat_id

int – Shortcut for *telegram.Chat.id* for *chat*.

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_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.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`

new_chat_member

Deprecated

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_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

`str` – Type of the entity.

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. Url that will be opened after user taps on the text.

user

telegram.User – Optional. The mentioned 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', 'bot_command', 'url', 'email', 'bold', 'italic',`

`BOLD = 'bold'`

```
BOT_COMMAND = 'bot_command'
CODE = 'code'
EMAIL = 'email'
HASHTAG = 'hashtag'
ITALIC = 'italic'
MENTION = 'mention'
PRE = 'pre'
TEXT_LINK = 'text_link'
TEXT_MENTION = 'text_mention'
URL = 'url'

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'
    MARKDOWN = 'Markdown'

class telegram.PhotoSize(file_id, width, height, file_size=None, **kwargs)
    Bases: telegram.base.TelegramObject

    This object represents one size of a photo or a file/sticker thumbnail.

    file_id
        str – Unique identifier for this file.

    width
        int – Photo width.

    height
        int – Photo height.

    file_size
        int – Optional. File size.
```

Parameters

- **file_id** (str) – Unique identifier for this file.
- **width** (int) – Photo width.
- **height** (int) – Photo height.
- **file_size** (int, optional) – File size.
- ****kwargs** (dict) – Arbitrary keyword arguments.

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

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

`True` – Requests clients to remove the custom keyboard.

selective

`bool` – Optional. Use this parameter if you want to remove the keyboard for specific users only.

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

`List[List[telegram.KeyboardButton | str]]` – Array of button rows.

resize_keyboard

`bool` – Optional. Requests clients to resize the keyboard.

one_time_keyboard

`bool` – Optional. Requests clients to hide the keyboard as soon as it's been used.

selective

`bool` – Optional. Show the keyboard to specific users only.

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.

`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`, ****kwargs**)

Bases: `telegram.base.TelegramObject`

This object represents a sticker.

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.

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.

```
classmethod de_json (data, bot)
classmethod de_list (data, bot)
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
int – The update’s unique identifier.

message
telegram.Message – Optional. New incoming message.

edited_message
telegram.Message – Optional. New version of a message.

channel_post
telegram.Message – Optional. New incoming channel post.

edited_channel_post
telegram.Message – Optional. New version of a channel post.

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.

callback_query
telegram.CallbackQuery – Optional. New incoming callback query.

shipping_query
telegram.ShippingQuery – Optional. New incoming shipping query.

pre_checkout_query
telegram.PreCheckoutQuery – Optional. New incoming pre-checkout query.

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

telegram.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*.

effective_message

telegram.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*.

effective_user

telegram.User – The user that sent this update, no matter what kind of update this is. Will be None for *channel_post*.

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

`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.

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

str – Convenience property. The user’s *first_name*, followed by (if available) *last_name*.

get_profile_photos (**args*, ***kwargs*)

Shortcut for:

```
bot.get_user_profile_photos(update.message.from_user.id, *args, **kwargs)
```

mention_html (*name=None*)

Parameters **name** (*str*) – If provided, will overwrite the user’s name.

Returns The inline mention for the user as HTML.

Return type *str*

mention_markdown (*name=None*)

Parameters **name** (*str*) – If provided, will overwrite the user’s name.

Returns The inline mention for the user as markdown.

Return type *str*

name

str – Convenience property. If available, returns the user’s *username* prefixed with “@”. If *username* is not available, returns *full_name*.

class telegram.UserProfilePhotos (*total_count*, *photos*, ***kwargs*)

Bases: *telegram.base.TelegramObject*

This object represent a user’s profile pictures.

total_count

int – Total number of profile pictures.

photos

List[List[*telegram.PhotoSize*]] – Requested profile pictures.

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, **kwargs)
```

Bases: telegram.base.TelegramObject

This object represents a venue.

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.

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.
- ****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, **kwargs)
```

Bases: telegram.base.TelegramObject

This object represents a video file.

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.

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

classmethod de_json (*data, bot*)

class telegram.Voice (*file_id, duration, mime_type=None, file_size=None, **kwargs*)

Bases: telegram.base.TelegramObject

This object represents a voice note.

file_id
str – Unique identifier for this file.

duration
int – Duration of the audio in seconds as defined by sender.

mime_type
str – Optional. MIME type of the file as defined by sender.

file_size
int – Optional. File size.

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

classmethod de_json (*data, bot*)

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
str – Webhook URL.

has_custom_certificate
bool – If a custom certificate was provided for webhook.

pending_update_count
int – Number of updates awaiting delivery.

last_error_date
int – Optional. Unix time for the most recent error that happened.

last_error_message
str – Optional. Error message in human-readable format.

max_connections
int – Optional. Maximum allowed number of simultaneous HTTPS connections.

allowed_updates
List[str] – Optional. A list of update types the bot is subscribed to.

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, 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

str – Unique file identifier.

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.

Parameters

- **file_id** (str) – Unique file identifier.
- **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

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`.

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.

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 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`

`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

`int` – Position in high score table for the game.

user

`telegram.User` – User.

score

`int` – Score.

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, **kwargs*)

Bases: `telegram.base.TelegramObject`

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

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.

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

classmethod `de_json` (*data, bot*)

class `telegram.LabeledPrice` (*label, amount, **kwargs*)

Bases: `telegram.base.TelegramObject`

This object represents a portion of the price for goods or services.

label

str – Portion label.

amount

int – Price of the product in the smallest units of the currency.

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

str – Three-letter ISO 4217 currency code.

total_amount

int – Total price in the smallest units of the currency.

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.

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 keywords arguments.

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

```
class telegram.ShippingOption (id, title, prices, **kwargs)
```

Bases: telegram.base.TelegramObject

This object represents one shipping option.

id

str – Shipping option identifier.

title

str – Option title.

prices

List[telegram.LabeledPrice] – List of price portions.

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

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.

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

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.

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.

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.

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

to_dict ()

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

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.

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

str – Product name.

description

str – Product description.

start_parameter

str – Unique bot deep-linking parameter.

currency

str – Three-letter ISO 4217 currency code.

total_amount

int – Total price in the smallest units of the currency.

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.

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

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

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.

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.

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

to_dict ()

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

str – Unique file identifier of small (160x160) chat photo.

big_file_id

str – Unique file identifier of big (640x640) chat photo.

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

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.

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

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.

y_shift

float – Shift by Y-axis measured in heights of the mask scaled to the face size, from top to bottom.

scale

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

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'
```

```
EYES = 'eyes'
```

```
FOREHEAD = 'forehead'
```

```
MOUTH = 'mouth'
```

```
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.InputMediaPhoto](#) and [telegram.InputMediaVideo](#) for detailed use.

```
class telegram.InputMediaPhoto(media, caption=None)
```

Bases: telegram.files.inputmedia.InputMedia

Represents a photo to be sent.

type

str – photo.

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, 0-200 characters.

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, 0-200 characters.

Note: At the moment using a new file is not yet supported.

class `telegram.InputMediaVideo` (*media*, *caption=None*, *width=None*, *height=None*, *duration=None*)

Bases: `telegram.files.inputmedia.InputMedia`

Represents a video to be sent.

type

`str` – video.

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, 0-200 characters.

width

`int` – Optional. Video width.

height

`int` – Optional. Video height.

duration

`int` – Optional. Video duration.

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, 0-200 characters.
- **width** (`int`, optional) – Video width.
- **height** (`int`, optional) – Video height.
- **duration** (`int`, optional) – Video duration.

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. At the moment using a new file is not yet supported.

CHAPTER 2

Changelog

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`

t

- `telegram.constants`, [71](#)
- `telegram.error`, [73](#)
- `telegram.ext.filters`, [8](#)
- `telegram.utils.helpers`, [3](#)

Symbols

`__call__()` (telegram.ext.DelayQueue method), 17
`__call__()` (telegram.ext.MessageQueue method), 15
`__weakref__` (telegram.ext.MessageQueue attribute), 16
`_queue` (telegram.ext.JobQueue attribute), 13

A

`add_error_handler()` (telegram.ext.Dispatcher method), 6
`add_handler()` (telegram.ext.Dispatcher method), 6
`add_sticker_to_set()` (telegram.Bot method), 35
`address` (telegram.InlineQueryResultVenue attribute), 119, 188
`address` (telegram.InputVenueMessageContent attribute), 123, 193
`address` (telegram.Venue attribute), 97, 212
`ADMINISTRATOR` (telegram.ChatMember attribute), 71
`all` (telegram.ext.filters.Filters attribute), 9
`all_members_are_administrators` (telegram.Chat attribute), 66, 162
`ALL_TYPES` (telegram.MessageEntity attribute), 91
`allow_edited` (telegram.ext.CommandHandler attribute), 23
`allow_edited` (telegram.ext.MessageHandler attribute), 26
`allow_reentry` (telegram.ext.ConversationHandler attribute), 21
`allowed_updates` (telegram.WebhookInfo attribute), 99, 213
`amount` (telegram.LabeledPrice attribute), 124, 217
`Animation` (class in telegram), 131
`animation` (telegram.Game attribute), 130, 215
`answer()` (telegram.CallbackQuery method), 65
`answer()` (telegram.InlineQuery method), 102
`answer()` (telegram.PreCheckoutQuery method), 129
`answer()` (telegram.ShippingQuery method), 128
`answer_callback_query()` (telegram.Bot method), 35
`answer_inline_query()` (telegram.Bot method), 36
`answer_pre_checkout_query()` (telegram.Bot method), 37
`answer_shipping_query()` (telegram.Bot method), 37

`Audio` (class in telegram), 34
`audio` (telegram.ext.filters.Filters attribute), 9
`audio` (telegram.Message attribute), 81, 196
`audio_duration` (telegram.InlineQueryResultAudio attribute), 105, 175
`audio_file_id` (telegram.InlineQueryResultCachedAudio attribute), 105, 176
`audio_url` (telegram.InlineQueryResultAudio attribute), 104, 175
`author_signature` (telegram.Message attribute), 82, 197

B

`BadRequest`, 73
`BaseFilter` (class in telegram.ext.filters), 8
`big_file_id` (telegram.ChatPhoto attribute), 71, 222
`BOLD` (telegram.MessageEntity attribute), 91
`Bot` (class in telegram), 35
`bot` (telegram.ext.Dispatcher attribute), 6
`bot` (telegram.ext.JobQueue attribute), 13
`bot` (telegram.ext.Updater attribute), 4
`bot` (telegram.Message attribute), 82, 197
`bot` (telegram.PreCheckoutQuery attribute), 129, 219
`bot` (telegram.ShippingQuery attribute), 128, 221
`bot` (telegram.User attribute), 96, 210
`BOT_COMMAND` (telegram.MessageEntity attribute), 91
`burst_limit` (telegram.ext.DelayQueue attribute), 16

C

`callback` (telegram.ext.CallbackQueryHandler attribute), 18
`callback` (telegram.ext.ChosenInlineResultHandler attribute), 20
`callback` (telegram.ext.CommandHandler attribute), 23
`callback` (telegram.ext.Handler attribute), 17
`callback` (telegram.ext.InlineQueryHandler attribute), 24
`callback` (telegram.ext.Job attribute), 12
`callback` (telegram.ext.MessageHandler attribute), 26
`callback` (telegram.ext.PreCheckoutQueryHandler attribute), 27
`callback` (telegram.ext.RegexHandler attribute), 28
`callback` (telegram.ext.ShippingQueryHandler attribute), 30

`callback` (`telegram.ext.StringCommandHandler` attribute), 31

`callback` (`telegram.ext.StringRegexHandler` attribute), 32

`callback` (`telegram.ext.TypeHandler` attribute), 33

`callback_data` (`telegram.InlineKeyboardButton` attribute), 75, 171

`callback_game` (`telegram.InlineKeyboardButton` attribute), 75, 172

`callback_query` (`telegram.Update` attribute), 94, 209

`CallbackGame` (class in `telegram`), 132

`CallbackQuery` (class in `telegram`), 64

`CallbackQueryHandler` (class in `telegram.ext`), 18

`can_add_web_page_previews` (`telegram.ChatMember` attribute), 70, 165

`can_be_edited` (`telegram.ChatMember` attribute), 69, 164

`can_change_info` (`telegram.ChatMember` attribute), 69, 164

`can_delete_messages` (`telegram.ChatMember` attribute), 70, 165

`can_edit_messages` (`telegram.ChatMember` attribute), 69, 165

`can_invite_users` (`telegram.ChatMember` attribute), 70, 165

`can_pin_messages` (`telegram.ChatMember` attribute), 70, 165

`can_post_messages` (`telegram.ChatMember` attribute), 69, 164

`can_promote_members` (`telegram.ChatMember` attribute), 70, 165

`can_restrict_members` (`telegram.ChatMember` attribute), 70, 165

`can_send_media_messages` (`telegram.ChatMember` attribute), 70, 165

`can_send_messages` (`telegram.ChatMember` attribute), 70, 165

`can_send_other_messages` (`telegram.ChatMember` attribute), 70, 165

`can_set_sticker_set` (`telegram.Chat` attribute), 66, 162

`caption` (`telegram.InlineQueryResultAudio` attribute), 105, 176

`caption` (`telegram.InlineQueryResultCachedAudio` attribute), 105, 176

`caption` (`telegram.InlineQueryResultCachedDocument` attribute), 106, 177

`caption` (`telegram.InlineQueryResultCachedGif` attribute), 107, 178

`caption` (`telegram.InlineQueryResultCachedMpeg4Gif` attribute), 108, 178

`caption` (`telegram.InlineQueryResultCachedPhoto` attribute), 109, 179

`caption` (`telegram.InlineQueryResultCachedVideo` attribute), 110, 181

`caption` (`telegram.InlineQueryResultCachedVoice` attribute), 111, 181

`caption` (`telegram.InlineQueryResultDocument` attribute), 113, 183

`caption` (`telegram.InlineQueryResultGif` attribute), 115, 184

`caption` (`telegram.InlineQueryResultMpeg4Gif` attribute), 117, 186

`caption` (`telegram.InlineQueryResultPhoto` attribute), 118, 187

`caption` (`telegram.InlineQueryResultVideo` attribute), 120, 189

`caption` (`telegram.InlineQueryResultVoice` attribute), 121, 190

`caption` (`telegram.InputMediaPhoto` attribute), 77, 224

`caption` (`telegram.InputMediaVideo` attribute), 78, 224

`caption` (`telegram.Message` attribute), 81, 196

`caption_entities` (`telegram.Message` attribute), 81, 195

`CHANNEL` (`telegram.Chat` attribute), 67

`channel_chat_created` (`telegram.Message` attribute), 81, 196

`channel_post` (`telegram.Update` attribute), 94, 209

`channel_post_updates` (`telegram.ext.MessageHandler` attribute), 26

`Chat` (class in `telegram`), 66

`chat` (`telegram.Message` attribute), 80, 195

`chat_created` (`telegram.ext.filters.Filters` attribute), 10

`chat_data` (`telegram.ext.Dispatcher` attribute), 7

`chat_id` (`telegram.Message` attribute), 84

`chat_instance` (`telegram.CallbackQuery` attribute), 64, 167

`ChatAction` (class in `telegram`), 68

`ChatMember` (class in `telegram`), 69

`ChatMigrated`, 73

`ChatPhoto` (class in `telegram`), 71

`check_update()` (`telegram.ext.CallbackQueryHandler` method), 19

`check_update()` (`telegram.ext.ChosenInlineResultHandler` method), 20

`check_update()` (`telegram.ext.CommandHandler` method), 24

`check_update()` (`telegram.ext.ConversationHandler` method), 22

`check_update()` (`telegram.ext.Handler` method), 18

`check_update()` (`telegram.ext.InlineQueryHandler` method), 25

`check_update()` (`telegram.ext.MessageHandler` method), 27

`check_update()` (`telegram.ext.PreCheckoutQueryHandler` method), 28

`check_update()` (`telegram.ext.RegexHandler` method), 30

`check_update()` (`telegram.ext.ShippingQueryHandler` method), 31

`check_update()` (`telegram.ext.StringCommandHandler` method), 32

`check_update()` (`telegram.ext.StringRegexHandler` method), 33

`check_update()` (`telegram.ext.TypeHandler` method), 34

`CHIN` (`telegram.MaskPosition` attribute), 102

chosen_inline_result (telegram.Update attribute), 94, 209

ChosenInlineResult (class in telegram), 124

ChosenInlineResultHandler (class in telegram.ext), 20

city (telegram.ShippingAddress attribute), 125, 218

CODE (telegram.MessageEntity attribute), 91

collect_optional_args() (telegram.ext.Handler method), 18

command (telegram.ext.CommandHandler attribute), 23

command (telegram.ext.filters.Filters attribute), 9

command (telegram.ext.StringCommandHandler attribute), 31

CommandHandler (class in telegram.ext), 23

Contact (class in telegram), 72

contact (telegram.ext.filters.Filters attribute), 9

contact (telegram.Message attribute), 81, 196

contains_masks (telegram.StickerSet attribute), 101, 222

content_type (telegram.InputFile attribute), 77

context (telegram.ext.Job attribute), 12

ConversationHandler (class in telegram.ext), 21

country_code (telegram.ShippingAddress attribute), 125, 218

create_new_sticker_set() (telegram.Bot method), 38

CREATOR (telegram.ChatMember attribute), 71

currency (telegram.Invoice attribute), 125, 220

currency (telegram.PreCheckoutQuery attribute), 129, 219

currency (telegram.SuccessfulPayment attribute), 127, 217

description (telegram.InlineQueryResultCachedPhoto attribute), 109, 179

description (telegram.InlineQueryResultCachedVideo attribute), 110, 181

description (telegram.InlineQueryResultDocument attribute), 113, 183

description (telegram.InlineQueryResultPhoto attribute), 118, 187

description (telegram.InlineQueryResultVideo attribute), 120, 189

description (telegram.Invoice attribute), 125, 220

disable_web_page_preview (telegram.InputTextMessageContent attribute), 122, 193

dispatch_error() (telegram.ext.Dispatcher method), 7

Dispatcher (class in telegram.ext), 6

dispatcher (telegram.ext.Updater attribute), 4

Document (class in telegram), 73

document (telegram.ext.filters.Filters attribute), 9

document (telegram.Message attribute), 81, 196

document_file_id (telegram.InlineQueryResultCachedDocument attribute), 106, 177

document_url (telegram.InlineQueryResultDocument attribute), 113, 183

download() (telegram.File method), 74

duration (telegram.Audio attribute), 34, 132

duration (telegram.InputMediaVideo attribute), 78, 224

duration (telegram.Video attribute), 97, 212

duration (telegram.VideoNote attribute), 98, 216

duration (telegram.Voice attribute), 99, 213

D

data (telegram.CallbackQuery attribute), 64, 167

data (telegram.InputFile attribute), 76, 192

date (telegram.Message attribute), 80, 195

days (telegram.ext.Job attribute), 12

de_json() (telegram.Update class method), 95

de_json() (telegram.User class method), 96

de_list() (telegram.User class method), 96

DelayQueue (class in telegram.ext), 16

delete() (telegram.Message method), 84

delete_chat_photo (telegram.ext.filters.Filters attribute), 10

delete_chat_photo (telegram.Message attribute), 81, 196

delete_chat_photo() (telegram.Bot method), 39

delete_chat_sticker_set() (telegram.Bot method), 39

delete_message() (telegram.Bot method), 39

delete_sticker_from_set() (telegram.Bot method), 40

delete_webhook() (telegram.Bot method), 40

description (telegram.Chat attribute), 66, 162

description (telegram.Game attribute), 130, 214

description (telegram.InlineQueryResultArticle attribute), 104, 175

description (telegram.InlineQueryResultCachedDocument attribute), 106, 177

E

edit_caption() (telegram.Message method), 84

edit_date (telegram.Message attribute), 80, 195

edit_message_caption() (telegram.Bot method), 40

edit_message_caption() (telegram.CallbackQuery method), 65

edit_message_live_location() (telegram.Bot method), 41

edit_message_reply_markup() (telegram.Bot method), 41

edit_message_reply_markup() (telegram.CallbackQuery method), 65

edit_message_text() (telegram.Bot method), 42

edit_message_text() (telegram.CallbackQuery method), 65

edit_reply_markup() (telegram.Message method), 84

edit_text() (telegram.Message method), 85

edited_channel_post (telegram.Update attribute), 94, 209

edited_message (telegram.Update attribute), 94, 209

edited_updates (telegram.ext.MessageHandler attribute), 26

effective_attachment (telegram.Message attribute), 85

effective_chat (telegram.Update attribute), 95

effective_message (telegram.Update attribute), 95

effective_user (telegram.Update attribute), 95

EMAIL (telegram.MessageEntity attribute), 91
email (telegram.OrderInfo attribute), 126, 220
emoji (telegram.Sticker attribute), 100, 208
enabled (telegram.ext.Job attribute), 12
END (telegram.ext.ConversationHandler attribute), 22
entities (telegram.Message attribute), 80, 195
entry_points (telegram.ext.ConversationHandler attribute), 21
error_handlers (telegram.ext.Dispatcher attribute), 7
escape_markdown() (in module telegram.utils.helpers), 3
exc_route (telegram.ext.DelayQueue attribute), 16
export_chat_invite_link() (telegram.Bot method), 42
EYES (telegram.MaskPosition attribute), 102

F

fallbacks (telegram.ext.ConversationHandler attribute), 21
File (class in telegram), 74
file_id (telegram.Animation attribute), 131, 214
file_id (telegram.Audio attribute), 34, 132
file_id (telegram.Document attribute), 73, 170
file_id (telegram.File attribute), 74, 170
file_id (telegram.PhotoSize attribute), 92, 206
file_id (telegram.Sticker attribute), 100, 208
file_id (telegram.Video attribute), 97, 212
file_id (telegram.VideoNote attribute), 98, 216
file_id (telegram.Voice attribute), 98, 213
file_name (telegram.Animation attribute), 131, 214
file_name (telegram.Document attribute), 73, 170
file_path (telegram.File attribute), 74, 170
file_size (telegram.Animation attribute), 131, 214
file_size (telegram.Audio attribute), 34, 133
file_size (telegram.Document attribute), 73, 170
file_size (telegram.File attribute), 74, 170
file_size (telegram.PhotoSize attribute), 92, 206
file_size (telegram.Sticker attribute), 100, 208
file_size (telegram.Video attribute), 98, 212
file_size (telegram.VideoNote attribute), 98, 216
file_size (telegram.Voice attribute), 99, 213
filter() (telegram.ext.filters.BaseFilter method), 8
Filters (class in telegram.ext.filters), 9
filters (telegram.ext.CommandHandler attribute), 23
filters (telegram.ext.MessageHandler attribute), 26
Filters.chat (class in telegram.ext.filters), 9
Filters.entity (class in telegram.ext.filters), 9
Filters.language (class in telegram.ext.filters), 10
Filters.user (class in telegram.ext.filters), 11
FIND_LOCATION (telegram.ChatAction attribute), 69
first_name (telegram.Bot attribute), 42
first_name (telegram.Chat attribute), 66, 162
first_name (telegram.Contact attribute), 72, 169
first_name (telegram.InlineQueryResultContact attribute), 112, 182
first_name (telegram.InputContactMessageContent attribute), 123, 191
first_name (telegram.User attribute), 95, 210
force_reply (telegram.ForceReply attribute), 75, 171

ForceReply (class in telegram), 75
FOREHEAD (telegram.MaskPosition attribute), 102
forward() (telegram.Message method), 85
forward_date (telegram.Message attribute), 80, 195
forward_from (telegram.Message attribute), 80, 195
forward_from_chat (telegram.Message attribute), 80, 195
forward_from_message_id (telegram.Message attribute), 80, 195
forward_message() (telegram.Bot method), 42
forward_signature (telegram.Message attribute), 82, 197
forwarded (telegram.ext.filters.Filters attribute), 9
foursquare_id (telegram.InlineQueryResultVenue attribute), 119, 188
foursquare_id (telegram.InputVenueMessageContent attribute), 123, 193
foursquare_id (telegram.Venue attribute), 97, 212
from_timestamp() (in module telegram.utils.helpers), 3
from_user (telegram.CallbackQuery attribute), 64, 167
from_user (telegram.ChosenInlineResult attribute), 124, 166
from_user (telegram.InlineQuery attribute), 102, 173
from_user (telegram.Message attribute), 80, 195
from_user (telegram.PreCheckoutQuery attribute), 129, 219
from_user (telegram.ShippingQuery attribute), 128, 221
full_name (telegram.User attribute), 96

G

Game (class in telegram), 130
game (telegram.ext.filters.Filters attribute), 9
game (telegram.Message attribute), 81, 196
game_short_name (telegram.CallbackQuery attribute), 64, 167
game_short_name (telegram.InlineQueryResultGame attribute), 114, 191
GameHighScore (class in telegram), 132
get_administrators() (telegram.Chat method), 67
get_chat() (telegram.Bot method), 43
get_chat_administrators() (telegram.Bot method), 43
get_chat_member() (telegram.Bot method), 43
get_chat_members_count() (telegram.Bot method), 44
get_file() (telegram.Bot method), 44
get_game_high_scores() (telegram.Bot method), 44
get_instance() (telegram.ext.Dispatcher class method), 7
get_me() (telegram.Bot method), 45
get_member() (telegram.Chat method), 68
get_members_count() (telegram.Chat method), 68
get_profile_photos() (telegram.User method), 96
get_signal_name() (in module telegram.utils.helpers), 3
get_sticker_set() (telegram.Bot method), 45
get_updates() (telegram.Bot method), 45
get_user_profile_photos() (telegram.Bot method), 46
get_webhook_info() (telegram.Bot method), 46

gif_duration (telegram.InlineQueryResultGif attribute), 115, 184
 gif_file_id (telegram.InlineQueryResultCachedGif attribute), 107, 178
 gif_height (telegram.InlineQueryResultGif attribute), 114, 184
 gif_url (telegram.InlineQueryResultGif attribute), 114, 184
 gif_width (telegram.InlineQueryResultGif attribute), 114, 184
 GROUP (telegram.Chat attribute), 67
 group (telegram.ext.filters.Filters attribute), 10
 group_chat_created (telegram.Message attribute), 81, 196
 groups (telegram.ext.Dispatcher attribute), 7

H

handle_update() (telegram.ext.CallbackQueryHandler method), 19
 handle_update() (telegram.ext.ChosenInlineResultHandler method), 20
 handle_update() (telegram.ext.CommandHandler method), 24
 handle_update() (telegram.ext.ConversationHandler method), 22
 handle_update() (telegram.ext.Handler method), 18
 handle_update() (telegram.ext.InlineQueryHandler method), 25
 handle_update() (telegram.ext.MessageHandler method), 27
 handle_update() (telegram.ext.PreCheckoutQueryHandler method), 28
 handle_update() (telegram.ext.RegexHandler method), 30
 handle_update() (telegram.ext.ShippingQueryHandler method), 31
 handle_update() (telegram.ext.StringCommandHandler method), 32
 handle_update() (telegram.ext.StringRegexHandler method), 33
 handle_update() (telegram.ext.TypeHandler method), 34
 Handler (class in telegram.ext), 17
 handlers (telegram.ext.Dispatcher attribute), 7
 has_custom_certificate (telegram.WebhookInfo attribute), 99, 213
 HASHTAG (telegram.MessageEntity attribute), 91
 headers (telegram.InputFile attribute), 77
 height (telegram.InputMediaVideo attribute), 78, 224
 height (telegram.PhotoSize attribute), 92, 206
 height (telegram.Sticker attribute), 100, 208
 height (telegram.Video attribute), 97, 212
 hide_url (telegram.InlineQueryResultArticle attribute), 104, 175
 HTML (telegram.ParseMode attribute), 91

I

id (telegram.Bot attribute), 46
 id (telegram.CallbackQuery attribute), 64, 167
 id (telegram.Chat attribute), 66, 162
 id (telegram.InlineQuery attribute), 102, 173
 id (telegram.InlineQueryResult attribute), 103, 174
 id (telegram.InlineQueryResultArticle attribute), 103, 174
 id (telegram.InlineQueryResultAudio attribute), 104, 175
 id (telegram.InlineQueryResultCachedAudio attribute), 105, 176
 id (telegram.InlineQueryResultCachedDocument attribute), 106, 177
 id (telegram.InlineQueryResultCachedGif attribute), 107, 178
 id (telegram.InlineQueryResultCachedMpeg4Gif attribute), 108, 178
 id (telegram.InlineQueryResultCachedPhoto attribute), 109, 179
 id (telegram.InlineQueryResultCachedSticker attribute), 109, 180
 id (telegram.InlineQueryResultCachedVideo attribute), 110, 180
 id (telegram.InlineQueryResultCachedVoice attribute), 111, 181
 id (telegram.InlineQueryResultContact attribute), 112, 182
 id (telegram.InlineQueryResultDocument attribute), 113, 183
 id (telegram.InlineQueryResultGame attribute), 114, 191
 id (telegram.InlineQueryResultGif attribute), 114, 184
 id (telegram.InlineQueryResultLocation attribute), 115, 185
 id (telegram.InlineQueryResultMpeg4Gif attribute), 117, 186
 id (telegram.InlineQueryResultPhoto attribute), 118, 187
 id (telegram.InlineQueryResultVenue attribute), 119, 188
 id (telegram.InlineQueryResultVideo attribute), 120, 189
 id (telegram.InlineQueryResultVoice attribute), 121, 190
 id (telegram.PreCheckoutQuery attribute), 129, 219
 id (telegram.ShippingOption attribute), 126, 218
 id (telegram.ShippingQuery attribute), 128, 221
 id (telegram.User attribute), 95, 210
 idle() (telegram.ext.Updater method), 5
 inline_keyboard (telegram.InlineKeyboardMarkup attribute), 76, 172
 inline_message_id (telegram.CallbackQuery attribute), 64, 167
 inline_message_id (telegram.ChosenInlineResult attribute), 124, 167
 inline_query (telegram.Update attribute), 94, 209
 InlineKeyboardButton (class in telegram), 75

- InlineKeyboardMarkup (class in telegram), 76
 - InlineQuery (class in telegram), 102
 - InlineQueryHandler (class in telegram.ext), 24
 - InlineQueryResult (class in telegram), 103
 - InlineQueryResultArticle (class in telegram), 103
 - InlineQueryResultAudio (class in telegram), 104
 - InlineQueryResultCachedAudio (class in telegram), 105
 - InlineQueryResultCachedDocument (class in telegram), 106
 - InlineQueryResultCachedGif (class in telegram), 107
 - InlineQueryResultCachedMpeg4Gif (class in telegram), 108
 - InlineQueryResultCachedPhoto (class in telegram), 108
 - InlineQueryResultCachedSticker (class in telegram), 109
 - InlineQueryResultCachedVideo (class in telegram), 110
 - InlineQueryResultCachedVoice (class in telegram), 111
 - InlineQueryResultContact (class in telegram), 112
 - InlineQueryResultDocument (class in telegram), 113
 - InlineQueryResultGame (class in telegram), 114
 - InlineQueryResultGif (class in telegram), 114
 - InlineQueryResultLocation (class in telegram), 115
 - InlineQueryResultMpeg4Gif (class in telegram), 116
 - InlineQueryResultPhoto (class in telegram), 117
 - InlineQueryResultVenue (class in telegram), 119
 - InlineQueryResultVideo (class in telegram), 120
 - InlineQueryResultVoice (class in telegram), 121
 - input_message_content (telegram.InlineQueryResultArticle attribute), 103, 174
 - input_message_content (telegram.InlineQueryResultAudio attribute), 105, 176
 - input_message_content (telegram.InlineQueryResultCachedAudio attribute), 106, 176
 - input_message_content (telegram.InlineQueryResultCachedDocument attribute), 106, 177
 - input_message_content (telegram.InlineQueryResultCachedGif attribute), 107, 178
 - input_message_content (telegram.InlineQueryResultCachedMpeg4Gif attribute), 108, 179
 - input_message_content (telegram.InlineQueryResultCachedPhoto attribute), 109, 179
 - input_message_content (telegram.InlineQueryResultCachedSticker attribute), 110, 180
 - input_message_content (telegram.InlineQueryResultCachedVideo attribute), 110, 181
 - input_message_content (telegram.InlineQueryResultCachedVoice attribute), 111, 181
 - input_message_content (telegram.InlineQueryResultContact attribute), 112, 182
 - input_message_content (telegram.InlineQueryResultDocument attribute), 113, 183
 - input_message_content (telegram.InlineQueryResultGif attribute), 115, 184
 - input_message_content (telegram.InlineQueryResultLocation attribute), 116, 185
 - input_message_content (telegram.InlineQueryResultMpeg4Gif attribute), 117, 186
 - input_message_content (telegram.InlineQueryResultPhoto attribute), 118, 187
 - input_message_content (telegram.InlineQueryResultVenue attribute), 119, 188
 - input_message_content (telegram.InlineQueryResultVideo attribute), 120, 190
 - input_message_content (telegram.InlineQueryResultVoice attribute), 121, 190
 - InputContactMessageContent (class in telegram), 123
 - InputFile (class in telegram), 76
 - InputLocationMessageContent (class in telegram), 122
 - InputMedia (class in telegram), 77
 - InputMediaPhoto (class in telegram), 77
 - InputMediaVideo (class in telegram), 78
 - InputMessageContent (class in telegram), 122
 - InputTextMessageContent (class in telegram), 122
 - InputVenueMessageContent (class in telegram), 123
 - interval (telegram.ext.Job attribute), 12
 - interval_seconds (telegram.ext.Job attribute), 12
 - InvalidToken, 73
 - InvertedFilter (class in telegram.ext.filters), 11
 - invite_link (telegram.Chat attribute), 66, 162
 - Invoice (class in telegram), 125
 - invoice (telegram.ext.filters.Filters attribute), 10
 - invoice (telegram.Message attribute), 82, 197
 - invoice_payload (telegram.PreCheckoutQuery attribute), 129, 219
 - invoice_payload (telegram.ShippingQuery attribute), 128, 221
 - invoice_payload (telegram.SuccessfulPayment attribute), 127, 217
 - is_bot (telegram.User attribute), 95, 210
 - is_image() (telegram.InputFile static method), 77
 - is_inputfile() (telegram.InputFile static method), 77
 - ITALIC (telegram.MessageEntity attribute), 91
- ## J
- Job (class in telegram.ext), 11

job_queue (telegram.ext.Dispatcher attribute), 6
 job_queue (telegram.ext.Job attribute), 12
 job_queue (telegram.ext.Updater attribute), 4
 JobQueue (class in telegram.ext), 13
 jobs() (telegram.ext.JobQueue method), 13

K

keyboard (telegram.ReplyKeyboardMarkup attribute), 93, 207
 KeyboardButton (class in telegram), 78
 kick_chat_member() (telegram.Bot method), 46
 kick_member() (telegram.Chat method), 68
 KICKED (telegram.ChatMember attribute), 71

L

label (telegram.LabeledPrice attribute), 124, 216
 LabeledPrice (class in telegram), 124
 language_code (telegram.User attribute), 96, 210
 last_error_date (telegram.WebhookInfo attribute), 99, 213
 last_error_message (telegram.WebhookInfo attribute), 99, 213
 last_name (telegram.Bot attribute), 47
 last_name (telegram.Chat attribute), 66, 162
 last_name (telegram.Contact attribute), 72, 169
 last_name (telegram.InlineQueryResultContact attribute), 112, 182
 last_name (telegram.InputContactMessageContent attribute), 123, 191
 last_name (telegram.User attribute), 95, 210
 latitude (telegram.InlineQueryResultLocation attribute), 115, 185
 latitude (telegram.InlineQueryResultVenue attribute), 119, 188
 latitude (telegram.InputLocationMessageContent attribute), 122, 192
 latitude (telegram.InputVenueMessageContent attribute), 123, 193
 latitude (telegram.Location attribute), 79, 194
 leave() (telegram.Chat method), 68
 leave_chat() (telegram.Bot method), 47
 LEFT (telegram.ChatMember attribute), 71
 left_chat_member (telegram.ext.filters.Filters attribute), 10
 left_chat_member (telegram.Message attribute), 81, 196
 length (telegram.MessageEntity attribute), 90, 205
 length (telegram.VideoNote attribute), 98, 216
 live_period (telegram.InlineQueryResultLocation attribute), 116, 185
 Location (class in telegram), 79
 location (telegram.ChosenInlineResult attribute), 124, 167
 location (telegram.ext.filters.Filters attribute), 10
 location (telegram.InlineQuery attribute), 102, 173
 location (telegram.Message attribute), 81, 196
 location (telegram.Venue attribute), 97, 212

longitude (telegram.InlineQueryResultLocation attribute), 115, 185
 longitude (telegram.InlineQueryResultVenue attribute), 119, 188
 longitude (telegram.InputLocationMessageContent attribute), 122, 192
 longitude (telegram.InputVenueMessageContent attribute), 123, 193
 longitude (telegram.Location attribute), 79, 194

M

MARKDOWN (telegram.ParseMode attribute), 91
 mask_position (telegram.Sticker attribute), 100, 208
 MaskPosition (class in telegram), 101
 MAX_CAPTION_LENGTH (in module telegram.constants), 72
 max_connections (telegram.WebhookInfo attribute), 99, 213
 MAX_FILESIZE_DOWNLOAD (in module telegram.constants), 72
 MAX_FILESIZE_UPLOAD (in module telegram.constants), 72
 MAX_INLINE_QUERY_RESULTS (in module telegram.constants), 72
 MAX_MESSAGE_ENTITIES (in module telegram.constants), 72
 MAX_MESSAGE_LENGTH (in module telegram.constants), 71
 MAX_MESSAGES_PER_MINUTE_PER_GROUP (in module telegram.constants), 72
 MAX_MESSAGES_PER_SECOND (in module telegram.constants), 72
 MAX_MESSAGES_PER_SECOND_PER_CHAT (in module telegram.constants), 72
 media (telegram.InputMediaPhoto attribute), 77, 223
 media (telegram.InputMediaVideo attribute), 78, 224
 media_group_id (telegram.Message attribute), 80, 195
 MEMBER (telegram.ChatMember attribute), 71
 MENTION (telegram.MessageEntity attribute), 91
 mention_html() (in module telegram.utils.helpers), 3
 mention_html() (telegram.User method), 96
 mention_markdown() (in module telegram.utils.helpers), 3
 mention_markdown() (telegram.User method), 96
 MergedFilter (class in telegram.ext.filters), 11
 Message (class in telegram), 80
 message (telegram.CallbackQuery attribute), 64, 167
 message (telegram.Update attribute), 94, 209
 message_id (telegram.Message attribute), 80, 195
 message_text (telegram.InputTextMessageContent attribute), 122, 193
 message_updates (telegram.ext.MessageHandler attribute), 26
 MessageEntity (class in telegram), 90
 MessageHandler (class in telegram.ext), 26
 MessageQueue (class in telegram.ext), 15
 migrate (telegram.ext.filters.Filters attribute), 10

`migrate_from_chat_id` (`telegram.Message` attribute), 82, 197
`migrate_to_chat_id` (`telegram.Message` attribute), 82, 196
`mime_type` (`telegram.Animation` attribute), 131, 214
`mime_type` (`telegram.Audio` attribute), 34, 132
`mime_type` (`telegram.Document` attribute), 73, 170
`mime_type` (`telegram.InlineQueryResultDocument` attribute), 113, 183
`mime_type` (`telegram.InlineQueryResultVideo` attribute), 120, 189
`mime_type` (`telegram.Video` attribute), 97, 212
`mime_type` (`telegram.Voice` attribute), 99, 213
`MOUTH` (`telegram.MaskPosition` attribute), 102
`mpeg4_duration` (`telegram.InlineQueryResultMpeg4Gif` attribute), 117, 186
`mpeg4_file_id` (`telegram.InlineQueryResultCachedMpeg4Gif` attribute), 108, 178
`mpeg4_height` (`telegram.InlineQueryResultMpeg4Gif` attribute), 117, 186
`mpeg4_url` (`telegram.InlineQueryResultMpeg4Gif` attribute), 117, 186
`mpeg4_width` (`telegram.InlineQueryResultMpeg4Gif` attribute), 117, 186

N

`name` (`telegram.Bot` attribute), 47
`name` (`telegram.ext.DelayQueue` attribute), 16
`name` (`telegram.ext.filters.BaseFilter` attribute), 8
`name` (`telegram.ext.Job` attribute), 12
`name` (`telegram.OrderInfo` attribute), 126, 220
`name` (`telegram.StickerSet` attribute), 101, 222
`name` (`telegram.User` attribute), 96
`NetworkError`, 73
`new_chat_member` (`telegram.Message` attribute), 85
`new_chat_members` (`telegram.ext.filters.Filters` attribute), 10
`new_chat_members` (`telegram.Message` attribute), 81, 196
`new_chat_photo` (`telegram.ext.filters.Filters` attribute), 10
`new_chat_photo` (`telegram.Message` attribute), 81, 196
`new_chat_title` (`telegram.ext.filters.Filters` attribute), 10
`new_chat_title` (`telegram.Message` attribute), 81, 196

O

`offset` (`telegram.InlineQuery` attribute), 102, 173
`offset` (`telegram.MessageEntity` attribute), 90, 205
`one_time_keyboard` (`telegram.ReplyKeyboardMarkup` attribute), 93, 207
`order_info` (`telegram.PreCheckoutQuery` attribute), 129, 219
`order_info` (`telegram.SuccessfulPayment` attribute), 127, 217
`OrderInfo` (class in `telegram`), 126

P

`parse_caption_entities()` (`telegram.Message` method), 85
`parse_caption_entity()` (`telegram.Message` method), 86
`parse_entities()` (`telegram.Message` method), 86
`parse_entity()` (`telegram.Message` method), 86
`parse_mode` (`telegram.InputTextMessageContent` attribute), 122, 193
`parse_text_entities()` (`telegram.Game` method), 131
`parse_text_entity()` (`telegram.Game` method), 131
`ParseMode` (class in `telegram`), 91
`pass_args` (`telegram.ext.CommandHandler` attribute), 23
`pass_args` (`telegram.ext.StringCommandHandler` attribute), 31
`pass_chat_data` (`telegram.ext.CallbackQueryHandler` attribute), 19
`pass_chat_data` (`telegram.ext.ChosenInlineResultHandler` attribute), 20
`pass_chat_data` (`telegram.ext.CommandHandler` attribute), 23
`pass_chat_data` (`telegram.ext.Handler` attribute), 17
`pass_chat_data` (`telegram.ext.InlineQueryHandler` attribute), 25
`pass_chat_data` (`telegram.ext.MessageHandler` attribute), 26
`pass_chat_data` (`telegram.ext.PreCheckoutQueryHandler` attribute), 27
`pass_chat_data` (`telegram.ext.RegexHandler` attribute), 29
`pass_chat_data` (`telegram.ext.ShippingQueryHandler` attribute), 30
`pass_groupdict` (`telegram.ext.CallbackQueryHandler` attribute), 19
`pass_groupdict` (`telegram.ext.InlineQueryHandler` attribute), 24
`pass_groupdict` (`telegram.ext.RegexHandler` attribute), 29
`pass_groupdict` (`telegram.ext.StringRegexHandler` attribute), 32
`pass_groups` (`telegram.ext.CallbackQueryHandler` attribute), 19
`pass_groups` (`telegram.ext.InlineQueryHandler` attribute), 24
`pass_groups` (`telegram.ext.RegexHandler` attribute), 28
`pass_groups` (`telegram.ext.StringRegexHandler` attribute), 32
`pass_job_queue` (`telegram.ext.CallbackQueryHandler` attribute), 18
`pass_job_queue` (`telegram.ext.ChosenInlineResultHandler` attribute), 20
`pass_job_queue` (`telegram.ext.CommandHandler` attribute), 23
`pass_job_queue` (`telegram.ext.Handler` attribute), 17
`pass_job_queue` (`telegram.ext.InlineQueryHandler` attribute), 24
`pass_job_queue` (`telegram.ext.MessageHandler` at-

- tribute), 26
- pass_job_queue (telegram.ext.PreCheckoutQueryHandler attribute), 27
- pass_job_queue (telegram.ext.RegexHandler attribute), 29
- pass_job_queue (telegram.ext.ShippingQueryHandler attribute), 30
- pass_job_queue (telegram.ext.StringCommandHandler attribute), 31
- pass_job_queue (telegram.ext.StringRegexHandler attribute), 32
- pass_job_queue (telegram.ext.TypeHandler attribute), 33
- pass_update_queue (telegram.ext.CallbackQueryHandler attribute), 18
- pass_update_queue (telegram.ext.ChosenInlineResultHandler attribute), 20
- pass_update_queue (telegram.ext.CommandHandler attribute), 23
- pass_update_queue (telegram.ext.Handler attribute), 17
- pass_update_queue (telegram.ext.InlineQueryHandler attribute), 24
- pass_update_queue (telegram.ext.MessageHandler attribute), 26
- pass_update_queue (telegram.ext.PreCheckoutQueryHandler attribute), 27
- pass_update_queue (telegram.ext.RegexHandler attribute), 29
- pass_update_queue (telegram.ext.ShippingQueryHandler attribute), 30
- pass_update_queue (telegram.ext.StringCommandHandler attribute), 31
- pass_update_queue (telegram.ext.StringRegexHandler attribute), 32
- pass_update_queue (telegram.ext.TypeHandler attribute), 33
- pass_user_data (telegram.ext.CallbackQueryHandler attribute), 19
- pass_user_data (telegram.ext.ChosenInlineResultHandler attribute), 20
- pass_user_data (telegram.ext.CommandHandler attribute), 23
- pass_user_data (telegram.ext.Handler attribute), 17
- pass_user_data (telegram.ext.InlineQueryHandler attribute), 24
- pass_user_data (telegram.ext.MessageHandler attribute), 26
- pass_user_data (telegram.ext.PreCheckoutQueryHandler attribute), 27
- pass_user_data (telegram.ext.RegexHandler attribute), 29
- pass_user_data (telegram.ext.ShippingQueryHandler attribute), 30
- pattern (telegram.ext.CallbackQueryHandler attribute), 19
- pattern (telegram.ext.InlineQueryHandler attribute), 24
- pattern (telegram.ext.RegexHandler attribute), 28
- pattern (telegram.ext.StringRegexHandler attribute), 32
- pay (telegram.InlineKeyboardButton attribute), 75, 172
- pending_update_count (telegram.WebhookInfo attribute), 99, 213
- per_chat (telegram.ext.ConversationHandler attribute), 21
- per_message (telegram.ext.ConversationHandler attribute), 22
- per_user (telegram.ext.ConversationHandler attribute), 22
- performer (telegram.Audio attribute), 34, 132
- performer (telegram.InlineQueryResultAudio attribute), 105, 175
- phone_number (telegram.Contact attribute), 72, 169
- phone_number (telegram.InlineQueryResultContact attribute), 112, 182
- phone_number (telegram.InputContactMessageContent attribute), 123, 191
- phone_number (telegram.OrderInfo attribute), 126, 220
- photo (telegram.Chat attribute), 66, 162
- photo (telegram.ext.filters.Filters attribute), 10
- photo (telegram.Game attribute), 130, 214
- photo (telegram.Message attribute), 81, 196
- photo_file_id (telegram.InlineQueryResultCachedPhoto attribute), 109, 179
- photo_height (telegram.InlineQueryResultPhoto attribute), 118, 187
- photo_url (telegram.InlineQueryResultPhoto attribute), 118, 187
- photo_width (telegram.InlineQueryResultPhoto attribute), 118, 187
- photos (telegram.UserProfilePhotos attribute), 96, 211
- PhotoSize (class in telegram), 92
- pin_chat_message() (telegram.Bot method), 47
- pinned_message (telegram.Chat attribute), 66, 162
- pinned_message (telegram.ext.filters.Filters attribute), 11
- pinned_message (telegram.Message attribute), 82, 197
- point (telegram.MaskPosition attribute), 101, 223
- position (telegram.GameHighScore attribute), 132, 216
- post_code (telegram.ShippingAddress attribute), 126, 218
- PRE (telegram.MessageEntity attribute), 91
- pre_checkout_query (telegram.Update attribute), 94, 209
- PreCheckoutQuery (class in telegram), 129
- PreCheckoutQueryHandler (class in telegram.ext), 27
- prices (telegram.ShippingOption attribute), 127, 218
- PRIVATE (telegram.Chat attribute), 67
- private (telegram.ext.filters.Filters attribute), 10
- process_update() (telegram.ext.Dispatcher method), 7
- promote_chat_member() (telegram.Bot method), 48
- provider_payment_charge_id (telegram.PaymentInfo attribute), 30

gram.SuccessfulPayment attribute), 127, 217
put() (telegram.ext.JobQueue method), 13

Q

query (telegram.ChosenInlineResult attribute), 124, 167
query (telegram.InlineQuery attribute), 102, 173

R

RECORD_AUDIO (telegram.ChatAction attribute), 69
RECORD_VIDEO (telegram.ChatAction attribute), 69
RECORD_VIDEO_NOTE (telegram.ChatAction attribute), 69
RegexHandler (class in telegram.ext), 28
remove_error_handler() (telegram.ext.Dispatcher method), 7
remove_handler() (telegram.ext.Dispatcher method), 7
remove_keyboard (telegram.ReplyKeyboardRemove attribute), 92, 206
removed (telegram.ext.Job attribute), 12
repeat (telegram.ext.Job attribute), 12
reply (telegram.ext.filters.Filters attribute), 10
reply_audio() (telegram.Message method), 87
reply_contact() (telegram.Message method), 87
reply_document() (telegram.Message method), 87
reply_html() (telegram.Message method), 87
reply_location() (telegram.Message method), 87
reply_markdown() (telegram.Message method), 88
reply_markup (telegram.InlineQueryResultArticle attribute), 104, 174
reply_markup (telegram.InlineQueryResultAudio attribute), 105, 176
reply_markup (telegram.InlineQueryResultCachedAudio attribute), 105, 176
reply_markup (telegram.InlineQueryResultCachedDocument attribute), 106, 177
reply_markup (telegram.InlineQueryResultCachedGif attribute), 107, 178
reply_markup (telegram.InlineQueryResultCachedMpeg4Gif attribute), 108, 178
reply_markup (telegram.InlineQueryResultCachedPhoto attribute), 109, 179
reply_markup (telegram.InlineQueryResultCachedSticker attribute), 109, 180
reply_markup (telegram.InlineQueryResultCachedVideo attribute), 110, 181
reply_markup (telegram.InlineQueryResultCachedVoice attribute), 111, 181
reply_markup (telegram.InlineQueryResultContact attribute), 112, 182
reply_markup (telegram.InlineQueryResultDocument attribute), 113, 183
reply_markup (telegram.InlineQueryResultGame attribute), 114, 191
reply_markup (telegram.InlineQueryResultGif attribute), 115, 184
reply_markup (telegram.InlineQueryResultLocation attribute), 116, 185
reply_markup (telegram.InlineQueryResultMpeg4Gif attribute), 117, 186
reply_markup (telegram.InlineQueryResultPhoto attribute), 118, 187
reply_markup (telegram.InlineQueryResultVenue attribute), 119, 188
reply_markup (telegram.InlineQueryResultVideo attribute), 120, 189
reply_markup (telegram.InlineQueryResultVoice attribute), 121, 190
reply_media_group() (telegram.Message method), 88
reply_photo() (telegram.Message method), 88
reply_sticker() (telegram.Message method), 88
reply_text() (telegram.Message method), 89
reply_to_message (telegram.Message attribute), 80, 195
reply_venue() (telegram.Message method), 89
reply_video() (telegram.Message method), 89
reply_video_note() (telegram.Message method), 89
reply_voice() (telegram.Message method), 89
ReplyKeyboardMarkup (class in telegram), 93
ReplyKeyboardRemove (class in telegram), 92
ReplyMarkup (class in telegram), 93
request_contact (telegram.KeyboardButton attribute), 79, 194
request_location (telegram.KeyboardButton attribute), 79, 194
resize_keyboard (telegram.ReplyKeyboardMarkup attribute), 93, 207
restrict_chat_member() (telegram.Bot method), 48
RESTRICTED (telegram.ChatMember attribute), 71
result_id (telegram.ChosenInlineResult attribute), 124, 166
RetryAfter, 73
run() (telegram.ext.DelayQueue method), 17
run() (telegram.ext.Job method), 12
run_async() (telegram.ext.Dispatcher method), 7
run_async_timeout (telegram.ext.ConversationHandler attribute), 21
run_daily() (telegram.ext.JobQueue method), 13
run_once() (telegram.ext.JobQueue method), 14
run_repeating() (telegram.ext.JobQueue method), 14
running (telegram.ext.Dispatcher attribute), 8
running (telegram.ext.Updater attribute), 4

S

scale (telegram.MaskPosition attribute), 101, 223
schedule_removal() (telegram.ext.Job method), 12
score (telegram.GameHighScore attribute), 132, 216
selective (telegram.ForceReply attribute), 75, 171
selective (telegram.ReplyKeyboardMarkup attribute), 93, 207
selective (telegram.ReplyKeyboardRemove attribute), 92, 207
send_action() (telegram.Chat method), 68
send_audio() (telegram.Bot method), 49

- send_chat_action() (telegram.Bot method), 50
 - send_contact() (telegram.Bot method), 50
 - send_document() (telegram.Bot method), 51
 - send_game() (telegram.Bot method), 51
 - send_invoice() (telegram.Bot method), 52
 - send_location() (telegram.Bot method), 53
 - send_media_group() (telegram.Bot method), 54
 - send_message() (telegram.Bot method), 54
 - send_photo() (telegram.Bot method), 55
 - send_sticker() (telegram.Bot method), 55
 - send_venue() (telegram.Bot method), 56
 - send_video() (telegram.Bot method), 57
 - send_video_note() (telegram.Bot method), 57
 - send_voice() (telegram.Bot method), 58
 - set_chat_description() (telegram.Bot method), 59
 - set_chat_photo() (telegram.Bot method), 59
 - set_chat_sticker_set() (telegram.Bot method), 60
 - set_chat_title() (telegram.Bot method), 60
 - set_game_score() (telegram.Bot method), 60
 - set_name (telegram.Sticker attribute), 100, 208
 - set_sticker_position_in_set() (telegram.Bot method), 61
 - set_webhook() (telegram.Bot method), 61
 - shipping_address (telegram.OrderInfo attribute), 126, 220
 - shipping_address (telegram.ShippingQuery attribute), 128, 221
 - shipping_option_id (telegram.PreCheckoutQuery attribute), 129, 219
 - shipping_option_id (telegram.SuccessfulPayment attribute), 127, 217
 - shipping_query (telegram.Update attribute), 94, 209
 - ShippingAddress (class in telegram), 125
 - ShippingOption (class in telegram), 126
 - ShippingQuery (class in telegram), 128
 - ShippingQueryHandler (class in telegram.ext), 30
 - small_file_id (telegram.ChatPhoto attribute), 71, 222
 - start() (telegram.ext.Dispatcher method), 8
 - start() (telegram.ext.JobQueue method), 15
 - start() (telegram.ext.MessageQueue method), 16
 - start_parameter (telegram.Invoice attribute), 125, 220
 - start_polling() (telegram.ext.Updater method), 5
 - start_webhook() (telegram.ext.Updater method), 5
 - state (telegram.ShippingAddress attribute), 125, 218
 - states (telegram.ext.ConversationHandler attribute), 21
 - status (telegram.ChatMember attribute), 69, 164
 - status_update (telegram.ext.filters.Filters attribute), 10
 - Sticker (class in telegram), 100
 - sticker (telegram.ext.filters.Filters attribute), 11
 - sticker (telegram.Message attribute), 81, 196
 - sticker_file_id (telegram.InlineQueryResultCachedSticker attribute), 109, 180
 - sticker_set_name (telegram.Chat attribute), 66, 162
 - stickers (telegram.StickerSet attribute), 101, 222
 - StickerSet (class in telegram), 101
 - stop() (telegram.ext.DelayQueue method), 17
 - stop() (telegram.ext.Dispatcher method), 8
 - stop() (telegram.ext.JobQueue method), 15
 - stop() (telegram.ext.MessageQueue method), 16
 - stop() (telegram.ext.Updater method), 6
 - stop_message_live_location() (telegram.Bot method), 62
 - street_line1 (telegram.ShippingAddress attribute), 126, 218
 - street_line2 (telegram.ShippingAddress attribute), 126, 218
 - strict (telegram.ext.TypeHandler attribute), 33
 - StringCommandHandler (class in telegram.ext), 31
 - StringRegexHandler (class in telegram.ext), 32
 - successful_payment (telegram.ext.filters.Filters attribute), 11
 - successful_payment (telegram.Message attribute), 82, 197
 - SuccessfulPayment (class in telegram), 127
 - SUPERGROUP (telegram.Chat attribute), 67
 - supergroup_chat_created (telegram.Message attribute), 81, 196
 - SUPPORTED_WEBHOOK_PORTS (in module telegram.constants), 72
 - switch_inline_query (telegram.InlineKeyboardButton attribute), 75, 172
 - switch_inline_query_current_chat (telegram.InlineKeyboardButton attribute), 75, 172
- ## T
- telegram.constants (module), 71
 - telegram.error (module), 73
 - telegram.ext.filters (module), 8
 - telegram.utils.helpers (module), 3
 - telegram_payment_charge_id (telegram.SuccessfulPayment attribute), 127, 217
 - TelegramError, 73
 - TelegramObject (class in telegram), 94
 - text (telegram.ext.filters.Filters attribute), 11
 - text (telegram.Game attribute), 130, 214
 - text (telegram.InlineKeyboardButton attribute), 75, 171
 - text (telegram.KeyboardButton attribute), 79, 194
 - text (telegram.Message attribute), 80, 195
 - text_entities (telegram.Game attribute), 130, 215
 - text_html (telegram.Message attribute), 90
 - text_html_urled (telegram.Message attribute), 90
 - TEXT_LINK (telegram.MessageEntity attribute), 91
 - text_markdown (telegram.Message attribute), 90
 - text_markdown_urled (telegram.Message attribute), 90
 - TEXT_MENTION (telegram.MessageEntity attribute), 91
 - thumb (telegram.Animation attribute), 131, 214
 - thumb (telegram.Document attribute), 73, 170
 - thumb (telegram.Sticker attribute), 100, 208
 - thumb (telegram.Video attribute), 97, 212
 - thumb (telegram.VideoNote attribute), 98, 216
 - thumb_height (telegram.InlineQueryResultArticle attribute), 104, 175

thumb_height (telegram.InlineQueryResultContact attribute), 112, 182

thumb_height (telegram.InlineQueryResultDocument attribute), 113, 183

thumb_height (telegram.InlineQueryResultLocation attribute), 116, 185

thumb_height (telegram.InlineQueryResultVenue attribute), 119, 188

thumb_url (telegram.InlineQueryResultArticle attribute), 104, 175

thumb_url (telegram.InlineQueryResultContact attribute), 112, 182

thumb_url (telegram.InlineQueryResultDocument attribute), 113, 183

thumb_url (telegram.InlineQueryResultGif attribute), 115, 184

thumb_url (telegram.InlineQueryResultLocation attribute), 116, 185

thumb_url (telegram.InlineQueryResultMpeg4Gif attribute), 117, 186

thumb_url (telegram.InlineQueryResultPhoto attribute), 118, 187

thumb_url (telegram.InlineQueryResultVenue attribute), 119, 188

thumb_url (telegram.InlineQueryResultVideo attribute), 120, 189

thumb_width (telegram.InlineQueryResultArticle attribute), 104, 175

thumb_width (telegram.InlineQueryResultContact attribute), 112, 182

thumb_width (telegram.InlineQueryResultDocument attribute), 113, 183

thumb_width (telegram.InlineQueryResultLocation attribute), 116, 185

thumb_width (telegram.InlineQueryResultVenue attribute), 119, 188

tick() (telegram.ext.JobQueue method), 15

time_limit (telegram.ext.DelayQueue attribute), 16

timed_out_behavior (telegram.ext.ConversationHandler attribute), 21

TimedOut, 73

title (telegram.Audio attribute), 34, 132

title (telegram.Chat attribute), 66, 162

title (telegram.Game attribute), 130, 214

title (telegram.InlineQueryResultArticle attribute), 103, 174

title (telegram.InlineQueryResultAudio attribute), 105, 175

title (telegram.InlineQueryResultCachedDocument attribute), 106, 177

title (telegram.InlineQueryResultCachedGif attribute), 107, 178

title (telegram.InlineQueryResultCachedMpeg4Gif attribute), 108, 178

title (telegram.InlineQueryResultCachedPhoto attribute), 109, 179

title (telegram.InlineQueryResultCachedVideo attribute), 110, 180

title (telegram.InlineQueryResultCachedVoice attribute), 111, 181

title (telegram.InlineQueryResultDocument attribute), 113, 183

title (telegram.InlineQueryResultGif attribute), 115, 184

title (telegram.InlineQueryResultLocation attribute), 116, 185

title (telegram.InlineQueryResultMpeg4Gif attribute), 117, 186

title (telegram.InlineQueryResultPhoto attribute), 118, 187

title (telegram.InlineQueryResultVenue attribute), 119, 188

title (telegram.InlineQueryResultVideo attribute), 120, 189

title (telegram.InlineQueryResultVoice attribute), 121, 190

title (telegram.InputVenueMessageContent attribute), 123, 193

title (telegram.Invoice attribute), 125, 220

title (telegram.ShippingOption attribute), 127, 218

title (telegram.StickerSet attribute), 101, 222

title (telegram.Venue attribute), 97, 212

to_form() (telegram.InputFile method), 77

to_json() (telegram.TelegramObject method), 94

to_timestamp() (in module telegram.utils.helpers), 3

total_amount (telegram.Invoice attribute), 125, 220

total_amount (telegram.PreCheckoutQuery attribute), 129, 219

total_amount (telegram.SuccessfulPayment attribute), 127, 217

total_count (telegram.UserProfilePhotos attribute), 96, 211

type (telegram.Chat attribute), 66, 162

type (telegram.ext.TypeHandler attribute), 33

type (telegram.InlineQueryResult attribute), 103, 174

type (telegram.InlineQueryResultArticle attribute), 103, 174

type (telegram.InlineQueryResultAudio attribute), 104, 175

type (telegram.InlineQueryResultCachedAudio attribute), 105, 176

type (telegram.InlineQueryResultCachedDocument attribute), 106, 177

type (telegram.InlineQueryResultCachedGif attribute), 107, 178

type (telegram.InlineQueryResultCachedMpeg4Gif attribute), 108, 178

type (telegram.InlineQueryResultCachedPhoto attribute), 108, 179

type (telegram.InlineQueryResultCachedSticker attribute), 109, 180

type (telegram.InlineQueryResultCachedVideo attribute), 110, 180

type (telegram.InlineQueryResultCachedVoice attribute), 111, 181

type (telegram.InlineQueryResultContact attribute), 112, 182

type (telegram.InlineQueryResultDocument attribute), 113, 183

type (telegram.InlineQueryResultGame attribute), 114, 191

type (telegram.InlineQueryResultGif attribute), 114, 184

type (telegram.InlineQueryResultLocation attribute), 115, 185

type (telegram.InlineQueryResultMpeg4Gif attribute), 116, 186

type (telegram.InlineQueryResultPhoto attribute), 118, 187

type (telegram.InlineQueryResultVenue attribute), 119, 188

type (telegram.InlineQueryResultVideo attribute), 120, 189

type (telegram.InlineQueryResultVoice attribute), 121, 190

type (telegram.InputMediaPhoto attribute), 77, 223

type (telegram.InputMediaVideo attribute), 78, 224

type (telegram.MessageEntity attribute), 90, 205

TypeHandler (class in telegram.ext), 33

TYPING (telegram.ChatAction attribute), 69

U

Unauthorized, 73

unban_chat_member() (telegram.Bot method), 63

unban_member() (telegram.Chat method), 68

unpin_chat_message() (telegram.Bot method), 63

until_date (telegram.ChatMember attribute), 69, 164

Update (class in telegram), 94

update_id (telegram.Update attribute), 94, 209

update_queue (telegram.ext.Dispatcher attribute), 6

update_queue (telegram.ext.Updater attribute), 4

Updater (class in telegram.ext), 4

UPLOAD_AUDIO (telegram.ChatAction attribute), 69

UPLOAD_DOCUMENT (telegram.ChatAction attribute), 69

UPLOAD_PHOTO (telegram.ChatAction attribute), 69

upload_sticker_file() (telegram.Bot method), 63

UPLOAD_VIDEO (telegram.ChatAction attribute), 69

UPLOAD_VIDEO_NOTE (telegram.ChatAction attribute), 69

url (telegram.InlineKeyboardButton attribute), 75, 171

url (telegram.InlineQueryResultArticle attribute), 104, 174

URL (telegram.MessageEntity attribute), 91

url (telegram.MessageEntity attribute), 90, 205

url (telegram.WebhookInfo attribute), 99, 213

User (class in telegram), 95

user (telegram.ChatMember attribute), 69, 164

user (telegram.GameHighScore attribute), 132, 216

user (telegram.MessageEntity attribute), 90, 205

user_data (telegram.ext.Dispatcher attribute), 8

user_id (telegram.Contact attribute), 72, 169

user_sig_handler (telegram.ext.Updater attribute), 4

username (telegram.Bot attribute), 64

username (telegram.Chat attribute), 66, 162

username (telegram.User attribute), 95, 210

UserProfilePhotos (class in telegram), 96

V

Venue (class in telegram), 97

venue (telegram.ext.filters.Filters attribute), 11

venue (telegram.Message attribute), 81, 196

Video (class in telegram), 97

video (telegram.ext.filters.Filters attribute), 11

video (telegram.Message attribute), 81, 196

video_duration (telegram.InlineQueryResultVideo attribute), 120, 189

video_file_id (telegram.InlineQueryResultCachedVideo attribute), 110, 180

video_height (telegram.InlineQueryResultVideo attribute), 120, 189

video_note (telegram.Message attribute), 81, 196

video_url (telegram.InlineQueryResultVideo attribute), 120, 189

video_width (telegram.InlineQueryResultVideo attribute), 120, 189

VideoNote (class in telegram), 98

Voice (class in telegram), 98

voice (telegram.ext.filters.Filters attribute), 11

voice (telegram.Message attribute), 81, 196

voice_duration (telegram.InlineQueryResultVoice attribute), 121, 190

voice_file_id (telegram.InlineQueryResultCachedVoice attribute), 111, 181

voice_url (telegram.InlineQueryResultVoice attribute), 121, 190

W

WebhookInfo (class in telegram), 99

width (telegram.InputMediaVideo attribute), 78, 224

width (telegram.PhotoSize attribute), 92, 206

width (telegram.Sticker attribute), 100, 208

width (telegram.Video attribute), 97, 212

workers (telegram.ext.Dispatcher attribute), 6

X

x_shift (telegram.MaskPosition attribute), 101, 223

Y

y_shift (telegram.MaskPosition attribute), 101, 223