



# **Python Telegram Bot Documentation**

*Release 6.0.0*

**Leandro Toledo**

**May 19, 2017**



<b>1</b>	<b>telegram package</b>	<b>1</b>
1.1	Submodules	1
1.1.1	telegram.contrib package	1
1.1.2	telegram.ext package	1
1.1.3	telegram.animation module	38
1.1.4	telegram.audio module	39
1.1.5	telegram.base module	39
1.1.6	telegram.bot module	40
1.1.7	telegram.callbackgame module	71
1.1.8	telegram.callbackquery module	71
1.1.9	telegram.chat module	72
1.1.10	telegram.chataction module	73
1.1.11	telegram.chatmember module	74
1.1.12	telegram.choseninlineresult module	74
1.1.13	telegram.constants module	75
1.1.14	telegram.contact module	76
1.1.15	telegram.document module	77
1.1.16	telegram.error module	77
1.1.17	telegram.file module	78
1.1.18	telegram.forcereply module	79
1.1.19	telegram.game module	79
1.1.20	telegram.gamehighscore module	80
1.1.21	telegram.inlinekeyboardbutton module	81
1.1.22	telegram.inlinekeyboardmarkup module	82
1.1.23	telegram.inlinequery module	82
1.1.24	telegram.inlinequeryresult module	83
1.1.25	telegram.inlinequeryresultarticle module	84
1.1.26	telegram.inlinequeryresultaudio module	85
1.1.27	telegram.inlinequeryresultcachedaudio module	86
1.1.28	telegram.inlinequeryresultcacheddocument module	87
1.1.29	telegram.inlinequeryresultcachedgif module	88
1.1.30	telegram.inlinequeryresultcachedmpeg4gif module	89
1.1.31	telegram.inlinequeryresultcachedphoto module	90
1.1.32	telegram.inlinequeryresultcachedsticker module	91
1.1.33	telegram.inlinequeryresultcachedvideo module	91
1.1.34	telegram.inlinequeryresultcachedvoice module	92
1.1.35	telegram.inlinequeryresultcontact module	93
1.1.36	telegram.inlinequeryresultdocument module	94
1.1.37	telegram.inlinequeryresultgame module	95
1.1.38	telegram.inlinequeryresultgif module	96

1.1.39	telegram.inlinequeryresultlocation module . . . . .	97
1.1.40	telegram.inlinequeryresultmpeg4gif module . . . . .	98
1.1.41	telegram.inlinequeryresultphoto module . . . . .	99
1.1.42	telegram.inlinequeryresultvenue module . . . . .	100
1.1.43	telegram.inlinequeryresultvideo module . . . . .	100
1.1.44	telegram.inlinequeryresultvoice module . . . . .	100
1.1.45	telegram.inputcontactmessagecontent module . . . . .	101
1.1.46	telegram.inputfile module . . . . .	101
1.1.47	telegram.inputlocationmessagecontent module . . . . .	101
1.1.48	telegram.inputmessagecontent module . . . . .	102
1.1.49	telegram.inputtextmessagecontent module . . . . .	102
1.1.50	telegram.inputvenuemessagecontent module . . . . .	102
1.1.51	telegram.keyboardbutton module . . . . .	102
1.1.52	telegram.location module . . . . .	103
1.1.53	telegram.message module . . . . .	103
1.1.54	telegram.messageentity module . . . . .	109
1.1.55	telegram.parsemode module . . . . .	110
1.1.56	telegram.photosize module . . . . .	110
1.1.57	telegram.replykeyboardremove module . . . . .	111
1.1.58	telegram.replykeyboardmarkup module . . . . .	112
1.1.59	telegram.replymarkup module . . . . .	113
1.1.60	telegram.sticker module . . . . .	113
1.1.61	telegram.update module . . . . .	114
1.1.62	telegram.user module . . . . .	115
1.1.63	telegram.userprofilephotos module . . . . .	116
1.1.64	telegram.venue module . . . . .	116
1.1.65	telegram.video module . . . . .	117
1.1.66	telegram.voice module . . . . .	117
1.1.67	telegram.webhookinfo module . . . . .	118
1.2	Module contents . . . . .	119

**2 Indices and tables 193**

**Python Module Index 195**

### Submodules

#### telegram.contrib package

##### Submodules

#### telegram.contrib.botan module

**class** telegram.contrib.botan.**Botan** (*token*)

Bases: object

This class helps to send incoming events to your botan analytics account. See more: <https://github.com/botanio/sdk#botan-sdk>

**token** = ‘

**track** (*message*, *event\_name*=‘event’)

**url\_template** = ‘https://api.botan.io/track?token={token}&uid={uid}&name={name}&src=python-telegram-bot’

##### Module contents

#### telegram.ext package

##### Submodules

#### telegram.ext.updater module

This module contains the class Updater, which tries to make creating Telegram bots intuitive.

**class** telegram.ext.updater.**Updater** (*token=None*, *base\_url=None*, *workers=4*, *bot=None*,  
*user\_sig\_handler=None*, *request\_kwargs=None*)

Bases: object

This class, which employs the Dispatcher class, 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.

Attributes:

#### Parameters

- **token** (*Optional[str]*) – The bot’s token given by the `@BotFather`
- **base\_url** (*Optional[str]*) –
- **workers** (*Optional[int]*) – Amount of threads in the thread pool for functions decorated with `@run_async`
- **bot** (*Optional[Bot]*) – A pre-initialized bot instance. If a pre-initizlied 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** (*Optional[function]*) – Takes `signum`, `frame` as positional arguments. This will be called when a signal is received, defaults are (`SIGINT`, `SIGTERM`, `SIGABRT`) setable with `Updater.idle(stop_signals=(signals))`
- **request\_kwargs** (*Optional[dict]*) – Keyword args to control the creation of a request object (ignored if `bot` argument is used).

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

**idle** (*stop\_signals=(2, 15, 6)*)

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

**Parameters**`stop_signals` – 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`)

**signal\_handler** (*signum, frame*)

**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** (*Optional[float]*) – Time to wait between polling updates from Telegram in
- **Default is 0.0** (*seconds.*) –
- **timeout** (*Optional[float]*) – Passed to `Bot.getUpdates`
- **network\_delay** – Deprecated. Will be honoured as `read_latency` for a while but will be removed in the future.
- **clean** (*Optional[bool]*) – Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is `False`.
- **bootstrap\_retries** (*Optional[int]*) – 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** (*Optional[list[str]]*) – Passed to `Bot.getUpdates`
- **read\_latency** (*Optional[float|int]*) – Grace time in seconds for receiving the reply from server. Will be added to the `timeout` value and used as the read timeout from server (Default: 2).

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

**Return type**Queue

**start\_webhook** (*listen='127.0.0.1', port=80, url\_path='', cert=None, key=None, clean=False, bootstrap\_retries=0, webhook\_url=None, allowed\_updates=None*)

Starts a small http server to listen for updates via webhook. If cert and key are not provided, the webhook will be started directly on [http://listen:port/url\\_path](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](https://listen:port/url_path)

**Parameters**

- **listen** (*Optional[str]*) – IP-Address to listen on
- **port** (*Optional[int]*) – Port the bot should be listening on
- **url\_path** (*Optional[str]*) – Path inside url
- **cert** (*Optional[str]*) – Path to the SSL certificate file
- **key** (*Optional[str]*) – Path to the SSL key file
- **clean** (*Optional[bool]*) – Whether to clean any pending updates on Telegram servers before actually starting the webhook. Default is False.
- **bootstrap\_retries** (*Optional[int]*) – 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** (*Optional[str]*) – Explicitly specify the webhook url. Useful behind NAT, reverse proxy, etc. Default is derived from *listen, port & url\_path*.

- **allowed\_updates** (*Optional[list[str]]*) – Passed to Bot.setWebhook

**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

## telegram.ext.dispatcher module

This module contains the Dispatcher class.

**class** telegram.ext.dispatcher.**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.

**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** (*Optional[telegram.ext.JobQueue]*) – The *JobQueue* instance to pass onto handler callbacks
- **workers** (*Optional[int]*) – Number of maximum concurrent worker threads for the `@run_async` decorator

**add\_error\_handler** (*callback*)

Registers an error handler in the Dispatcher.

**Parameters****handler** (*function*) – 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.

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 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** (*Optional[int]*) – The group identifier. Default is 0

**chat\_data = None**

*type* – dict[int, dict]

**dispatch\_error** (*update, error*)

Dispatches an error.

**Parameters**

•**update** (*object*) – The update that caused the error

•**error** (*telegram.TelegramError*) – The Telegram error that was raised.

**classmethod get\_instance** ()

Get the singleton instance of this class.

**Returns**Dispatcher

**groups = None**

*type* – list[int]

**handlers = None**

*type* – dict[int, list[Handler]]

**has\_running\_threads**

**logger = <logging.Logger object>**

**process\_update** (*update*)

Processes a single update.

**Parameters****update** (*object*) –

**remove\_error\_handler** (*callback*)

De-registers an error handler.

**Parameters****handler** (*function*) –

**remove\_handler** (*handler, group=0*)

Remove a handler from the specified group

**Parameters**

•**handler** (*telegram.ext.Handler*) – A Handler instance



- **group** (*optional[object]*) – The group identifier. Default is 0

**run\_async** (*func, \*args, \*\*kwargs*)

Queue a function (with given args/kwags) to be run asynchronously.

#### Parameters

- **func** (*function*) – The function to run in the thread.

- **args** (*Optional[tuple]*) – Arguments to *func*.

- **kwargs** (*Optional[dict]*) – Keyword arguments to *func*.

#### Returns

**start** ()

Thread target of thread ‘dispatcher’. Runs in background and processes the update queue.

**stop** ()

Stops the thread

**user\_data = None**

*type* – dict[int, dict]

`telegram.ext.dispatcher.run_async` (*func*)

Function decorator that will run the function in a new thread.

Using this decorator is only possible when only a single Dispatcher exist in the system.

#### Parameters

- **func** (*function*) – The function to run in the thread.

- **async\_queue** (*Queue*) – The queue of the functions to be executed asynchronously.

#### Returns

**Return type**function

## telegram.ext.jobqueue module

This module contains the classes JobQueue and Job.

**class** `telegram.ext.jobqueue.Days`

Bases: object

**EVERY\_DAY** = (0, 1, 2, 3, 4, 5, 6)

**FRI** = 4

**MON** = 0

**SAT** = 5

**SUN** = 6

**THU** = 3

**TUE** = 1

**WED** = 2

**class** `telegram.ext.jobqueue.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**

*function* – The function that the job executes when it’s due

**interval**

*int, float, datetime.timedelta* – The interval in which the job runs

**days**

*tuple[int]* – A tuple of `int` values that determine on which days of the week the job runs

**repeat**

*bool* – If the job runs periodically or only once

**name**

*str* – The name of this job

**job\_queue**

*JobQueue* – The `JobQueue` this job belongs to

**enabled**

*bool* – Boolean property that decides if this job is currently active

**Parameters**

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

**days**

**enabled**

**interval**

**interval\_seconds**

**job\_queue**

*rtype* – `JobQueue`

**removed**

**repeat**

**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.

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

Bases: `object`

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

**queue**  
*PriorityQueue*

**bot**  
*telegram.Bot*

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

**Deprecated: 5.2** *prevent\_autostart* (Optional[bool]): 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.

#### Parameters

- **job** (*telegram.ext.Job*) – The *Job* instance representing the new job
- **next\_t** (Optional[*int*, *float*, *datetime.timedelta*, *datetime.datetime*, *datetime.time*]) – 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** (*function*) – The callback function that should be executed by the new job. It should take two parameters *bot* and *job*, where *job* is the *Job* instance. It can be used to access its *context* or terminate the job.
- **time** (*datetime.time*) – Time of day at which the job should run.
- **days** (Optional[*tuple[int]*]) – Defines on which days of the week the job should run.
- **to** *Days.EVERY\_DAY* (Defaults) –
- **context** (Optional[*object*]) – Additional data needed for the callback function. Can be accessed through *job.context* in the callback. Defaults to *None*
- **name** (Optional[*str*]) – 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** *Job*

**run\_once** (*callback*, *when*, *context=None*, *name=None*)  
Creates a new *Job* that runs once and adds it to the queue.

#### Parameters

- **callback** (*function*) – The callback function that should be executed by the new job. It should take two parameters *bot* and *job*, where *job* is the *Job* instance. It can be used to access its *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** (*Optional[object]*) – Additional data needed for the callback function. Can be accessed through `job.context` in the callback. Defaults to `None`

•**name** (*Optional[str]*) – 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**`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** (*function*) – The callback function that should be executed by the new job. It should take two parameters `bot` and `job`, where `job` is the `Job` instance. It can be used to access its `context`, terminate the job or change its interval.

•**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*) –

–*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** (*Optional[object]*) – Additional data needed for the callback function. Can be accessed through `job.context` in the callback. Defaults to `None`

•**name** (*Optional[str]*) – 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**`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.

## telegram.ext.handler module

This module contains the base class for handlers as used by the Dispatcher

```
class telegram.ext.handler.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. You can create your own handlers by inheriting from this class.

### Parameters

- **callback** (*function*) – 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** (*optional [bool]*) – 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 `Updater` and `Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.
- **pass\_job\_queue** (*optional [bool]*) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `JobQueue` instance created by the `Updater` which can be used to schedule new jobs. Default is `False`.
- **pass\_user\_data** (*optional [bool]*) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the user that sent the update. For each update of the same user, it will be the same `dict`. Default is `False`.
- **pass\_chat\_data** (*optional [bool]*) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the chat that the update was sent in. For each update in the same chat, it will be the same `dict`. 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` (*object*) – 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`) –

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

## telegram.ext.callbackqueryhandler module

This module contains the `CallbackQueryHandler` class

```
class telegram.ext.callbackqueryhandler.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.

### Parameters

- **callback** (*function*) – 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** (*optional[bool]*) – 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 `Updater` and `Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.
- **pass\_job\_queue** (*optional[bool]*) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `JobQueue` instance created by the `Updater` which can be used to schedule new jobs. Default is `False`.
- **pattern** (*optional[str or Pattern]*) – Optional regex pattern. If not `None` `re.match` is used to determine if an update should be handled by this handler.
- **pass\_groups** (*optional[bool]*) – 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** (*optional[bool]*) – 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** (*optional[bool]*) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the user that sent the update. For each update of the same user, it will be the same `dict`. Default is `False`.
- **pass\_chat\_data** (*optional[bool]*) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the chat that the update was sent in. For each update in the same chat, it will be the same `dict`. Default is `False`.

`check_update` (*update*)

`handle_update` (*update, dispatcher*)

## telegram.ext.choseninlineresulthandler module

This module contains the `ChosenInlineResultHandler` class

**class** telegram.ext.choseninlineresulthandler.**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.

#### Parameters

- **callback** (*function*) – 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** (*optional[bool]*) – 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 *Updater* and *Dispatcher* that contains new updates which can be used to insert updates. Default is *False*.
- **pass\_job\_queue** (*optional[bool]*) – If set to *True*, a keyword argument called *job\_queue* will be passed to the callback function. It will be a *JobQueue* instance created by the *Updater* which can be used to schedule new jobs. Default is *False*.
- **pass\_user\_data** (*optional[bool]*) – If set to *True*, a keyword argument called *user\_data* will be passed to the callback function. It will be a *dict* you can use to keep any data related to the user that sent the update. For each update of the same user, it will be the same *dict*. Default is *False*.
- **pass\_chat\_data** (*optional[bool]*) – If set to *True*, a keyword argument called *chat\_data* will be passed to the callback function. It will be a *dict* you can use to keep any data related to the chat that the update was sent in. For each update in the same chat, it will be the same *dict*. Default is *False*.

**checkUpdate** (*\*args, \*\*kwargs*)

**check\_update** (*update*)

**handleUpdate** (*\*args, \*\*kwargs*)

**handle\_update** (*update, dispatcher*)

**m** = 'telegram.ChosenInlineResultHandler.'

## telegram.ext.conversationhandler module

This module contains the *ConversationHandler*

**class** telegram.ext.conversationhandler.**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 *CommandHandler* or *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 `MessageHandler` and `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 `CallbackHandler.END` or `-1`.

### Parameters

- **entry\_points** (*list*) – 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*) – A `dict[object: list[Handler]]` 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*) – 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** (*Optional[bool]*) – 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** (*Optional[float]*) – 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** (*Optional[list]*) – 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.

**END = -1**

**check\_update** (*update*)

**entry\_points = None**

*type* – list[telegram.ext.Handler]

**fallbacks = None**

*type* – list[telegram.ext.Handler]

**handle\_update** (*update, dispatcher*)

**per\_message = None**

*type* – dict[tuple: object]

**states = None**

*type* – dict[str: telegram.ext.Handler]

**timed\_out\_behavior = None**

*type* – list[telegram.ext.Handler]

**update\_state** (*new\_state, key*)



## telegram.ext.commandhandler module

This module contains the `CommandHandler` class

```
class telegram.ext.commandhandler.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.

### Parameters

- **command** (*str/list*) – The name of the command or list of command this handler should listen for.
- **callback** (*function*) – 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`) – 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).
- **allow\_edited** (*Optional[bool]*) – If the handler should also accept edited messages. Default is `False`
- **pass\_args** (*optional[bool]*) – If 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** (*optional[bool]*) – 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 `Updater` and `Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.
- **pass\_job\_queue** (*optional[bool]*) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `JobQueue` instance created by the `Updater` which can be used to schedule new jobs. Default is `False`.
- **pass\_user\_data** (*optional[bool]*) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the user that sent the update. For each update of the same user, it will be the same `dict`. Default is `False`.
- **pass\_chat\_data** (*optional[bool]*) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the chat that the update was sent in. For each update in the same chat, it will be the same `dict`. Default is `False`.

**check\_update** (*update*)

**handle\_update** (*update, dispatcher*)

## telegram.ext.inlinequeryhandler module

This module contains the `InlineQueryHandler` class

```
class telegram.ext.inlinequeryhandler.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.

### Parameters

- **callback** (*function*) – 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** (*optional[bool]*) – 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 `Updater` and `Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.
- **pass\_job\_queue** (*optional[bool]*) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `JobQueue` instance created by the `Updater` which can be used to schedule new jobs. Default is `False`.
- **pattern** (*optional[str or Pattern]*) – Optional regex pattern. If not `None` `re.match` is used to determine if an update should be handled by this handler.
- **pass\_groups** (*optional[bool]*) – If the callback should be passed the result of `re.match(pattern, query).groups()` as a keyword argument called `groups`. Default is `False`.
- **pass\_groupdict** (*optional[bool]*) – If the callback should be passed the result of `re.match(pattern, query).groupdict()` as a keyword argument called `groupdict`. Default is `False`.
- **pass\_user\_data** (*optional[bool]*) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the user that sent the update. For each update of the same user, it will be the same `dict`. Default is `False`.
- **pass\_chat\_data** (*optional[bool]*) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the chat that the update was sent in. For each update in the same chat, it will be the same `dict`. Default is `False`.

`checkUpdate` (*\*args, \*\*kwargs*)

`check_update` (*update*)

`handleUpdate` (*\*args, \*\*kwargs*)

`handle_update` (*update, dispatcher*)

`m` = `'telegram.InlineQueryHandler.'`

## telegram.ext.messagehandler module

This module contains the MessageHandler class

```
class telegram.ext.messagehandler.MessageHandler (filters,          callback,          al-
                                                  low_edited=False,
                                                  pass_update_queue=False,
                                                  pass_job_queue=False,
                                                  pass_user_data=False,
                                                  pass_chat_data=False,      mes-
                                                  sage_updates=True,        chan-
                                                  nel_post_updates=True,
                                                  edited_updates=False)
```

Bases: `telegram.ext.handler.Handler`

Handler class to handle telegram messages. Messages are Telegram Updates that do not contain a command. They might contain text, media or status updates.

### Parameters

- **filters** (`telegram.ext.BaseFilter`) – 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).
- **callback** (`function`) – 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** (`optional[bool]`) – If the handler should be passed the update queue as a keyword argument called `update_queue`. It can be used to insert updates. Default is `False`
- **pass\_user\_data** (`optional[bool]`) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the user that sent the update. For each update of the same user, it will be the same `dict`. Default is `False`.
- **pass\_chat\_data** (`optional[bool]`) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the chat that the update was sent in. For each update in the same chat, it will be the same `dict`. Default is `False`.
- **message\_updates** (`Optional[bool]`) – Should “normal” message updates be handled? Default is `True`.
- **allow\_edited** (`Optional[bool]`) – If the handler should also accept edited messages. Default is `False` - Deprecated. use edited updates instead.
- **channel\_post\_updates** (`Optional[bool]`) – Should channel posts updates be handled? Default is `True`.
- **edited\_updates** (`Optional[bool]`) – Should “edited” message updates be handled? Default is `False`.

**check\_update** (`update`)

**handle\_update** (`update`, `dispatcher`)

## telegram.ext.messagequeue module

A throughput-limiting message processor for Telegram bots

```
class telegram.ext.messagequeue.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.

#### Parameters

- **queue** (`queue.Queue`, optional) – used to pass callbacks to thread. Creates `queue.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.

**run()**

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

**stop** (`timeout=None`)

Used to gently stop processor and shutdown its thread.

**Parameterstimeout** (`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. If `timeout` set to `None`, blocks until processor is shut down. Defaults to `None`.

**Returns**`None`

```
exception telegram.ext.messagequeue.DelayQueueError
```

Bases: `exceptions.RuntimeError`

Indicates processing errors

```
class telegram.ext.messagequeue.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`'s, 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) – numer 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) – numer 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`.

**`_all_delayq`**

`telegram.ext.messagequeue.DelayQueue` – actual *DelayQueue* used for *all-type* callback processing

**`_group_delayq`**

`telegram.ext.messagequeue.DelayQueue` – actual *DelayQueue* used for *group-type* callback processing

**`start()`**

Method is used to manually start the *MessageQueue* processing

**Returns** `None`

**`stop(timeout=None)`**

Used to gently stop processor and shutdown its thread.

**Parameter** `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. If `timeout` set to `None`, blocks until processor is shut down. Defaults to `None`.

**Returns** `None`

`telegram.ext.messagequeue.queuedmessage` (*method*)

A decorator to be used with *telegram.bot.Bot send\** methods.

---

**Note:** As it probably wouldn't be a good idea to make this decorator a property, it had been coded as decorator function, so it implies that **first positional argument to wrapped MUST be self**.

---

The next object attributes are used by decorator:

`self._is_messages_queued_default`

`bool` – Value to provide class-defaults to *queued* kwarg if not provided during wrapped method call.

`self._msg_queue`

`telegram.ext.messagequeue.MessageQueue` – The actual *MessageQueue* used to delay outbound messages according to specified time-limits.

Wrapped method starts accepting the next kwargs:

**Parameters**

- **queued** (`bool`, optional) – if set to `True`, the *MessageQueue* is used to process output messages. Defaults to *self.\_is\_queued\_out*.
- **isgroup** (`bool`, optional) – if set to `True`, the message is meant to be group-type (as there's no obvious way to determine its type in other way at the moment). Group-type messages could have additional processing delay according to limits set in *self.\_out\_queue*. Defaults to `False`.

**Returns** Either `telegram.utils.promise.Promise` in case call is queued, or original method's return value if it's not.

## 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).

**filter** (*message*)

**class** telegram.ext.filters.**Filters**

Bases: object

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

**all** = <telegram.ext.filters.\_All object>

**audio** = <telegram.ext.filters.\_Audio object>

**command** = <telegram.ext.filters.\_Command object>

**contact** = <telegram.ext.filters.\_Contact object>

**document** = <telegram.ext.filters.\_Document object>

**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*.

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

Returns: function to use as filter

**filter** (*message*)

**Filters.forwarded** = <telegram.ext.filters.\_Forwarded object>

**Filters.game** = <telegram.ext.filters.\_Game object>

**Filters.group** = <telegram.ext.filters.\_Group object>

**Filters.location** = <telegram.ext.filters.\_Location object>

```

Filters.photo = <telegram.ext.filters._Photo object>
Filters.private = <telegram.ext.filters._Private object>
Filters.reply = <telegram.ext.filters._Reply object>
Filters.status_update = <telegram.ext.filters._StatusUpdate object>
Filters.sticker = <telegram.ext.filters._Sticker object>
Filters.text = <telegram.ext.filters._Text object>
Filters.venue = <telegram.ext.filters._Venue object>
Filters.video = <telegram.ext.filters._Video object>
Filters.voice = <telegram.ext.filters._Voice object>

```

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

Bases: *telegram.ext.filters.BaseFilter*

Represents a filter that has been inverted.

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

**filter** (*message*)

**class** telegram.ext.filters.**MergedFilter** (*base\_filter*, *and\_filter=None*, *or\_filter=None*)

Bases: *telegram.ext.filters.BaseFilter*

Represents a filter consisting of two other filters.

**Parameters**

• **base\_filter** – Filter 1 of the merged filter

• **and\_filter** – Optional filter to “and” with *base\_filter*. Mutually exclusive with *or\_filter*.

• **or\_filter** – Optional filter to “or” with *base\_filter*. Mutually exclusive with *and\_filter*.

**filter** (*message*)

## telegram.ext.regexhandler module

This module contains the `RegexHandler` class

**class** telegram.ext.regexhandler.**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*, *message\_*  
*updates=True*, *channel\_*  
*post\_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.

**Parameters**

• **pattern** (*str* or *Pattern*) – The regex pattern.

- **callback** (*function*) – 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** (*optional[bool]*) – If the callback should be passed the result of `re.match(pattern, text).groups()` as a keyword argument called `groups`. Default is `False`
- **pass\_groupdict** (*optional[bool]*) – If the callback should be passed the result of `re.match(pattern, text).groupdict()` as a keyword argument called `groupdict`. Default is `False`
- **pass\_update\_queue** (*optional[bool]*) – 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 `Updater` and `Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.
- **pass\_job\_queue** (*optional[bool]*) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `JobQueue` instance created by the `Updater` which can be used to schedule new jobs. Default is `False`.
- **pass\_user\_data** (*optional[bool]*) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the user that sent the update. For each update of the same user, it will be the same `dict`. Default is `False`.
- **pass\_chat\_data** (*optional[bool]*) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the chat that the update was sent in. For each update in the same chat, it will be the same `dict`. Default is `False`.

`checkUpdate` (*\*args, \*\*kwargs*)

`check_update` (*update*)

`handleUpdate` (*\*args, \*\*kwargs*)

`handle_update` (*update, dispatcher*)

`m = 'telegram.RegexHandler.'`

## telegram.ext.stringcommandhandler module

This module contains the `StringCommandHandler` class

```
class telegram.ext.stringcommandhandler.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 `/`.

### Parameters

- **command** (*str*) – The name of the command this handler should listen for.
- **callback** (*function*) – 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\_args** (*optional[bool]*) – If 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 white-space characters. Default is `False`

- **`pass_update_queue`** (*optional[bool]*) – 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 `Updater` and `Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.

- **`pass_job_queue`** (*optional[bool]*) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `JobQueue` instance created by the `Updater` which can be used to schedule new jobs. Default is `False`.

**`check_update`** (*update*)

**`handle_update`** (*update, dispatcher*)

## telegram.ext.stringregexhandler module

This module contains the `StringRegexHandler` class

```
class telegram.ext.stringregexhandler.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. It uses a regular expression to check 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.

### Parameters

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

- **`callback`** (*function*) – 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`** (*optional[bool]*) – If the callback should be passed the result of `re.match(pattern, update).groups()` as a keyword argument called `groups`. Default is `False`

- **`pass_groupdict`** (*optional[bool]*) – If the callback should be passed the result of `re.match(pattern, update).groupdict()` as a keyword argument called `groupdict`. Default is `False`

- **`pass_update_queue`** (*optional[bool]*) – 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 `Updater` and `Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.

- **`pass_job_queue`** (*optional[bool]*) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `JobQueue` instance created by the `Updater` which can be used to schedule new jobs. Default is `False`.

**`check_update`** (*update*)

**`handle_update`** (*update, dispatcher*)

## telegram.ext.typehandler module

This module contains the TypeHandler class

```
class telegram.ext.typehandler.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.

### Parameters

- **type** (*type*) – The type of updates this handler should process, as determined by `isinstance`
- **callback** (*function*) – 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** (*optional[bool]*) – Use `type` instead of `isinstance`. Default is `False`
- **pass\_update\_queue** (*optional[bool]*) – 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 `Updater` and `Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.
- **pass\_job\_queue** (*optional[bool]*) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `JobQueue` instance created by the `Updater` which can be used to schedule new jobs. Default is `False`.

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

```
check_update (update)
```

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

```
handle_update (update, dispatcher)
```

```
m = 'telegram.TypeHandler.'
```

## Module contents

Extensions over the Telegram Bot API to facilitate bot making

```
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.

### 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** (*Optional[telegram.ext.JobQueue]*) – The `JobQueue` instance to pass onto handler callbacks
- **workers** (*Optional[int]*) – Number of maximum concurrent worker threads for the `@run_async` decorator

```
add_error_handler (callback)
```

Registers an error handler in the `Dispatcher`.

**Parametershandler** (*function*) – 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.

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 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** (`Optional[int]`) – The group identifier. Default is 0

**dispatch\_error** (*update, error*)

Dispatches an error.

#### Parameters

- **update** (*object*) – The update that caused the error
- **error** (`telegram.TelegramError`) – The Telegram error that was raised.

**classmethod get\_instance** ()

Get the singleton instance of this class.

Returns Dispatcher

**has\_running\_threads**

**logger** = <logging.Logger object>

**process\_update** (*update*)

Processes a single update.

**Parametersupdate** (*object*) –

**remove\_error\_handler** (*callback*)

De-registers an error handler.

**Parametershandler** (*function*) –

**remove\_handler** (*handler, group=0*)

Remove a handler from the specified group

#### Parameters

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

**run\_async** (*func, \*args, \*\*kwargs*)

Queue a function (with given args/kwags) to be run asynchronously.

#### Parameters

- **func** (*function*) – The function to run in the thread.
- **args** (`Optional[tuple]`) – Arguments to *func*.
- **kwargs** (`Optional[dict]`) – Keyword arguments to *func*.

**Returns**Promise

**start** ()

Thread target of thread 'dispatcher'. Runs in background and processes the update queue.

**stop** ()

Stops the thread

**class** telegram.ext . **JobQueue** (*bot*, *prevent\_autostart=None*)

Bases: object

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

**queue**

*PriorityQueue*

**bot**

*telegram.Bot*

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

**Deprecated: 5.2***prevent\_autostart* (Optional[bool]): 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.

**Parameters**

- **job** (*telegram.ext.Job*) – The Job instance representing the new job
- **next\_t** (Optional[*int*, *float*, *datetime.timedelta*, *datetime.datetime*, *datetime.time*]) – 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** (*function*) – The callback function that should be executed by the new job. It should take two parameters *bot* and *job*, where *job* is the Job instance. It can be used to access its *context* or terminate the job.
- **time** (*datetime.time*) – Time of day at which the job should run.
- **days** (Optional[*tuple[int]*]) – Defines on which days of the week the job should run.
- **to Days.EVERY\_DAY** (Defaults) –
- **context** (Optional[*object*]) – Additional data needed for the callback function. Can be accessed through *job.context* in the callback. Defaults to None
- **name** (Optional[*str*]) – 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***Job*

**run\_once** (*callback, when, context=None, name=None*)

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

#### Parameters

- **callback** (*function*) – The callback function that should be executed by the new job. It should take two parameters `bot` and `job`, where `job` is the `Job` instance. It can be used to access its `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** (*Optional[object]*) – Additional data needed for the callback function. Can be accessed through `job.context` in the callback. Defaults to `None`
- **name** (*Optional[str]*) – 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** `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** (*function*) – The callback function that should be executed by the new job. It should take two parameters `bot` and `job`, where `job` is the `Job` instance. It can be used to access its `context`, terminate the job or change its interval.
- **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*) –
  - `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** (*Optional[object]*) – Additional data needed for the callback function. Can be accessed through `job.context` in the callback. Defaults to `None`
- **name** (*Optional[str]*) – 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**`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.

**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**

*function* – The function that the job executes when it’s due

**interval**

*int, float, datetime.timedelta* – The interval in which the job runs

**days**

*tuple[int]* – A tuple of `int` values that determine on which days of the week the job runs

**repeat**

*bool* – If the job runs periodically or only once

**name**

*str* – The name of this job

**job\_queue**

*JobQueue* – The `JobQueue` this job belongs to

**enabled**

*bool* – Boolean property that decides if this job is currently active

#### Parameters

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

**days**

**enabled**

**interval****interval\_seconds****job\_queue***rtype* – JobQueue**removed****repeat****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.

**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 Dispatcher class, 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.

Attributes:

**Parameters**

- **token** (*Optional[str]*) – The bot’s token given by the @BotFather
- **base\_url** (*Optional[str]*) –
- **workers** (*Optional[int]*) – Amount of threads in the thread pool for functions decorated with @run\_async
- **bot** (*Optional[Bot]*) – A pre-initialized bot instance. If a pre-initizlied 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** (*Optional[function]*) – Takes *signum, frame* as positional arguments. This will be called when a signal is received, defaults are (SIGINT, SIGTERM, SIGABRT) setable with `Updater.idle(stop_signals=(signals))`
- **request\_kwargs** (*Optional[dict]*) – Keyword args to control the creation of a request object (ignored if *bot* argument is used).

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

**idle** (*stop\_signals=(2, 15, 6)*)

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

**Parameters** **stop\_signals** – 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)

**signal\_handler** (*signum, frame*)

**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** (*Optional[float]*) – Time to wait between polling updates from Telegram in

- **Default is 0.0** (*seconds.*) –
- **timeout** (*Optional[float]*) – Passed to Bot.getUpdates
- **network\_delay** – Deprecated. Will be honoured as *read\_latency* for a while but will be removed in the future.
- **clean** (*Optional[bool]*) – Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is False.
- **bootstrap\_retries** (*Optional[int]*) – 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** (*Optional[list[str]]*) – Passed to Bot.getUpdates
- **read\_latency** (*Optional[float|int]*) – Grace time in seconds for receiving the reply from server. Will be added to the *timeout* value and used as the read timeout from server (Default: 2).

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

**Return type**Queue

**start\_webhook** (*listen='127.0.0.1', port=80, url\_path='', cert=None, key=None, clean=False, bootstrap\_retries=0, webhook\_url=None, allowed\_updates=None*)

Starts a small http server to listen for updates via webhook. If cert and key are not provided, the webhook will be started directly on [http://listen:port/url\\_path](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](https://listen:port/url_path)

**Parameters**

- **listen** (*Optional[str]*) – IP-Address to listen on
- **port** (*Optional[int]*) – Port the bot should be listening on
- **url\_path** (*Optional[str]*) – Path inside url
- **cert** (*Optional[str]*) – Path to the SSL certificate file
- **key** (*Optional[str]*) – Path to the SSL key file
- **clean** (*Optional[bool]*) – Whether to clean any pending updates on Telegram servers before actually starting the webhook. Default is False.
- **bootstrap\_retries** (*Optional[int]*) – 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** (*Optional[str]*) – Explicitly specify the webhook url. Useful behind NAT, reverse proxy, etc. Default is derived from *listen, port & url\_path*.
- **allowed\_updates** (*Optional[list[str]]*) – Passed to Bot.setWebhook

**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



```
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.

### Parameters

- **callback** (*function*) – 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** (*optional[bool]*) – 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 `Updater` and `Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.
- **pass\_job\_queue** (*optional[bool]*) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `JobQueue` instance created by the `Updater` which can be used to schedule new jobs. Default is `False`.
- **pattern** (*optional[str or Pattern]*) – Optional regex pattern. If not `None` `re.match` is used to determine if an update should be handled by this handler.
- **pass\_groups** (*optional[bool]*) – 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** (*optional[bool]*) – 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** (*optional[bool]*) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the user that sent the update. For each update of the same user, it will be the same `dict`. Default is `False`.
- **pass\_chat\_data** (*optional[bool]*) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the chat that the update was sent in. For each update in the same chat, it will be the same `dict`. Default is `False`.

**check\_update** (*update*)

**handle\_update** (*update, dispatcher*)

```
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.

### Parameters

- **callback** (*function*) – 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** (*optional[bool]*) – 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 `Updater` and `Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.

•**pass\_job\_queue** (*optional[bool]*) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `JobQueue` instance created by the `Updater` which can be used to schedule new jobs. Default is `False`.

•**pass\_user\_data** (*optional[bool]*) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the user that sent the update. For each update of the same user, it will be the same `dict`. Default is `False`.

•**pass\_chat\_data** (*optional[bool]*) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the chat that the update was sent in. For each update in the same chat, it will be the same `dict`. Default is `False`.

`checkUpdate` (\*args, \*\*kwargs)

`check_update` (update)

`handleUpdate` (\*args, \*\*kwargs)

`handle_update` (update, dispatcher)

m = 'telegram.ChosenInlineResultHandler.'

```
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.

#### Parameters

•**command** (*str/list*) – The name of the command or list of command this handler should listen for.

•**callback** (*function*) – 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`) – 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).

•**allow\_edited** (*Optional[bool]*) – If the handler should also accept edited messages. Default is `False`

•**pass\_args** (*optional[bool]*) – If 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** (*optional[bool]*) – 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 `Updater` and `Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.

•**pass\_job\_queue** (*optional[bool]*) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `JobQueue`

instance created by the `Updater` which can be used to schedule new jobs. Default is `False`.

- **`pass_user_data`** (*optional[bool]*) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the user that sent the update. For each update of the same user, it will be the same `dict`. Default is `False`.

- **`pass_chat_data`** (*optional[bool]*) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the chat that the update was sent in. For each update in the same chat, it will be the same `dict`. Default is `False`.

**`check_update`** (*update*)

**`handle_update`** (*update, dispatcher*)

**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. You can create your own handlers by inheriting from this class.

#### Parameters

- **`callback`** (*function*) – 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`** (*optional[bool]*) – 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 `Updater` and `Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.

- **`pass_job_queue`** (*optional[bool]*) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `JobQueue` instance created by the `Updater` which can be used to schedule new jobs. Default is `False`.

- **`pass_user_data`** (*optional[bool]*) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the user that sent the update. For each update of the same user, it will be the same `dict`. Default is `False`.

- **`pass_chat_data`** (*optional[bool]*) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the chat that the update was sent in. For each update in the same chat, it will be the same `dict`. 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` (*object*) – 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`) –

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

```
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.

### Parameters

- **callback** (*function*) – 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** (*optional[bool]*) – 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 `Updater` and `Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.
- **pass\_job\_queue** (*optional[bool]*) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `JobQueue` instance created by the `Updater` which can be used to schedule new jobs. Default is `False`.
- **pattern** (*optional[str or Pattern]*) – Optional regex pattern. If not `None` `re.match` is used to determine if an update should be handled by this handler.
- **pass\_groups** (*optional[bool]*) – If the callback should be passed the result of `re.match(pattern, query).groups()` as a keyword argument called `groups`. Default is `False`
- **pass\_groupdict** (*optional[bool]*) – If the callback should be passed the result of `re.match(pattern, query).groupdict()` as a keyword argument called `groupdict`. Default is `False`
- **pass\_user\_data** (*optional[bool]*) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the user that sent the update. For each update of the same user, it will be the same `dict`. Default is `False`.
- **pass\_chat\_data** (*optional[bool]*) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the chat that the update was sent in. For each update in the same chat, it will be the same `dict`. Default is `False`.

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

```
check_update(update)
```

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

```
handle_update(update, dispatcher)
```

```
m = 'telegram.InlineQueryHandler.'
```

```
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. Messages are Telegram Updates that do not contain a command. They might contain text, media or status updates.

### Parameters

- **filters** (`telegram.ext.BaseFilter`) – 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).
- **callback** (`function`) – 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** (`optional[bool]`) – If the handler should be passed the update queue as a keyword argument called `update_queue`. It can be used to insert updates. Default is `False`
- **pass\_user\_data** (`optional[bool]`) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the user that sent the update. For each update of the same user, it will be the same `dict`. Default is `False`.
- **pass\_chat\_data** (`optional[bool]`) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the chat that the update was sent in. For each update in the same chat, it will be the same `dict`. Default is `False`.
- **message\_updates** (`Optional[bool]`) – Should “normal” message updates be handled? Default is `True`.
- **allow\_edited** (`Optional[bool]`) – If the handler should also accept edited messages. Default is `False` - Deprecated. use `edited_updates` instead.
- **channel\_post\_updates** (`Optional[bool]`) – Should channel posts updates be handled? Default is `True`.
- **edited\_updates** (`Optional[bool]`) – Should “edited” message updates be handled? Default is `False`.

**check\_update** (`update`)

**handle\_update** (`update`, `dispatcher`)

**class** `telegram.ext.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).

**filter** (*message*)

**class** telegram.ext.**Filters**

Bases: object

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

**all** = <telegram.ext.filters.\_All object>

**audio** = <telegram.ext.filters.\_Audio object>

**command** = <telegram.ext.filters.\_Command object>

**contact** = <telegram.ext.filters.\_Contact object>

**document** = <telegram.ext.filters.\_Document object>

**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*.

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

Returns: function to use as filter

**filter** (*message*)

**Filters.forwarded** = <telegram.ext.filters.\_Forwarded object>

**Filters.game** = <telegram.ext.filters.\_Game object>

**Filters.group** = <telegram.ext.filters.\_Group object>

**Filters.location** = <telegram.ext.filters.\_Location object>

**Filters.photo** = <telegram.ext.filters.\_Photo object>

**Filters.private** = <telegram.ext.filters.\_Private object>

**Filters.reply** = <telegram.ext.filters.\_Reply object>

**Filters.status\_update** = <telegram.ext.filters.\_StatusUpdate object>

**Filters.sticker** = <telegram.ext.filters.\_Sticker object>

**Filters.text** = <telegram.ext.filters.\_Text object>

**Filters.venue** = <telegram.ext.filters.\_Venue object>

**Filters.video** = <telegram.ext.filters.\_Video object>

**Filters.voice** = <telegram.ext.filters.\_Voice object>

**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, message\_updates=True, channel\_post\_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.

### Parameters

- **pattern** (*str or Pattern*) – The regex pattern.
- **callback** (*function*) – 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** (*optional[bool]*) – If the callback should be passed the result of `re.match(pattern, text).groups()` as a keyword argument called `groups`. Default is `False`
- **pass\_groupdict** (*optional[bool]*) – If the callback should be passed the result of `re.match(pattern, text).groupdict()` as a keyword argument called `groupdict`. Default is `False`
- **pass\_update\_queue** (*optional[bool]*) – 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 `Updater` and `Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.
- **pass\_job\_queue** (*optional[bool]*) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `JobQueue` instance created by the `Updater` which can be used to schedule new jobs. Default is `False`.
- **pass\_user\_data** (*optional[bool]*) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the user that sent the update. For each update of the same user, it will be the same `dict`. Default is `False`.
- **pass\_chat\_data** (*optional[bool]*) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. It will be a `dict` you can use to keep any data related to the chat that the update was sent in. For each update in the same chat, it will be the same `dict`. Default is `False`.

`checkUpdate` (*\*args, \*\*kwargs*)

`check_update` (*update*)

`handleUpdate` (*\*args, \*\*kwargs*)

`handle_update` (*update, dispatcher*)

`m = 'telegram.RegexHandler.'`

```
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 `/`.

### Parameters

- **command** (*str*) – The name of the command this handler should listen for.
- **callback** (*function*) – 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\_args** (*optional[bool]*) – If 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** (*optional[bool]*) – 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 `Updater` and `Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.

• **pass\_job\_queue** (*optional[bool]*) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `JobQueue` instance created by the `Updater` which can be used to schedule new jobs. Default is `False`.

**check\_update** (*update*)

**handle\_update** (*update, dispatcher*)

```
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. It uses a regular expression to check 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.

#### Parameters

• **pattern** (*str or Pattern*) – The regex pattern.

• **callback** (*function*) – 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** (*optional[bool]*) – If the callback should be passed the result of `re.match(pattern, update).groups()` as a keyword argument called `groups`. Default is `False`

• **pass\_groupdict** (*optional[bool]*) – If the callback should be passed the result of `re.match(pattern, update).groupdict()` as a keyword argument called `groupdict`. Default is `False`

• **pass\_update\_queue** (*optional[bool]*) – 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 `Updater` and `Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.

• **pass\_job\_queue** (*optional[bool]*) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `JobQueue` instance created by the `Updater` which can be used to schedule new jobs. Default is `False`.

**check\_update** (*update*)

**handle\_update** (*update, dispatcher*)

```
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.

#### Parameters

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



- **callback** (*function*) – 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** (*optional[bool]*) – Use `type` instead of `isinstance`. Default is `False`
- **pass\_update\_queue** (*optional[bool]*) – 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 `Updater` and `Dispatcher` that contains new updates which can be used to insert updates. Default is `False`.
- **pass\_job\_queue** (*optional[bool]*) – If set to `True`, a keyword argument called `job_queue` will be passed to the callback function. It will be a `JobQueue` instance created by the `Updater` which can be used to schedule new jobs. Default is `False`.

`checkUpdate` (*\*args, \*\*kwargs*)

`check_update` (*update*)

`handleUpdate` (*\*args, \*\*kwargs*)

`handle_update` (*update, dispatcher*)

`m = 'telegram.TypeHandler.'`

```
class telegram.ext.ConversationHandler (entry_points, states, fallbacks, al-
                                     low_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 `CommandHandler` or `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 `MessageHandler` and `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 `CallbackHandler.END` or `-1`.

### Parameters

- **entry\_points** (*list*) – 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*) – A `dict[object: list[Handler]]` 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*) – 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** (*Optional[bool]*) – 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** (*Optional[float]*) – 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** (*Optional[list]*) – 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.

**END = -1**

**check\_update** (*update*)

**handle\_update** (*update, dispatcher*)

**update\_state** (*new\_state, key*)

## telegram.animation module

This module contains an object that represents a Telegram Animation.

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

This object represents a Telegram Animation.

**file\_id**

*str* – Unique file identifier.

### Keyword Arguments

- **thumb** (*Optional[telegram.PhotoSize]*) – Animation thumbnail as defined by sender.
- **file\_name** (*Optional[str]*) – Original animation filename as defined by sender.
- **mime\_type** (*Optional[str]*) – MIME type of the file as defined by sender.
- **file\_size** (*Optional[int]*) – File size.

**static de\_json** (*data, bot*)

### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

### Returns

Return type *telegram.Game*

## telegram.audio module

This module contains an object that represents a Telegram Audio.

**class** telegram.audio.**Audio** (*file\_id*, *duration*, *performer=None*, *title=None*, *mime\_type=None*, *file\_size=None*, *\*\*kwargs*)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram Audio.

**file\_id**  
*str*

**duration**  
*int*

**performer**  
*str*

**title**  
*str*

**mime\_type**  
*str*

**file\_size**  
*int*

### Parameters

- **file\_id** (*str*) –
- **duration** (*int*) –
- **performer** (*Optional[str]*) –
- **title** (*Optional[str]*) –
- **mime\_type** (*Optional[str]*) –
- **file\_size** (*Optional[int]*) –
- **\*\*kwargs** – Arbitrary keyword arguments.

**static de\_json** (*data*, *bot*)

### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

### Returns

Return type *telegram.Audio*

## telegram.base module

Base class for Telegram Objects.

**class** telegram.base.**TelegramObject**

Bases: *object*

Base class for most telegram objects.

**static de\_json** (*data*, *bot*)

### Parameters

- **data** (*dict*) –

•**bot** (`telegram.Bot`) –

**Returns**

**Return typedict**

`to_dict()`

**Returns**

**Return typedict**

`to_json()`

**Returns**

**Return typestr**

## telegram.bot module

This module contains an object that represents a Telegram Bot.

**class** `telegram.bot.Bot` (*token*, *base\_url=None*, *base\_file\_url=None*, *request=None*)

Bases: `telegram.base.TelegramObject`

This object represents a Telegram Bot.

**id**

*int* – Unique identifier for this bot.

**first\_name**

*str* – Bot’s first name.

**last\_name**

*str* – Bot’s last name.

**username**

*str* – Bot’s username.

**name**

*str* – Bot’s @username.

### Parameters

- token** (*str*) – Bot’s unique authentication.
- base\_url** (*Optional[str]*) – Telegram Bot API service URL.
- base\_file\_url** (*Optional[str]*) – Telegram Bot API file URL.
- request** (*Optional[Request]*) – Pre initialized *Request* class.

**answerCallbackQuery** (*\*args*, *\*\*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.

### Parameters

- callback\_query\_id** (*str*) – Unique identifier for the query to be answered.
- text** (*Optional[str]*) – Text of the notification. If not specified, nothing will be shown to the user.
- show\_alert** (*Optional[bool]*) – 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** (*Optional[str]*) – URL that will be opened by the user’s client.

- **cache\_time** (*Optional[int]*) – The maximum amount of time in seconds that the result of the callback query may be cached client-side. Telegram apps will support caching starting in version 3.14. Defaults to 0.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type** *bool*

**Raises** *telegram.TelegramError*

**answerInlineQuery** (*\*args, \*\*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** (*Optional[int]*) – The maximum amount of time the result of the inline query may be cached on the server.
- **is\_personal** (*Optional[bool]*) – 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** (*Optional[str]*) – 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** (*Optional[str]*) – 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** (*Optional[str]*) – Parameter for the start message sent to the bot when user presses the switch button.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type** *bool*

**Raises** *telegram.TelegramError*

**answer\_callback\_query** (*\*args, \*\*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.

**Parameters**

- **callback\_query\_id** (*str*) – Unique identifier for the query to be answered.
- **text** (*Optional[str]*) – Text of the notification. If not specified, nothing will be shown to the user.
- **show\_alert** (*Optional[bool]*) – 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** (*Optional[str]*) – URL that will be opened by the user’s client.
- **cache\_time** (*Optional[int]*) – The maximum amount of time in seconds that the result of the callback query may be cached client-side. Telegram apps will support caching starting in version 3.14. Defaults to 0.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type** bool

**Raises** *telegram.TelegramError*

**answer\_inline\_query** (*\*args, \*\*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** (*Optional[int]*) – The maximum amount of time the result of the inline query may be cached on the server.
- **is\_personal** (*Optional[bool]*) – 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** (*Optional[str]*) – 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** (*Optional[str]*) – 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** (*Optional[str]*) – Parameter for the start message sent to the bot when user presses the switch button.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type** bool

**Raises** *telegram.TelegramError*

**deleteMessage** (*\*args, \*\*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*) – Unique message identifier.

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

**Return type**bool

**Raises***telegram.TelegramError*

**deleteWebhook** (*\*args, \*\*kwargs*)

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

**Parameters**

- timeout** (*Optional[float]*) – If this value is specified, use it as the definitive timeout (in seconds) for `urlopen()` operations.

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

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

**Return type**bool

**Raises***telegram.TelegramError*

**delete\_message** (*\*args, \*\*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*) – Unique message identifier.

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

**Return type**bool

**Raises***telegram.TelegramError*

**delete\_webhook** (*\*args, \*\*kwargs*)

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

**Parameters**

- timeout** (*Optional[float]*) – If this value is specified, use it as the definitive timeout (in seconds) for `urlopen()` operations.

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

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

**Return type**bool

**Raises***telegram.TelegramError*

**editMessageCaption** (*\*args, \*\*kwargs*)

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

**Parameters**

- **chat\_id** (*Optional[int|str]*) – Required if `inline_message_id` is not specified. Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **message\_id** (*Optional[int]*) – Required if `inline_message_id` is not specified. Unique identifier of the sent message.
- **inline\_message\_id** (*Optional[str]*) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.
- **caption** (*Optional[str]*) – New caption of the message.
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) – A JSON-serialized object for an inline keyboard.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*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`

**editMessageReplyMarkup** (*\*args, \*\*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** (*Optional[int|str]*) – Required if `inline_message_id` is not specified. Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **message\_id** (*Optional[int]*) – Required if `inline_message_id` is not specified. Unique identifier of the sent message.
- **inline\_message\_id** (*Optional[str]*) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) – A JSON-serialized object for an inline keyboard.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*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`

**editMessageText** (*\*args, \*\*kwargs*)

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

#### Parameters

- **text** (*str*) – New text of the message.



- **chat\_id** (*Optional[int|str]*) – Required if `inline_message_id` is not specified. Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **message\_id** (*Optional[int]*) – Required if `inline_message_id` is not specified. Unique identifier of the sent message.
- **inline\_message\_id** (*Optional[str]*) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.
- **parse\_mode** (:class:`telegram.ParseMode`|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.
- **disable\_web\_page\_preview** (*bool*) – Disables link previews for links in this message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*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_caption(*args, **kwargs)`

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

#### Parameters

- **chat\_id** (*Optional[int|str]*) – Required if `inline_message_id` is not specified. Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **message\_id** (*Optional[int]*) – Required if `inline_message_id` is not specified. Unique identifier of the sent message.
- **inline\_message\_id** (*Optional[str]*) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.
- **caption** (*Optional[str]*) – New caption of the message.
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) – A JSON-serialized object for an inline keyboard.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*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\_reply\_markup** (\*args, \*\*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** (*Optional[int|str]*) – Required if `inline_message_id` is not specified. Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **message\_id** (*Optional[int]*) – Required if `inline_message_id` is not specified. Unique identifier of the sent message.
- **inline\_message\_id** (*Optional[str]*) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) – A JSON-serialized object for an inline keyboard.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*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\_text** (\*args, \*\*kwargs)

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

#### Parameters

- **text** (*str*) – New text of the message.
- **chat\_id** (*Optional[int|str]*) – Required if `inline_message_id` is not specified. Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **message\_id** (*Optional[int]*) – Required if `inline_message_id` is not specified. Unique identifier of the sent message.
- **inline\_message\_id** (*Optional[str]*) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.
- **parse\_mode** (:class:`*telegram.ParseMode*`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.
- **disable\_web\_page\_preview** (*bool*) – Disables link previews for links in this message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*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`

**forwardMessage** (*\*args*, *\*\*kwargs*)

Use this method to forward messages of any kind.

#### Parameters

- **chat\_id** (*int* / *str*) – Unique identifier for the message recipient - Chat id.
- **from\_chat\_id** (*int* / *str*) – Unique identifier for the chat where the original message was sent - Chat id.
- **message\_id** (*int*) – Unique message identifier.
- **disable\_notification** (*Optional*[*bool*]) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **timeout** (*Optional*[*int* / *float*]) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message forwarded.

**Return type** `telegram.Message`

**Raises** `telegram.TelegramError`

**forward\_message** (*\*args*, *\*\*kwargs*)

Use this method to forward messages of any kind.

#### Parameters

- **chat\_id** (*int* / *str*) – Unique identifier for the message recipient - Chat id.
- **from\_chat\_id** (*int* / *str*) – Unique identifier for the chat where the original message was sent - Chat id.
- **message\_id** (*int*) – Unique message identifier.
- **disable\_notification** (*Optional*[*bool*]) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **timeout** (*Optional*[*int* / *float*]) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message forwarded.

**Return type** `telegram.Message`

**Raises** `telegram.TelegramError`

**getChat** (*\*args*, *\*\*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** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

**Returns**On success, *telegram.Chat* is returned.

**Return type***telegram.Chat*

**Raises***telegram.TelegramError*

**getChatAdministrators** (*\*args, \*\*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** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

**Returns**A list of chat member objects.

**Return type**list[*telegram.ChatMember*]

**Raises***telegram.TelegramError*

**getChatMember** (*\*args, \*\*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** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

**Returns**On success, chat member object is returned.

**Return type***telegram.ChatMember*

**Raises***telegram.TelegramError*

**getChatMembersCount** (*\*args, \*\*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** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

**Returns**On success, an *int* is returned.

**Return type** `int`**Raises** `telegram.TelegramError`**getFile** (\*args, \*\*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.

**Parameters**

- **file\_id** (*str*) – File identifier to get info about.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, a `telegram.File` object is returned.**Return type** `telegram.File`**Raises** `telegram.TelegramError`**getGameHighScores** (*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.

**Parameters**

- **user\_id** (*int*) – User identifier.
- **chat\_id** (*Optional[int|str]*) – Required if *inline\_message\_id* is not specified. Unique identifier for the target chat (or username of the target channel in the format `@channelusername`)
- **message\_id** (*Optional[int]*) – Required if *inline\_message\_id* is not specified. Identifier of the sent message.
- **inline\_message\_id** (*Optional[str]*) – Required if *chat\_id* and *message\_id* are not specified. Identifier of the inline message.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

**Returns**

Scores of the specified user and several of his neighbors in a game.

**Return type** `list[telegram.GameHighScore]`**getMe** (\*args, \*\*kwargs)

A simple method for testing your bot's auth token.

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

**Returns**

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

**Return type** `telegram.User`**Raises** `telegram.TelegramError`**getUpdates** (\*args, \*\*kwargs)

Use this method to receive incoming updates using long polling.

**Parameters**

- **offset** (*Optional[int]*) – 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`.
- **limit** (*Optional[int]*) – Limits the number of updates to be retrieved. Values between 1-100 are accepted. Defaults to 100.
- **allowed\_updates** (*Optional[list[str]]*) – 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 `setWebhook`, so unwanted updates may be received for a short period of time.
- **timeout** (*Optional[int]*) – Timeout in seconds for long polling. Defaults to 0, i.e. usual short polling. Be careful not to set this timeout too high, as the connection might be dropped and there's no way of knowing it immediately (so most likely the failure will be detected after the timeout had passed).
- **network\_delay** – Deprecated. Will be honoured as *read\_latency* for a while but will be removed in the future.
- **read\_latency** (*Optional[float|int]*) – 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).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

## Notes

The main problem with long polling is that a connection will be dropped and we won't be getting the notification in time for it. For that, we need to use long polling, but not too long as well read latency which is short, but not too short. Long polling improves performance, but if it's too long and the connection is dropped on many cases we won't know the connection dropped before the long polling timeout and the read latency time had passed. If you experience connection timeouts, you should tune these settings.

**Returns** `list[telegram.Update]`

**Raises** `telegram.TelegramError`

**getUserProfilePhotos** (*\*args, \*\*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** (*Optional[int]*) – Sequential number of the first photo to be returned. By default, all photos are returned.
- **limit** (*Optional[int]*) – Limits the number of photos to be retrieved. Values between 1-100 are accepted. Defaults to 100.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

### Returns

A list of user profile photos objects is returned.

**Return type**list[*telegram.UserProfilePhotos*]

**Raises***telegram.TelegramError*

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

Use this method to get current webhook status.

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

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

**Returns**class: *telegram.WebhookInfo*

**get\_chat** (*\*args*, *\*\*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** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

**Returns**On success, *telegram.Chat* is returned.

**Return type***telegram.Chat*

**Raises***telegram.TelegramError*

**get\_chat\_administrators** (*\*args*, *\*\*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** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

**Returns**A list of chat member objects.

**Return type**list[*telegram.ChatMember*]

**Raises***telegram.TelegramError*

**get\_chat\_member** (*\*args*, *\*\*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** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

**Returns** On success, chat member object is returned.

**Return type** *telegram.ChatMember*

**Raises** *telegram.TelegramError*

**get\_chat\_members\_count** (*\*args, \*\*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** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

**Returns** On success, an *int* is returned.

**Return type** *int*

**Raises** *telegram.TelegramError*

**get\_file** (*\*args, \*\*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.

**Parameters**

- **file\_id** (*str*) – File identifier to get info about.

- **timeout** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

**Returns** On success, a *telegram.File* object is returned.

**Return type** *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.

**Parameters**

- **user\_id** (*int*) – User identifier.

- **chat\_id** (*Optional[int/str]*) – Required if *inline\_message\_id* is not specified. Unique identifier for the target chat (or username of the target channel in the format @channelusername)

- **message\_id** (*Optional[int]*) – Required if *inline\_message\_id* is not specified. Identifier of the sent message.

- **inline\_message\_id** (*Optional[str]*) – Required if *chat\_id* and *message\_id* are not specified. Identifier of the inline message.



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

#### Returns

Scores of the specified user and several of his neighbors in a game.

Return type `list[telegram.GameHighScore]`

**get\_me** (\*args, \*\*kwargs)

A simple method for testing your bot's auth token.

- **parametersttimeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

#### Returns

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

Return type `telegram.User`

Raises `telegram.TelegramError`

**get\_updates** (\*args, \*\*kwargs)

Use this method to receive incoming updates using long polling.

#### Parameters

- **offset** (*Optional[int]*) – 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`.
- **limit** (*Optional[int]*) – Limits the number of updates to be retrieved. Values between 1-100 are accepted. Defaults to 100.
- **allowed\_updates** (*Optional[list[str]]*) – 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 `setWebhook`, so unwanted updates may be received for a short period of time.
- **timeout** (*Optional[int]*) – Timeout in seconds for long polling. Defaults to 0, i.e. usual short polling. Be careful not to set this timeout too high, as the connection might be dropped and there's no way of knowing it immediately (so most likely the failure will be detected after the timeout had passed).
- **network\_delay** – Deprecated. Will be honoured as `read_latency` for a while but will be removed in the future.
- **read\_latency** (*Optional[float|int]*) – 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).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

## Notes

The main problem with long polling is that a connection will be dropped and we won't be getting the notification in time for it. For that, we need to use long polling, but not too long as well read latency which is short, but not too short. Long polling improves performance, but if it's too long and the connection is dropped on many cases we won't know the connection dropped before the long polling timeout and the read latency time had passed. If you experience connection timeouts, you should tune these settings.

**Returns**list[*telegram.Update*]

**Raises***telegram.TelegramError*

**get\_user\_profile\_photos** (\*args, \*\*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** (*Optional[int]*) – Sequential number of the first photo to be returned. By default, all photos are returned.
- **limit** (*Optional[int]*) – Limits the number of photos to be retrieved. Values between 1-100 are accepted. Defaults to 100.
- **timeout** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

### Returns

A list of user profile photos objects is returned.

**Return type**list[*telegram.UserProfilePhotos*]

**Raises***telegram.TelegramError*

**get\_webhook\_info** (timeout=None, \*\*kwargs)

Use this method to get current webhook status.

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

**Parameter****timeout** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

**Returns**class: *telegram.WebhookInfo*

**kickChatMember** (\*args, \*\*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 group or username of the target supergroup (in the format @supergroupusername).
- **user\_id** (*int/str*) – Unique identifier of the target user.
- **timeout** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type**bool

**Raises***telegram.TelegramError*

**kick\_chat\_member** (\*args, \*\*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 group or username of the target supergroup (in the format @supergroupusername).
- **user\_id** (*int*/*str*) – Unique identifier of the target user.
- **timeout** (*Optional*[*int*/*float*]) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type**bool

**Raises***telegram.TelegramError*

**leaveChat** (\*args, \*\*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** (*Optional*[*int*/*float*]) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type**bool

**Raises***telegram.TelegramError*

**leave\_chat** (\*args, \*\*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** (*Optional*[*int*/*float*]) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type**bool

**Raises***telegram.TelegramError*

**sendAudio** (\*args, \*\*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 an .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 backward compatibility, when both fields title and description are empty and mime-type of the sent file is not "audio/mpeg", file is sent as playable voice message. In this case, your audio must be in an .ogg file encoded with OPUS. This will be removed in the future. You need to use sendVoice method instead.

**Parameters**

- **chat\_id** (*int* / *str*) – Unique identifier for the message recipient - Chat id.
- **audio** – Audio file to send. You can either pass a file\_id as String to resend an audio that is already on the Telegram servers, or upload a new audio file using multipart/form-data.
- **duration** (*Optional[int]*) – Duration of sent audio in seconds.
- **performer** (*Optional[str]*) – Performer of sent audio.
- **title** (*Optional[str]*) – Title of sent audio.
- **caption** (*Optional[str]*) – Audio caption
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int/float]*) – Send file timeout (default: 20 seconds).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**sendChatAction** (\*args, \*\*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 message recipient - Chat id.
- **action** (:class:*\*telegram.ChatAction* | *str*) – Type of action to broadcast. Choose one, depending on what the user is about to receive:
  - *ChatAction.TYPING* for text messages,
  - *ChatAction.UPLOAD\_PHOTO* for photos,
  - *ChatAction.UPLOAD\_VIDEO* for videos,
  - *ChatAction.UPLOAD\_AUDIO* for audio files,
  - *ChatAction.UPLOAD\_DOCUMENT* for general files,
  - *ChatAction.FIND\_LOCATION* for location data.

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

**sendContact** (*\*args, \*\*kwargs*)

Use this method to send phone contacts.

#### Parameters

- **chat\_id** (*int/str*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **phone\_number** (*str*) – Contact’s phone number.
- **first\_name** (*str*) – Contact’s first name.
- **last\_name** (*Optional[str]*) – Contact’s last name.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**sendDocument** (*\*args, \*\*kwargs*)

Use this method to send general files.

#### Parameters

- **chat\_id** (*int/str*) – Unique identifier for the message recipient - Chat id.
- **document** – File to send. You can either pass a file\_id as String to resend a file that is already on the Telegram servers, or upload a new file using multipart/form-data.
- **filename** (*Optional[str]*) – File name that shows in telegram message (it is useful when you send file generated by temp module, for example).
- **caption** (*Optional[str]*) – Document caption (may also be used when resending documents by file\_id), 0-200 characters.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int/float]*) – Send file timeout (default: 20 seconds).

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

**Returns**On success, instance representing the message posted.

**Return type***telegram.Message*

**Raises***telegram.TelegramError*

**sendGame** (*\*args, \*\*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.

#### Keyword Arguments

- disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.

- reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.

- reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int / float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

**Returns**On success, the sent message is returned.

**Return type***telegram.Message*

**Raises***telegram.TelegramError*

**sendLocation** (*\*args, \*\*kwargs*)

Use this method to send point on the map.

#### Parameters

- chat\_id** (*int / str*) – Unique identifier for the message recipient - Chat id.

- latitude** (*float*) – Latitude of location.

- longitude** (*float*) – Longitude of location.

- disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.

- reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.

- reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int / float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

**Returns**On success, instance representing the message posted.

**Return type** `telegram.Message`

**Raises** `telegram.TelegramError`

**sendMessage** (\*args, \*\*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. The current maximum length is 4096 UTF-8 characters.
- **parse\_mode** (*Optional*[*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.
- **disable\_web\_page\_preview** (*Optional*[*bool*]) – Disables link previews for links in this message.
- **disable\_notification** (*Optional*[*bool*]) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional*[*int*]) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional*[`telegram.ReplyMarkup`]) – 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** (*Optional*[*int* / *float*]) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, the sent message is returned.

**Return type** `telegram.Message`

**Raises** `telegram.TelegramError`

**sendPhoto** (\*args, \*\*kwargs)

Use this method to send photos.

#### Parameters

- **chat\_id** (*int* / *str*) – Unique identifier for the message recipient - Chat id.
- **photo** – Photo to send. You can either pass a `file_id` as String to resend a photo that is already on the Telegram servers, or upload a new photo using multipart/form-data.
- **caption** (*Optional*[*str*]) – Photo caption (may also be used when resending photos by `file_id`).
- **disable\_notification** (*Optional*[*bool*]) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional*[*int*]) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional*[`telegram.ReplyMarkup`]) – 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** (*Optional*[*int* / *float*]) – Send file timeout (default: 20 seconds).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns**On success, instance representing the message posted.

**Return type***telegram.Message*

**Raises***telegram.TelegramError*

**sendSticker** (\*args, \*\*kwargs)

Use this method to send .webp stickers.

#### Parameters

- **chat\_id** (*int / str*) – Unique identifier for the message recipient - Chat id.
- **sticker** – Sticker to send. You can either pass a `file_id` as String to resend a sticker that is already on the Telegram servers, or upload a new sticker using multipart/form-data.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int / float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns**On success, instance representing the message posted.

**Return type***telegram.Message*

**Raises***telegram.TelegramError*

**sendVenue** (\*args, \*\*kwargs)

Use this method to send information about a venue.

#### Parameters

- **chat\_id** (*int / str*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **latitude** (*float*) – Latitude of the venue.
- **longitude** (*float*) – Longitude of the venue.
- **title** (*str*) – Name of the venue.
- **address** (*str*) – Address of the venue.
- **foursquare\_id** (*Optional[str]*) – Foursquare identifier of the venue.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**sendVideo** (*\*args, \*\*kwargs*)

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

#### Parameters

- **chat\_id** (*int|str*) – Unique identifier for the message recipient - Chat id.
- **video** – Video to send. You can either pass a file\_id as String to resend a video that is already on the Telegram servers, or upload a new video file using multipart/form-data.
- **duration** (*Optional[int]*) – Duration of sent video in seconds.
- **caption** (*Optional[str]*) – Video caption (may also be used when resending videos by file\_id).
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – Send file timeout (default: 20 seconds).

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**sendVoice** (*\*args, \*\*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). 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.

#### Parameters

- **chat\_id** (*int|str*) – Unique identifier for the message recipient - Chat id.
- **voice** – Audio file to send. You can either pass a file\_id as String to resend an audio that is already on the Telegram servers, or upload a new audio file using multipart/form-data.
- **duration** (*Optional[int]*) – Duration of sent audio in seconds.
- **caption** (*Optional[str]*) – Voice caption
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.

- **reply\_markup** (Optional[*telegram.ReplyMarkup*]) – 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** (Optional[*int|float*]) – Send file timeout (default: 20 seconds).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**send\_audio** (\*args, \*\*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 an .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 backward compatibility, when both fields title and description are empty and mime-type of the sent file is not “audio/mpeg”, file is sent as playable voice message. In this case, your audio must be in an .ogg file encoded with OPUS. This will be removed in the future. You need to use `sendVoice` method instead.

#### Parameters

- **chat\_id** (*int|str*) – Unique identifier for the message recipient - Chat id.
- **audio** – Audio file to send. You can either pass a `file_id` as String to resend an audio that is already on the Telegram servers, or upload a new audio file using multipart/form-data.
- **duration** (Optional[*int*]) – Duration of sent audio in seconds.
- **performer** (Optional[*str*]) – Performer of sent audio.
- **title** (Optional[*str*]) – Title of sent audio.
- **caption** (Optional[*str*]) – Audio caption
- **disable\_notification** (Optional[*bool*]) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (Optional[*int*]) – If the message is a reply, ID of the original message.
- **reply\_markup** (Optional[*telegram.ReplyMarkup*]) – 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** (Optional[*int|float*]) – Send file timeout (default: 20 seconds).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**send\_chat\_action** (\*args, \*\*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 message recipient - Chat id.

- **action** (:class:`telegram.ChatAction`|str) – Type of action to broadcast. Choose one, depending on what the user is about to receive:
  - ChatAction.TYPING for text messages,
  - ChatAction.UPLOAD\_PHOTO for photos,
  - ChatAction.UPLOAD\_VIDEO for videos,
  - ChatAction.UPLOAD\_AUDIO for audio files,
  - ChatAction.UPLOAD\_DOCUMENT for general files,
  - ChatAction.FIND\_LOCATION for location data.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**send\_contact** (\*args, \*\*kwargs)

Use this method to send phone contacts.

#### Parameters

- **chat\_id** (*int|str*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **phone\_number** (*str*) – Contact’s phone number.
- **first\_name** (*str*) – Contact’s first name.
- **last\_name** (*Optional[str]*) – Contact’s last name.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**send\_document** (\*args, \*\*kwargs)

Use this method to send general files.

#### Parameters

- **chat\_id** (*int|str*) – Unique identifier for the message recipient - Chat id.
- **document** – File to send. You can either pass a file\_id as String to resend a file that is already on the Telegram servers, or upload a new file using multipart/form-data.
- **filename** (*Optional[str]*) – File name that shows in telegram message (it is useful when you send file generated by temp module, for example).

- **caption** (*Optional[str]*) – Document caption (may also be used when resending documents by `file_id`), 0-200 characters.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int/float]*) – Send file timeout (default: 20 seconds).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**send\_game** (*\*args, \*\*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.

#### Keyword Arguments

- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

**Returns** On success, the sent message is returned.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**send\_location** (*\*args, \*\*kwargs*)

Use this method to send point on the map.

#### Parameters

- **chat\_id** (*int/str*) – Unique identifier for the message recipient - Chat id.
- **latitude** (*float*) – Latitude of location.
- **longitude** (*float*) – Longitude of location.

- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**send\_message** (*\*args, \*\*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. The current maximum length is 4096 UTF-8 characters.
- **parse\_mode** (*Optional[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.
- **disable\_web\_page\_preview** (*Optional[bool]*) – Disables link previews for links in this message.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, the sent message is returned.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**send\_photo** (*\*args, \*\*kwargs*)

Use this method to send photos.

#### Parameters

- **chat\_id** (*int|str*) – Unique identifier for the message recipient - Chat id.

- **photo** – Photo to send. You can either pass a `file_id` as `String` to resend a photo that is already on the Telegram servers, or upload a new photo using `multipart/form-data`.
- **caption** (*Optional[str]*) – Photo caption (may also be used when resending photos by `file_id`).
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – Send file timeout (default: 20 seconds).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

**Raises** `telegram.TelegramError`

**send\_sticker** (*\*args, \*\*kwargs*)

Use this method to send `.webp` stickers.

#### Parameters

- **chat\_id** (*int|str*) – Unique identifier for the message recipient - Chat id.
- **sticker** – Sticker to send. You can either pass a `file_id` as `String` to resend a sticker that is already on the Telegram servers, or upload a new sticker using `multipart/form-data`.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

**Raises** `telegram.TelegramError`

**send\_venue** (*\*args, \*\*kwargs*)

Use this method to send information about a venue.

#### Parameters

- **chat\_id** (*int|str*) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **latitude** (*float*) – Latitude of the venue.

- **longitude** (*float*) – Longitude of the venue.
- **title** (*str*) – Name of the venue.
- **address** (*str*) – Address of the venue.
- **foursquare\_id** (*Optional[str]*) – Foursquare identifier of the venue.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**send\_video** (*\*args, \*\*kwargs*)

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

**Parameters**

- **chat\_id** (*int|str*) – Unique identifier for the message recipient - Chat id.
- **video** – Video to send. You can either pass a *file\_id* as *String* to resend a video that is already on the Telegram servers, or upload a new video file using *multipart/form-data*.
- **duration** (*Optional[int]*) – Duration of sent video in seconds.
- **caption** (*Optional[str]*) – Video caption (may also be used when resending videos by *file\_id*).
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – Send file timeout (default: 20 seconds).

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**send\_voice** (*\*args, \*\*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*). 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.

### Parameters

- **chat\_id** (*int* / *str*) – Unique identifier for the message recipient - Chat id.
- **voice** – Audio file to send. You can either pass a `file_id` as String to resend an audio that is already on the Telegram servers, or upload a new audio file using `multipart/form-data`.
- **duration** (*Optional[int]*) – Duration of sent audio in seconds.
- **caption** (*Optional[str]*) – Voice caption
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – Send file timeout (default: 20 seconds).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

**Raises** `telegram.TelegramError`

**setGameScore** (*user\_id, score, chat\_id=None, message\_id=None, inline\_message\_id=None, edit\_message=None, force=None, disable\_edit\_message=None, timeout=None, \*\*kwargs*)

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

### Parameters

- **user\_id** (*int*) – User identifier.
- **score** (*int*) – New score, must be non-negative.
- **chat\_id** (*Optional[int|str]*) – Required if `inline_message_id` is not specified. Unique identifier for the target chat (or username of the target channel in the format `@channelusername`)
- **message\_id** (*Optional[int]*) – Required if `inline_message_id` is not specified. Identifier of the sent message.
- **inline\_message\_id** (*Optional[str]*) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.
- **force** (*Optional[bool]*) – Pass True, if the high score is allowed to decrease. This can be useful when fixing mistakes or banning cheaters.
- **disable\_edit\_message** (*Optional[bool]*) – Pass True, if the game message should not be automatically edited to include the current scoreboard.
- **edit\_message** (*Optional[bool]*) – Deprecated. Has the opposite logic for `disable_edit_message`.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

### Returns

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



**Return type** `telegram.Message` or `True`

**setWebhook** (*\*args*, *\*\*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.

#### Parameters

- **url** (*str*) – HTTPS url to send updates to. Use an empty string to remove webhook integration.
- **certificate** (*file*) – Upload your public key certificate so that the root certificate in use can be checked.
- **max\_connections** (*Optional[int]*) – 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** (*Optional[list[str]]*) – 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 `setWebhook`, so unwanted updates may be received for a short period of time.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type** `bool`

**Raises** `telegram.TelegramError`

**set\_game\_score** (*user\_id*, *score*, *chat\_id=None*, *message\_id=None*, *inline\_message\_id=None*, *edit\_message=None*, *force=None*, *disable\_edit\_message=None*, *timeout=None*, *\*\*kwargs*)

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

#### Parameters

- **user\_id** (*int*) – User identifier.
- **score** (*int*) – New score, must be non-negative.
- **chat\_id** (*Optional[int|str]*) – Required if *inline\_message\_id* is not specified. Unique identifier for the target chat (or username of the target channel in the format `@channelusername`)
- **message\_id** (*Optional[int]*) – Required if *inline\_message\_id* is not specified. Identifier of the sent message.
- **inline\_message\_id** (*Optional[str]*) – Required if *chat\_id* and *message\_id* are not specified. Identifier of the inline message.
- **force** (*Optional[bool]*) – Pass `True`, if the high score is allowed to decrease. This can be useful when fixing mistakes or banning cheaters.
- **disable\_edit\_message** (*Optional[bool]*) – Pass `True`, if the game message should not be automatically edited to include the current scoreboard.

- **edit\_message** (*Optional[bool]*) – Deprecated. Has the opposite logic for *disable\_edit\_message*.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

#### Returns

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

Return type *telegram.Message* or True

**set\_webhook** (*\*args, \*\*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.

#### Parameters

- **url** (*str*) – HTTPS url to send updates to. Use an empty string to remove webhook integration.
- **certificate** (*file*) – Upload your public key certificate so that the root certificate in use can be checked.
- **max\_connections** (*Optional[int]*) – 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** (*Optional[list[str]]*) – 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 *setWebhook*, so unwanted updates may be received for a short period of time.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

Returns On success, *True* is returned.

Return type *bool*

Raises *telegram.TelegramError*

**unbanChatMember** (*\*args, \*\*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 group or username of the target supergroup (in the format @supergroupusername).
- **user\_id** (*int|str*) – Unique identifier of the target user.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

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

**Return type**bool

**Raises***telegram.TelegramError*

**unban\_chat\_member** (*\*args, \*\*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 group or username of the target supergroup (in the format @supergroupusername).

•**user\_id** (*int|str*) – Unique identifier of the target user.

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

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

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

**Return type**bool

**Raises***telegram.TelegramError*

## telegram.callbackgame module

This module contains an object that represents a Telegram CallbackGame.

**class** `telegram.callbackgame.CallbackGame`

Bases: *telegram.base.TelegramObject*

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

## telegram.callbackquery module

This module contains an object that represents a Telegram CallbackQuery

**class** `telegram.callbackquery.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 a Telegram CallbackQuery.

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

Shortcut for `bot.answerCallbackQuery(update.callback_query.id, *args, **kwargs)`

**static de\_json** (*data, bot*)

**Parameters**

•**data** (*dict*) –

•**bot** (*telegram.Bot*) –

**Returns**

**Return type***telegram.CallbackQuery*

```
edit_message_caption (*args, **kwargs)
    Shortcut for either bot.editMessageCaption(chat_id=update.
    callback_query.message.chat_id, message_id=update.
    callback_query.message.message_id, *args, **kwargs) or bot.
    editMessageCaption(inline_message_id=update.callback_query.
    inline_message_id, *args, **kwargs)

edit_message_reply_markup (*args, **kwargs)
    Shortcut for either bot.editMessageReplyMarkup(chat_id=update.
    callback_query.message.chat_id, message_id=update.
    callback_query.message.message_id, *args, **kwargs) or bot.
    editMessageReplyMarkup(inline_message_id=update.callback_query.
    inline_message_id, *args, **kwargs)

edit_message_text (*args, **kwargs)
    Shortcut for either bot.editMessageText(chat_id=update.
    callback_query.message.chat_id, message_id=update.
    callback_query.message.message_id, *args, **kwargs) or bot.
    editMessageText(inline_message_id=update.callback_query.
    inline_message_id, *args, **kwargs)

to_dict ()

    Returns
    Return typedict
```

## telegram.chat module

This module contains an object that represents a Telegram Chat.

```
class telegram.chat.Chat (id, type, title=None, username=None, first_name=None,
    last_name=None, all_members_are_administrators=None, bot=None,
    **kwargs)
```

Bases: *telegram.base.TelegramObject*

This object represents a Telegram Chat.

**id**

*int*

**type**

*str* – Can be ‘private’, ‘group’, ‘supergroup’ or ‘channel’

**title**

*str* – Title, for channels and group chats

**username**

*str* – Username, for private chats and channels if available

**first\_name**

*str* – First name of the other party in a private chat

**last\_name**

*str* – Last name of the other party in a private chat

**all\_members\_are\_administrators**

*bool* – True if group has ‘All Members Are Administrators’

### Parameters

- **id** (*int*) –
- **type** (*str*) –
- **title** (*Optional[str]*) –

- username** (*Optional[str]*) –
- first\_name** (*Optional[str]*) –
- last\_name** (*Optional[str]*) –
- bot** (*Optional[Bot]*) – The Bot to use for instance methods
- kwargs** (*dict*) – Arbitrary keyword arguments.

**CHANNEL** = 'channel'

**GROUP** = 'group'

**PRIVATE** = 'private'

**SUPERGROUP** = 'supergroup'

static **de\_json** (*data, bot*)

#### Parameters

- data** (*dict*) –
- bot** (*telegram.Bot*) –

#### Returns

Return type *telegram.Chat*

**get\_administrators** (*\*args, \*\*kwargs*)

Shortcut for `bot.getChatAdministrators(update.message.chat.id, *args, **kwargs)`

**get\_member** (*\*args, \*\*kwargs*)

Shortcut for `bot.getChatMember(update.message.chat.id, *args, **kwargs)`

**get\_members\_count** (*\*args, \*\*kwargs*)

Shortcut for `bot.getChatMembersCount(update.message.chat.id, *args, **kwargs)`

**kick\_member** (*\*args, \*\*kwargs*)

Shortcut for `bot.kickChatMember(update.message.chat.id, *args, **kwargs)`

**leave** (*\*args, \*\*kwargs*)

Shortcut for `bot.leaveChat(update.message.chat.id, *args, **kwargs)`

**send\_action** (*\*args, \*\*kwargs*)

Shortcut for `bot.sendChatAction(update.message.chat.id, *args, **kwargs)`

**unban\_member** (*\*args, \*\*kwargs*)

Shortcut for `bot.unbanChatMember(update.message.chat.id, *args, **kwargs)`

## telegram.chataction module

This module contains an object that represents a Telegram ChatAction.

**class** `telegram.chataction.ChatAction`

Bases: `object`

This object represents a Telegram ChatAction.

**FIND\_LOCATION** = 'find\_location'

**RECORD\_AUDIO** = 'record\_audio'

**RECORD\_VIDEO** = 'record\_video'

**TYPING** = 'typing'

```
UPLOAD_AUDIO = 'upload_audio'
UPLOAD_DOCUMENT = 'upload_document'
UPLOAD_PHOTO = 'upload_photo'
UPLOAD_VIDEO = 'upload_video'
```

## telegram.chatmember module

This module contains an object that represents a Telegram ChatMember.

```
class telegram.chatmember.ChatMember (user, status, **kwargs)
    Bases: telegram.base.TelegramObject
```

This object represents a Telegram ChatMember.

### user

*telegram.User* – Information about the user.

### status

*str* – The member's status in the chat. Can be 'creator', 'administrator', 'member', 'left' or 'kicked'.

### Parameters

- **user** (*telegram.User*) –
- **status** (*str*) –
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

```
ADMINISTRATOR = 'administrator'
```

```
CREATOR = 'creator'
```

```
KICKED = 'kicked'
```

```
LEFT = 'left'
```

```
MEMBER = 'member'
```

```
static de_json (data, bot)
```

### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

### Returns

Return type *telegram.ChatMember*

## telegram.choseninlineresult module

This module contains an object that represents a Telegram ChosenInlineResult

```
class telegram.choseninlineresult.ChosenInlineResult (result_id, from_user,
                                                       query, location=None, in-
                                                       line_message_id=None,
                                                       **kwargs)
```

Bases: *telegram.base.TelegramObject*

This object represents a Telegram ChosenInlineResult.

---

### Note:

- In Python *from* is a reserved word, use *from\_user* instead.

**result\_id***str***from\_user***telegram.User***query***str***location***telegram.Location***inline\_message\_id***str***Parameters**

- **result\_id** (*str*) –
- **from\_user** (*telegram.User*) –
- **query** (*str*) –
- **location** (Optional[*telegram.Location*]) –
- **inline\_message\_id** (Optional[*str*]) –
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data*, *bot*)**Parameters**

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

**Returns**Return type *telegram.ChosenInlineResult***to\_dict** ()**Returns**Return type *dict*

## telegram.constants module

Constants in the Telegram network.

`telegram.constants.MAX_MESSAGE_LENGTH`*int* – from <https://core.telegram.org/method/messages.sendMessage#return-errors>`telegram.constants.MAX_CAPTION_LENGTH`*int* – from <https://core.telegram.org/bots/api#sendphoto>The following constants were extracted from the [Telegram Bots FAQ](#).`telegram.constants.SUPPORTED_WEBHOOK_PORTS`*List[int]*`telegram.constants.MAX_FILESIZE_DOWNLOAD`*int* – In bytes.`telegram.constants.MAX_FILESIZE_UPLOAD`*int* – Official limit, the actual limit can be a bit higher.

`telegram.constants.MAX_MESSAGES_PER_SECOND_PER_CHAT`

*int* – 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*

`telegram.constants.MAX_MESSAGES_PER_MINUTE_PER_GROUP`

*int*

`telegram.constants.MAX_INLINE_QUERY_RESULTS`

*int*

The following constant have been found by experimentation:

`telegram.constants.MAX_MESSAGE_ENTITIES`

*int* – Max number of entities that can be in a message. (Beyond this cap telegram will simply ignore further formatting styles)

## telegram.contact module

This module contains an object that represents a Telegram Contact.

**class** `telegram.contact.Contact` (*phone\_number, first\_name, last\_name=None, user\_id=None, \*\*kwargs*)

Bases: `telegram.base.TelegramObject`

This object represents a Telegram Contact.

**phone\_number**

*str*

**first\_name**

*str*

**last\_name**

*str*

**user\_id**

*int*

### Parameters

- **phone\_number** (*str*) –
- **first\_name** (*str*) –
- **last\_name** (*Optional[str]*) –
- **user\_id** (*Optional[int]*) –
- **\*\*kwargs** – Arbitrary keyword arguments.

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

### Parameters

- **data** (*dict*) –
- **bot** (`telegram.Bot`) –

### Returns

Return type `telegram.Contact`



## telegram.document module

This module contains an object that represents a Telegram Document.

```
class telegram.document.Document (file_id, thumb=None, file_name=None, mime_type=None,
                                  file_size=None, **kwargs)
```

Bases: *telegram.base.TelegramObject*

This object represents a Telegram Document.

**file\_id**

*str*

**thumb**

*telegram.PhotoSize*

**file\_name**

*str*

**mime\_type**

*str*

**file\_size**

*int*

### Parameters

- **file\_id** (*str*) –
- **thumb** (Optional[*telegram.PhotoSize*]) –
- **file\_name** (Optional[*str*]) –
- **mime\_type** (Optional[*str*]) –
- **file\_size** (Optional[*int*]) –
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data*, *bot*)

### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

### Returns

Return type *telegram.Document*

## telegram.error module

This module contains an object that represents a Telegram Error.

```
exception telegram.error.BadRequest (message)
```

Bases: *telegram.error.NetworkError*

```
exception telegram.error.ChatMigrated (new_chat_id)
```

Bases: *telegram.error.TelegramError*

```
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*

**exception** `telegram.error.TelegramError` (*message*)

Bases: `exceptions.Exception`

This object represents a Telegram Error.

**exception** `telegram.error.TimedOut`

Bases: `telegram.error.NetworkError`

**exception** `telegram.error.Unauthorized` (*message*)

Bases: `telegram.error.TelegramError`

## telegram.file module

This module contains an object that represents a Telegram File.

**class** `telegram.file.File` (*file\_id*, *bot*, *file\_size=None*, *file\_path=None*, *\*\*kwargs*)

Bases: `telegram.base.TelegramObject`

This object represents a Telegram File.

**file\_id**

*str*

**file\_size**

*str*

**file\_path**

*str*

### Parameters

- **file\_id** (*str*) –
- **bot** (`telegram.Bot`) –
- **file\_size** (*Optional[int]*) –
- **file\_path** (*Optional[str]*) –
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data*, *bot*)

### Parameters

- **data** (*dict*) –
- **bot** (`telegram.Bot`) –

### Returns

Return type `telegram.File`

**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. `custom_path` and `out` are mutually exclusive.

### Parameters

- **custom\_path** (*Optional[str]*) – Custom path.
- **out** (*Optional[object]*) – A file-like object. Must be opened in binary mode, if applicable.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

## telegramforcereply module

This module contains an object that represents a Telegram ForceReply.

**class** `telegramforcereply.ForceReply` (*force\_reply=True, selective=False, \*\*kwargs*)  
 Bases: `telegram.replymarkup.ReplyMarkup`

This object represents a Telegram ForceReply.

**force\_reply**  
*bool*

**selective**  
*bool*

### Parameters

- **force\_reply** (*bool*) –
- **selective** (*Optional[bool]*) –
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

### Parameters

- **data** (*dict*) –
- **bot** (`telegram.Bot`) –

### Returns

**Return type** `telegram.ForceReply`

## telegramgame module

This module contains an object that represents a Telegram Game.

**class** `telegramgame.Game` (*title, description, photo, text=None, text\_entities=None, animation=None, \*\*kwargs*)  
 Bases: `telegram.base.TelegramObject`

This object represents a Telegram Game.

**title**  
*str* – Title of the game.

**description**  
*str* – Description of the game.

**photo**  
 list[`telegram.PhotoSize`] – List of photos that will be displayed in the game message in chats.

### Keyword Arguments

- **text** (*Optional[str]*) – 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 `setGameScore`, or manually edited using `editMessageText`. 0-4096 characters.
- **text\_entities** (*Optional[list[telegram.MessageEntity]]*) – Special entities that appear in text, such as usernames, URLs, bot commands, etc.

- animation** (Optional[*telegram.Animation*]) – Animation that will be displayed in the game message in chats. Upload via BotFather.

**static de\_json** (*data*, *bot*)

**Parameters**

- data** (*dict*) –
- bot** (*telegram.Bot*) –

**Returns**

**Return type***telegram.Game*

**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 *entities* attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See *get\_entity\_text* for more info.

---

**Parameterstypes** (*Optional[list]*) – List of *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\_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.)

---

**Parametersentity** (*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** ()

**Returns**

**Return type**dict

## telegram.gamehighscore module

This module contains an object that represents a Telegram GameHighScore.

**class** telegram.gamehighscore.**GameHighScore** (*position, user, score*)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram GameHighScore.

**position**

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

**user**

*telegram.User* – User object.

**score**

*int* – Score.

**static de\_json** (*data, bot*)

**Parameters**

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

**Returns**

Return type *telegram.Game*

## telegram.inlinekeyboardbutton module

This module contains an object that represents a Telegram InlineKeyboardButton

**class** telegram.inlinekeyboardbutton.**InlineKeyboardButton** (*text, url=None, callback\_data=None, switch\_inline\_query=None, switch\_inline\_query\_current\_chat=None, callback\_game=None, \*\*kwargs*)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram InlineKeyboardButton.

**text**

*str*

**url**

*str*

**callback\_data**

*str*

**switch\_inline\_query**

*str*

**switch\_inline\_query\_current\_chat**

*str*

**callback\_game**

*telegram.CallbackGame*

**Parameters**

- **text** (*str*) – Label text on the button.
- **url** (*Optional[str]*) – HTTP url to be opened when button is pressed.
- **callback\_data** (*Optional[str]*) – Data to be sent in a callback query to the bot when button is pressed, 1-64 bytes.

- switch\_inline\_query** (*Optional[str]*) – 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.
- switch\_inline\_query\_current\_chat** (*Optional[str]*) – 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.
- callback\_game** (*Optional[telegram.CallbackGame]*) – Description of the game that will be launched when the user presses the button.
- \*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

**Parameters**

- data** (*dict*) –
- bot** (*telegram.Bot*) –

**Returns**

**Return type***telegram.InlineKeyboardButton*

**static de\_list** (*data, bot*)

## telegram.inlinekeyboardmarkup module

This module contains an object that represents a Telegram InlineKeyboardMarkup

**class** `telegram.inlinekeyboardmarkup.InlineKeyboardMarkup` (*inline\_keyboard, \*\*kwargs*)

Bases: *telegram.replymarkup.ReplyMarkup*

This object represents a Telegram InlineKeyboardMarkup.

**inline\_keyboard**

List[List[*telegram.InlineKeyboardButton*]]

**Parameters**

- inline\_keyboard** (List[List[*telegram.InlineKeyboardButton*]]) –
- \*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

**Parameters**

- data** (*dict*) –
- bot** (*telegram.Bot*) –

**Returns**

**Return type***telegram.InlineKeyboardMarkup*

**to\_dict** ()

## telegram.inlinequery module

This module contains an object that represents a Telegram InlineQuery

**class** telegram.inlinequery.**InlineQuery** (*id, from\_user, query, offset, location=None, bot=None, \*\*kwargs*)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram InlineQuery.

---

**Note:**

- In Python *from* is a reserved word, use *from\_user* instead.
- 

**id**

*str*

**from\_user**

*telegram.User*

**query**

*str*

**offset**

*str*

**Parameters**

- **id** (*int*) –
- **from\_user** (*telegram.User*) –
- **query** (*str*) –
- **offset** (*str*) –
- **location** (optional [*telegram.Location*]) –
- **bot** (Optional [*Bot*]) – The Bot to use for instance methods
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

Shortcut for `bot.answerInlineQuery(update.inline_query.id, *args, **kwargs)`

**static de\_json** (*data, bot*)

**Parameters**

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

**Returns**

**Return type** *telegram.InlineQuery*

**to\_dict** ()

**Returns**

**Return type** *dict*

## telegram.inlinequeryresult module

This module contains the classes that represent Telegram InlineQueryResult

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

Bases: *telegram.base.TelegramObject*

This object represents a Telegram InlineQueryResult.

**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.

**static de\_json** (*data, bot*)

## telegram.inlinequeryresultarticle module

This module contains the classes that represent Telegram InlineQueryResultArticle

**class** telegram.inlinequeryresultarticle.**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.inlinequeryresult.InlineQueryResult*

This object represents a Telegram InlineQueryResultArticle.

**id**

*str*

**title**

*str*

**input\_message\_content**

*telegram.InputMessageContent*

**reply\_markup**

*telegram.ReplyMarkup*

**url**

*str*

**hide\_url**

*bool*

**description**

*str*

**thumb\_url**

*str*

**thumb\_width**

*int*

**thumb\_height**

*int*



**Deprecated: 4.0** `message_text` (str): Use `InputTextMessageContent` instead.

`parse_mode` (str): Use `InputTextMessageContent` instead.

`disable_web_page_preview` (bool): Use `InputTextMessageContent` instead.

#### Parameters

- **id** (str) – Unique identifier for this result, 1-64 Bytes
- **title** (str) –
- **reply\_markup** (*telegram.ReplyMarkup*) –
- **url** (*Optional[str]*) –
- **hide\_url** (*Optional[bool]*) –
- **description** (*Optional[str]*) –
- **thumb\_url** (*Optional[str]*) –
- **thumb\_width** (*Optional[int]*) –
- **thumb\_height** (*Optional[int]*) –
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

`static de_json` (*data, bot*)

## telegram.inlinequeryresultaudio module

This module contains the classes that represent Telegram `InlineQueryResultAudio`

```
class telegram.inlinequeryresultaudio.InlineQueryResultAudio (id, audio_url,
title, performer=None, audio_duration=None,
caption=None, reply_markup=None,
input_message_content=None,
**kwargs)
```

Bases: *telegram.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.

```
id
    str
audio_url
    str
title
    str
performer
    Optional[str]
audio_duration
    Optional[str]
caption
    Optional[str]
reply_markup
    Optional[telegram.InlineKeyboardMarkup]
```

**input\_message\_content**  
 Optional[telegram.input\_message\_content]

**Deprecated:** 4.0message\_text (str): Use InputTextMessageContent instead.

parse\_mode (str): Use InputTextMessageContent instead.

disable\_web\_page\_preview (bool): Use InputTextMessageContent instead.

**Parameters**

- **audio\_url** (str) –
- **title** (str) –
- **performer** (Optional[str]) –
- **audio\_duration** (Optional[str]) –
- **caption** (Optional[str]) –
- **reply\_markup** (Optional[telegram.InlineKeyboardMarkup]) –
- **input\_message\_content** (Optional[telegram.input\_message\_content]) –
- **\*\*kwargs** (dict) – Arbitrary keyword arguments.

**static de\_json** (data, bot)

## telegram.inlinequeryresultcachedaudio module

This module contains the classes that represent Telegram InlineQueryResultCachedAudio

```
class telegram.inlinequeryresultcachedaudio.InlineQueryResultCachedAudio (id,
                                                                              au-
                                                                              dio_file_id,
                                                                              cap-
                                                                              tion=None,
                                                                              re-
                                                                              ply_markup=None,
                                                                              in-
                                                                              put_message_content=None,
                                                                              **kwargs)
```

Bases: telegram.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.

**id**  
 str

**audio\_file\_id**  
 str

**caption**  
 Optional[str]

**reply\_markup**  
 Optional[telegram.InlineKeyboardMarkup]

**input\_message\_content**  
 Optional[telegram.input\_message\_content]

**Deprecated:** `4.0message_text` (str): Use `InputTextMessageContent` instead.

`parse_mode` (str): Use `InputTextMessageContent` instead.

`disable_web_page_preview` (bool): Use `InputTextMessageContent` instead.

#### Parameters

• `audio_file_id` (str) –

• `caption` (Optional[str]) –

• `reply_markup` (Optional[`telegram.InlineKeyboardMarkup`]) –

• `input_message_content` (Optional[`telegram.InputMessageContent`]) –

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

`static de_json` (data, bot)

## telegram.inlinequeryresultcacheddocument module

This module contains the classes that represent Telegram `InlineQueryResultCachedDocument`

```
class telegram.inlinequeryresultcacheddocument.InlineQueryResultCachedDocument (id,
ti-
tle,
doc-
u-
ment_file_id,
de-
scrip-
tion=None,
cap-
tion=None,
re-
ply_markup=None,
in-
put_message_co-
**kwargs)
```

Bases: `telegram.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. Currently, only pdf-files and zip archives can be sent using this method.

#### `title`

`str` – Title for the result.

#### `document_file_id`

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

#### `description`

Optional[`str`] – Short description of the result.

#### `caption`

Optional[`str`] – Caption of the document to be sent, 0-200 characters.

#### `reply_markup`

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

#### `input_message_content`

Optional[`telegram.InputMessageContent`] – Content of the message to be sent instead of the file.

**Parameters**

- **id** (*str*) –
- **title** (*str*) –
- **document\_file\_id** (*str*) –
- **description** (*Optional[str]*) –
- **caption** (*Optional[str]*) –
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) –
- **input\_message\_content** (*Optional[telegram.InputMessageContent]*) –
- 
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

**telegram.inlinequeryresultcachedgif module**

This module contains the classes that represent Telegram InlineQueryResultCachedGif

```
class telegram.inlinequeryresultcachedgif.InlineQueryResultCachedGif (id,  
gif_file_id,  
ti-  
tle=None,  
cap-  
tion=None,  
re-  
ply_markup=None,  
in-  
put_message_content=None,  
**kwargs)
```

Bases: *telegram.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.

**gif\_file\_id**

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

**title**

*Optional[str]* – Title for the result.

**caption**

*Optional[str]* – Caption of the GIF file to be sent, 0-200 characters.

**reply\_markup**

*Optional[telegram.InlineKeyboardMarkup]* – Inline keyboard attached to the message.

**input\_message\_content**

*Optional[telegram.InputMessageContent]* – Content of the message to be sent instead of the GIF animation.

**Parameters**

- **id** (*str*) –
- **gif\_file\_id** (*str*) –
- **title** (*Optional[str]*) –
- **caption** (*Optional[str]*) –

- **reply\_markup** (Optional[*telegram.InlineKeyboardMarkup*]) –
- **input\_message\_content** (Optional[*telegram.InputMessageContent*]) –
- 
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

## telegram.inlinequeryresultcachedmpeg4gif module

This module contains the classes that represent Telegram InlineQueryResultMpeg4Gif

```
class telegram.inlinequeryresultcachedmpeg4gif.InlineQueryResultCachedMpeg4Gif (id,
                                                                    mpeg4_file_id,
                                                                    ti-
                                                                    tle=None,
                                                                    cap-
                                                                    tion=None,
                                                                    re-
                                                                    ply_markup=None,
                                                                    in-
                                                                    put_message_co
                                                                    **kwargs)
```

Bases: *telegram.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.

**mpeg4\_file\_id**

*str* – A valid file identifier for the MP4 file.

**title**

Optional[*str*] – Title for the result.

**caption**

Optional[*str*] – Caption of the MPEG-4 file to be sent, 0-200 characters.

**reply\_markup**

Optional[*telegram.InlineKeyboardMarkup*] – Inline keyboard attached to the message.

**input\_message\_content**

Optional[*telegram.InputMessageContent*] – Content of the message to be sent instead of the video animation

### Parameters

- **id** (*str*) –
- **mpeg4\_file\_id** (*str*) –
- **title** (Optional[*str*]) –
- **caption** (Optional[*str*]) –
- **reply\_markup** (Optional[*telegram.InlineKeyboardMarkup*]) –
- **input\_message\_content** (Optional[*telegram.InputMessageContent*]) –
- 
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

## telegram.inlinequeryresultcachedphoto module

This module contains the classes that represent Telegram InlineQueryResultPhoto

```
class telegram.inlinequeryresultcachedphoto.InlineQueryResultCachedPhoto (id,  
                                                                    photo_file_id,  
                                                                    ti-  
                                                                    tle=None,  
                                                                    de-  
                                                                    scrip-  
                                                                    tion=None,  
                                                                    cap-  
                                                                    tion=None,  
                                                                    re-  
                                                                    ply_markup=None,  
                                                                    in-  
                                                                    put_message_content=None,  
                                                                    **kwargs)
```

Bases: *telegram.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.

**photo\_file\_id**

*str* – A valid file identifier of the photo.

**title**

*Optional[str]* – Title for the result.

**description**

*Optional[str]* – Short description of the result.

**caption**

*Optional[str]* – Caption of the photo to be sent, 0-200 characters.

**reply\_markup**

*Optional[telegram.InlineKeyboardMarkup]* – Inline keyboard attached to the message.

**input\_message\_content**

*Optional[telegram.InputMessageContent]* – Content of the message to be sent instead of the photo

**Parameters**

- **id** (*str*) –
- **photo\_file\_id** (*str*) –
- **title** (*Optional[str]*) –
- **description** (*Optional[str]*) –
- **caption** (*Optional[str]*) –
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) –
- **input\_message\_content** (*Optional[telegram.InputMessageContent]*) –
- 
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

## telegram.inlinequeryresultcachedsticker module

This module contains the classes that represent Telegram InlineQueryResultCachedSticker

```
class telegram.inlinequeryresultcachedsticker.InlineQueryResultCachedSticker (id,
                                                                    sticker_file_id,
                                                                    re-
                                                                    ply_markup=None,
                                                                    in-
                                                                    put_message_content=None,
                                                                    **kwargs)
```

Bases: `telegram.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.

**sticker\_file\_id**

*str* – A valid file identifier of the sticker.

**reply\_markup**

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

**input\_message\_content**

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

### Parameters

- **id** (*str*) –
- **sticker\_file\_id** (*str*) –
- **reply\_markup** (Optional[`telegram.InlineKeyboardMarkup`]) –
- **input\_message\_content** (Optional[`telegram.InputMessageContent`]) –
- 
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data*, *bot*)

## telegram.inlinequeryresultcachedvideo module

This module contains the classes that represent Telegram InlineQueryResultCachedVideo

```
class telegram.inlinequeryresultcachedvideo.InlineQueryResultCachedVideo (id,
                                                                    video_file_id,
                                                                    ti-
                                                                    tle,
                                                                    de-
                                                                    scrip-
                                                                    tion=None,
                                                                    cap-
                                                                    tion=None,
                                                                    re-
                                                                    ply_markup=None,
                                                                    in-
                                                                    put_message_content=None,
                                                                    **kwargs)
```

Bases: `telegram.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.

**video\_file\_id**

*str* – A valid file identifier for the video file.

**title**

*str* – Title for the result.

**description**

*Optional[str]* – Short description of the result.

**caption**

*Optional[str]* – Caption of the video to be sent, 0-200 characters.

**reply\_markup**

*Optional[telegram.InlineKeyboardMarkup]* – Inline keyboard attached to the message

**input\_message\_content**

*Optional[telegram.InputMessageContent]* – Content of the message to be sent instead of the video

**Parameters**

- **id** (*str*) –
- **video\_file\_id** (*str*) –
- **title** (*str*) –
- **description** (*Optional[str]*) –
- **caption** (*Optional[str]*) –
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) –
- **input\_message\_content** (*Optional[telegram.InputMessageContent]*) –
- 
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

## telegram.inlinequeryresultcachedvoice module

This module contains the classes that represent Telegram InlineQueryResultCachedVoice

```
class telegram.inlinequeryresultcachedvoice.InlineQueryResultCachedVoice (id,
                                                                              voice_file_id,
                                                                              ti-
                                                                              tle,
                                                                              cap-
                                                                              tion=None,
                                                                              re-
                                                                              ply_markup=None,
                                                                              in-
                                                                              put_message_content=None,
                                                                              **kwargs)
```

Bases: *telegram.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.



**voice\_file\_id**  
*str* – A valid file identifier for the voice message.

**title**  
*str* – Voice message title.

**caption**  
*Optional[str]* – Caption, 0-200 characters.

**reply\_markup**  
*Optional[telegram.InlineKeyboardMarkup]* – Inline keyboard attached to the message.

**input\_message\_content**  
*Optional[telegram.InputMessageContent]* – Content of the message to be sent instead of the voice message.

#### Parameters

- **id** (*str*) –
- **voice\_file\_id** (*str*) –
- **title** (*str*) –
- **caption** (*Optional[str]*) –
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) –
- **input\_message\_content** (*Optional[telegram.InputMessageContent]*) –
- 
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

## telegram.inlinequeryresultcontact module

This module contains the classes that represent Telegram InlineQueryResultContact

```
class telegram.inlinequeryresultcontact.InlineQueryResultContact (id,
                                                                    phone_number,
                                                                    first_name,
                                                                    last_name=None,
                                                                    re-
                                                                    ply_markup=None,
                                                                    in-
                                                                    put_message_content=None,
                                                                    thumb_url=None,
                                                                    thumb_width=None,
                                                                    thumb_height=None,
                                                                    **kwargs)
```

Bases: *telegram.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.

**phone\_number**  
*str* – Contact's phone number.

**first\_name**  
*str* – Contact's first name.

**last\_name**  
*Optional[str]* – Contact's last name.

**reply\_markup**

Optional[*telegram.InlineKeyboardMarkup*] – Inline keyboard attached to the message.

**input\_message\_content**

Optional[*telegram.InputMessageContent*] – Content of the message to be sent instead of the contact.

**thumb\_url**

Optional[*str*] – Url of the thumbnail for the result.

**thumb\_width**

Optional[*int*] – Thumbnail width.

**thumb\_height**

Optional[*int*] – Thumbnail height.

**Parameters**

- **id** (*str*) –
- **phone\_number** (*str*) –
- **first\_name** (*str*) –
- **last\_name** (Optional[*str*]) –
- **reply\_markup** (Optional[*telegram.InlineKeyboardMarkup*]) –
- **input\_message\_content** (Optional[*telegram.InputMessageContent*]) –
- **thumb\_url** (Optional[*str*]) – Url of the thumbnail for the result.
- **thumb\_width** (Optional[*int*]) –
- **thumb\_height** (Optional[*int*]) –
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

## telegram.inlinequeryresultdocument module

This module contains the classes that represent Telegram InlineQueryResultDocument

```
class telegram.inlinequeryresultdocument.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.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.

**title**

*str* – Title for the result.

**caption**

*Optional[str]* – Caption of the document to be sent, 0-200 characters.

**document\_url**

*Optional[str]* – A valid URL for the file.

**mime\_type**

*Optional[str]* – Mime type of the content of the file, either “application/pdf” or “application/zip”.

**description**

*Optional[str]* – Short description of the result.

**reply\_markup**

*Optional[telegram.InlineKeyboardMarkup]* – Inline keyboard attached to the message.

**input\_message\_content**

*Optional[telegram.InputMessageContent]* – Content of the message to be sent instead of the file.

**thumb\_url**

*Optional[str]* – URL of the thumbnail (jpeg only) for the file.

**thumb\_width**

*Optional[int]* – Thumbnail width.

**thumb\_height**

*Optional[int]* – Thumbnail height.

**Parameters**

- **id** (*str*) –
- **document\_url** (*str*) –
- **title** (*str*) –
- **mime\_type** (*str*) –
- **caption** (*Optional[str]*) –
- **description** (*Optional[str]*) –
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) –
- **input\_message\_content** (*Optional[telegram.InputMessageContent]*) –
- 
- **thumb\_url** (*Optional[str]*) –
- **thumb\_width** (*Optional[int]*) –
- **thumb\_height** (*Optional[int]*) –
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

## telegram.inlinequeryresultgame module

This module contains the classes that represent Telegram InlineQueryResultGame

```
class telegram.inlinequeryresultgame.InlineQueryResultGame (id,
                                                            game_short_name,
                                                            reply_markup=None,
                                                            **kwargs)

Bases: telegram.inlinequeryresult.InlineQueryResult

static de_json (data, bot)
```

## telegram.inlinequeryresultgif module

This module contains the classes that represent Telegram `InlineQueryResultGif`

```
class telegram.inlinequeryresultgif.InlineQueryResultGif (id, gif_url, thumb_url,
                                                            gif_width=None,
                                                            gif_height=None,
                                                            title=None,          caption=
                                                            tion=None,          re-
                                                            ply_markup=None,   in-
                                                            put_message_content=None,
                                                            **kwargs)
```

Bases: `telegram.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.

### **gif\_url**

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

### **thumb\_url**

*str* – URL of the static thumbnail for the result (jpeg or gif).

### **gif\_width**

*Optional[int]* – Width of the GIF.

### **gif\_height**

*Optional[int]* – Height of the GIF.

### **title**

*Optional[str]* – Title for the result.

### **caption**

*Optional[str]* – Caption of the GIF file to be sent, 0-200 characters.

### **reply\_markup**

*Optional[telegram.InlineKeyboardMarkup]* – Inline keyboard attached to the message.

### **input\_message\_content**

*Optional[telegram.InputMessageContent]* – Content of the message to be sent instead of the GIF animation.

### Parameters

- **id** (*str*) –
- **gif\_url** (*str*) –
- **thumb\_url** (*str*) –
- **gif\_width** (*Optional[int]*) –
- **gif\_height** (*Optional[int]*) –
- **title** (*Optional[str]*) –
- **caption** (*Optional[str]*) –

- **reply\_markup** (Optional[*telegram.InlineKeyboardMarkup*]) –
- **input\_message\_content** (Optional[*telegram.InputMessageContent*]) –
- 
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

## telegram.inlinequeryresultlocation module

This module contains the classes that represent Telegram InlineQueryResultLocation

```
class telegram.inlinequeryresultlocation.InlineQueryResultLocation (id, latitude, longitude, title, reply_markup=None, input_message_content=None, thumb_url=None, thumb_width=None, thumb_height=None, **kwargs)
```

Bases: *telegram.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.

### **latitude**

*float* – Location latitude in degrees.

### **longitude**

*float* – Location longitude in degrees.

### **title**

*str* – Location title.

### **reply\_markup**

Optional[*telegram.InlineKeyboardMarkup*] – Inline keyboard attached to the message.

### **input\_message\_content**

Optional[*telegram.InputMessageContent*] – Content of the message to be sent instead of the location.

### **thumb\_url**

Optional[*str*] – Url of the thumbnail for the result.

### **thumb\_width**

Optional[*int*] – Thumbnail width.

### **thumb\_height**

Optional[*int*] – Thumbnail height.

### Parameters

- **latitude** (*float*) – Location latitude in degrees.
- **longitude** (*float*) – Location longitude in degrees.
- **title** (*str*) – Location title.
- **reply\_markup** (Optional[*telegram.InlineKeyboardMarkup*]) – Inline keyboard attached to the message.

- **input\_message\_content** (Optional[*telegram.InputMessageContent*]) – Content of the message to be sent instead of the location.
- **thumb\_url** (Optional[*str*]) – Url of the thumbnail for the result.
- **thumb\_width** (Optional[*int*]) – Thumbnail width.
- **thumb\_height** (Optional[*int*]) – Thumbnail height.
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

## telegram.inlinequeryresultmpeg4gif module

This module contains the classes that represent Telegram InlineQueryResultMpeg4Gif

```
class telegram.inlinequeryresultmpeg4gif.InlineQueryResultMpeg4Gif (id,  
                                                                mpeg4_url,  
                                                                thumb_url,  
                                                                mpeg4_width=None,  
                                                                mpeg4_height=None,  
                                                                title=None,  
                                                                caption=None,  
                                                                reply_markup=None,  
                                                                input_message_content=None,  
                                                                **kwargs)
```

Bases: *telegram.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.

### **mpeg4\_url**

*str* – A valid URL for the MP4 file. File size must not exceed 1MB.

### **thumb\_url**

*str* – URL of the static thumbnail (jpeg or gif) for the result.

### **mpeg4\_width**

Optional[*int*] – Video width.

### **mpeg4\_height**

Optional[*int*] – Video height.

### **title**

Optional[*str*] – Title for the result.

### **caption**

Optional[*str*] – Caption of the MPEG-4 file to be sent, 0-200 characters.

### **reply\_markup**

Optional[*telegram.InlineKeyboardMarkup*] – Inline keyboard attached to the message.

### **input\_message\_content**

Optional[*telegram.InputMessageContent*] – Content of the message to be sent instead of the video animation.

### Parameters

- **mpeg4\_url** (*str*) – A valid URL for the MP4 file. File size must not exceed 1MB.

- **thumb\_url** (*str*) – URL of the static thumbnail (jpeg or gif) for the result.
- **mpeg4\_width** (*Optional[int]*) – Video width.
- **mpeg4\_height** (*Optional[int]*) – Video height.
- **title** (*Optional[str]*) – Title for the result.
- **caption** (*Optional[str]*) – Caption of the MPEG-4 file to be sent, 0-200 characters.
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) – Inline keyboard attached to the message.
- **input\_message\_content** (*Optional[telegram.InputMessageContent]*) – Content of the message to be sent instead of the video animation.
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

## telegram.inlinequeryresultphoto module

This module contains the classes that represent Telegram InlineQueryResultPhoto

```
class telegram.inlinequeryresultphoto.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.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.

### **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**

*Optional[int]* – Width of the photo.

### **photo\_height**

*Optional[int]* – Height of the photo.

### **title**

*Optional[str]* – Title for the result.

### **description**

*Optional[str]* – Short description of the result.

### **caption**

*Optional[str]* – Caption of the photo to be sent, 0-200 characters.

### **reply\_markup**

*Optional[telegram.InlineKeyboardMarkup]* – Inline keyboard attached to the message.

**input\_message\_content**

Optional[*telegram.InputMessageContent*] – Content of the message to be sent instead of the photo.

**static de\_json** (*data, bot*)

## telegram.inlinequeryresultvenue module

This module contains the classes that represent Telegram InlineQueryResultVenue

**class telegram.inlinequeryresultvenue.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.inlinequeryresult.InlineQueryResult*

**static de\_json** (*data, bot*)

## telegram.inlinequeryresultvideo module

This module contains the classes that represent Telegram InlineQueryResultVideo

**class telegram.inlinequeryresultvideo.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.inlinequeryresult.InlineQueryResult*

**static de\_json** (*data, bot*)

## telegram.inlinequeryresultvoice module

This module contains the classes that represent Telegram InlineQueryResultVoice

**class telegram.inlinequeryresultvoice.InlineQueryResultVoice** (*id, voice\_url, title, voice\_duration=None, caption=None, reply\_markup=None, input\_message\_content=None, \*\*kwargs*)

Bases: *telegram.inlinequeryresult.InlineQueryResult*

**static de\_json** (*data, bot*)



## telegram.inputcontactmessagecontent module

This module contains the classes that represent Telegram InputContactMessageContent

```
class telegram.inputcontactmessagecontent.InputContactMessageContent (phone_number,
                                                                    first_name,
                                                                    last_name=None,
                                                                    **kwargs)

Bases: telegram.inputmessagecontent.InputMessageContent

Base class for Telegram InputContactMessageContent Objects

static de_json (data, bot)
```

## telegram.inputfile module

This module contains an object that represents a Telegram InputFile.

```
class telegram.inputfile.InputFile (data)
    Bases: object

    This object represents a Telegram InputFile.

    content_type
        **Returns* – str*

    headers
        **Returns* – str*

    static is_image (stream)
        Check if the content file is an image by analyzing its headers.

        Parametersstream (str) – A str representing the content of a file.

        ReturnsThe str mimetype of an image.

        Return typestr

    static is_inputfile (data)
        Check if the request is a file request.

        Parametersdata (dict) – A dict of (str, unicode) key/value pairs

        Returnsbool

    to_form ()

        Returns

        Return typestr
```

## telegram.inputlocationmessagecontent module

This module contains the classes that represent Telegram InputLocationMessageContent

```
class telegram.inputlocationmessagecontent.InputLocationMessageContent (latitude,
                                                                    lon-
                                                                    gi-
                                                                    tude,
                                                                    **kwargs)

Bases: telegram.inputmessagecontent.InputMessageContent

Base class for Telegram InputLocationMessageContent Objects

static de_json (data, bot)
```

## telegram.inputmessagecontent module

This module contains the classes that represent Telegram InputMessageContent

**class** telegram.inputmessagecontent.**InputMessageContent**

Bases: *telegram.base.TelegramObject*

Base class for Telegram InputMessageContent Objects

**static de\_json** (*data, bot*)

## telegram.inputtextmessagecontent module

This module contains the classes that represent Telegram InputTextMessageContent

**class** telegram.inputtextmessagecontent.**InputTextMessageContent** (*message\_text, parse\_mode=None, disable\_web\_page\_preview=None, \*\*kwargs*)

Bases: *telegram.inputmessagecontent.InputMessageContent*

Base class for Telegram InputTextMessageContent Objects

**static de\_json** (*data, bot*)

## telegram.inputvenuemessagecontent module

This module contains the classes that represent Telegram InputVenueMessageContent

**class** telegram.inputvenuemessagecontent.**InputVenueMessageContent** (*latitude, longitude, title, address, foursquare\_id=None, \*\*kwargs*)

Bases: *telegram.inputmessagecontent.InputMessageContent*

Base class for Telegram InputVenueMessageContent Objects

**static de\_json** (*data, bot*)

## telegram.keyboardbutton module

This module contains an object that represents a Telegram KeyboardButton.

**class** telegram.keyboardbutton.**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.

### Parameters

• **text** (*str*) –

• **request\_location** (*Optional[bool]*) –

• **request\_contact** (*Optional[bool]*) –

**static de\_json** (*data, bot*)

**static de\_list** (*data, bot*)

## telegram.location module

This module contains an object that represents a Telegram Location.

**class** telegram.location.**Location** (*longitude, latitude, \*\*kwargs*)  
 Bases: *telegram.base.TelegramObject*

This object represents a Telegram Location.

**longitude**  
*float*

**latitude**  
*float*

### Parameters

- **longitude** (*float*) –
- **latitude** (*float*) –

**static de\_json** (*data, bot*)

### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

### Returns

Return type *telegram.Location*

## telegram.message module

This module contains an object that represents a Telegram Message.

**class** telegram.message.**Message** (*message\_id, from\_user, date, chat, forward\_from=None, forward\_from\_chat=None, forward\_date=None, reply\_to\_message=None, edit\_date=None, text=None, entities=None, audio=None, document=None, photo=None, sticker=None, video=None, voice=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, migrate\_to\_chat\_id=None, migrate\_from\_chat\_id=None, channel\_chat\_created=False, pinned\_message=None, forward\_from\_message\_id=None, bot=None, \*\*kwargs*)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram Message.

---

### Note:

- In Python *from* is a reserved word, use *from\_user* instead.
- 

**message\_id**  
*int*

**from\_user**  
*telegram.User*

**date**  
datetime.datetime

**forward\_from**  
*telegram.User*

**forward\_from\_chat**  
*telegram.Chat*

**forward\_from\_message\_id**  
*int*

**forward\_date**  
datetime.datetime

**reply\_to\_message**  
*telegram.Message*

**edit\_date**  
datetime.datetime

**text**  
*str*

**audio**  
*telegram.Audio*

**document**  
*telegram.Document*

**game**  
*telegram.Game*

**photo**  
List[*telegram.PhotoSize*]

**sticker**  
*telegram.Sticker*

**video**  
*telegram.Video*

**voice**  
*telegram.Voice*

**caption**  
*str*

**contact**  
*telegram.Contact*

**location**  
*telegram.Location*

**new\_chat\_member**  
*telegram.User*

**left\_chat\_member**  
*telegram.User*

**new\_chat\_title**  
*str*

**new\_chat\_photo**  
List[*telegram.PhotoSize*]

**delete\_chat\_photo**  
*bool*

**group\_chat\_created**  
*bool*

**supergroup\_chat\_created**  
*bool*

**migrate\_to\_chat\_id**  
*int*

**migrate\_from\_chat\_id**  
*int*

**channel\_chat\_created**  
*bool*

**Deprecated: 4.0** `new_chat_participant` (*telegram.User*): Use `new_chat_member` instead.

`left_chat_participant` (*telegram.User*): Use `left_chat_member` instead.

### Parameters

- **message\_id** (*int*) –
- **from\_user** (*telegram.User*) –
- **date** (*datetime.datetime*) –
- **chat** (*telegram.Chat*) –
- **forward\_from** (Optional[*telegram.User*]) –
- **forward\_from\_chat** (Optional[*telegram.Chat*]) –
- **forward\_from\_message\_id** (Optional[*int*]) –
- **forward\_date** (Optional[*datetime.datetime*]) –
- **reply\_to\_message** (Optional[*telegram.Message*]) –
- **edit\_date** (Optional[*datetime.datetime*]) –
- **text** (Optional[*str*]) –
- **audio** (Optional[*telegram.Audio*]) –
- **document** (Optional[*telegram.Document*]) –
- **game** (Optional[*telegram.Game*]) –
- **photo** (Optional[List[*telegram.PhotoSize*]]) –
- **sticker** (Optional[*telegram.Sticker*]) –
- **video** (Optional[*telegram.Video*]) –
- **voice** (Optional[*telegram.Voice*]) –
- **caption** (Optional[*str*]) –
- **contact** (Optional[*telegram.Contact*]) –
- **location** (Optional[*telegram.Location*]) –
- **new\_chat\_member** (Optional[*telegram.User*]) –
- **left\_chat\_member** (Optional[*telegram.User*]) –
- **new\_chat\_title** (Optional[*str*]) –
- **new\_chat\_photo** (Optional[List[*telegram.PhotoSize*]]) –
- **delete\_chat\_photo** (Optional[*bool*]) –
- **group\_chat\_created** (Optional[*bool*]) –

- `supergroup_chat_created` (*Optional[bool]*) –
- `migrate_to_chat_id` (*Optional[int]*) –
- `migrate_from_chat_id` (*Optional[int]*) –
- `channel_chat_created` (*Optional[bool]*) –
- `bot` (*Optional[Bot]*) – The Bot to use for instance methods

`chat_id`

*int* – Short for `Message.chat.id`

`static de_json` (*data, bot*)

**Parameters**

- `data` (*dict*) –
- `bot` (`telegram.Bot`) –

**Returns**

**Return type** `telegram.Message`

`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.editMessageCaption(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.

---

`edit_reply_markup` (*\*args, \*\*kwargs*)

Shortcut for

```
>>> bot.editReplyMarkup(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.

---

`edit_text` (*\*args, \*\*kwargs*)

Shortcut for

```
>>> bot.editMessageText(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.

---

**forward** (*chat\_id*, *disable\_notification=False*)

Shortcut for

```
>>> bot.forwardMessage(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`

**parse\_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 `entities` attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See `get_entity_text` for more info.

---

**Parameter****types** (*Optional[list]*) – 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.)

---

**Parameter****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`

**reply\_audio** (*\*args, \*\*kwargs*)

Shortcut for `bot.sendAudio(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments****quote** (*Optional[bool]*) – If set to `True`, the audio 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.sendMessage(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments**`quote` (*Optional[bool]*) – If set to `True`, the contact 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.sendDocument(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments**`quote` (*Optional[bool]*) – If set to `True`, the document 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\_location** (\*args, \*\*kwargs)

Shortcut for `bot.sendLocation(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments**`quote` (*Optional[bool]*) – If set to `True`, the location 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\_photo** (\*args, \*\*kwargs)

Shortcut for `bot.sendPhoto(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments**`quote` (*Optional[bool]*) – 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.sendSticker(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments**`quote` (*Optional[bool]*) – If set to `True`, the sticker 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.sendMessage(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments**`quote` (*Optional[bool]*) – 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.sendVenue(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments****quote** (*Optional[bool]*) – If set to `True`, the venue 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.sendVideo(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments****quote** (*Optional[bool]*) – If set to `True`, the video 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.sendVoice(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments****quote** (*Optional[bool]*) – If set to `True`, the voice 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 (uses `parse_entities`).

Use this if you want to retrieve the original string sent by the bot, as opposed to the plain text with corresponding markup entities.

**Returns**`str`

**text\_markdown**

Creates a markdown-formatted string from the markup entities found in the message (uses `parse_entities`).

Use this if you want to retrieve the original string sent by the bot, as opposed to the plain text with corresponding markup entities.

**Returns**`str`

**to\_dict** ()

**Returns**

**Return type**`dict`

## telegram.messageentity module

This module contains an object that represents a Telegram MessageEntity.

**class** `telegram.messageentity.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.

**Parameters**

```
•type (str) –
•offset (int) –
•length (int) –
•url (Optional[str]) –
•user (Optional[telegram.User]) –

ALL_TYPES = ['mention', 'hashtag', 'bot_command', 'url', 'email', 'bold', 'italic', 'code', 'pre', 'text_link', 'text_mention']
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'

static de_json (data, bot)
static de_list (data, bot)
    Parametersdata (list) –
    Returns
    Return typeList<telegram.MessageEntity>
```

## telegram.parsemode module

This module contains an object that represents a Telegram Message Parse Modes.

```
class telegram.parsemode.ParseMode
    Bases: object

    This object represents a Telegram Message Parse Modes.

    HTML = 'HTML'
    MARKDOWN = 'Markdown'
```

## telegram.photosize module

This module contains an object that represents a Telegram PhotoSize.

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

    This object represents a Telegram PhotoSize.

    file_id
        str
    width
        int
```

**height**  
*int*

**file\_size**  
*int*

#### Parameters

- **file\_id** (*str*) –
- **width** (*int*) –
- **height** (*int*) –
- **\*\*kwargs** – Arbitrary keyword arguments.

**Keyword Arguments****file\_size** (*Optional[int]*) –

**static de\_json** (*data, bot*)

#### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

#### Returns

**Return type***telegram.PhotoSize*

**static de\_list** (*data, bot*)

#### Parameters

- **data** (*list*) –
- **bot** (*telegram.Bot*) –

#### Returns

**Return type**List<*telegram.PhotoSize*>

## telegram.replykeyboardremove module

This module contains an object that represents a Telegram ReplyKeyboardRemove.

**class** `telegram.replykeyboardremove.ReplyKeyboardHide`

Bases: `object`

**class** `telegram.replykeyboardremove.ReplyKeyboardRemove` (*selective=False, \*\*kwargs*)

Bases: `telegram.replymarkup.ReplyMarkup`

This object represents a Telegram ReplyKeyboardRemove.

**remove\_keyboard**

*bool* – Always True.

**selective**

*bool*

#### Parameters

- **selective** (*Optional[bool]*) – 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** – Arbitrary keyword arguments.

**static de\_json** (*data*, *bot*)

**Parameters**

- data** (*dict*) –
- bot** (*telegram.Bot*) –

**Return** *telegram.ReplyKeyboardRemove*

## telegram.replykeyboardmarkup module

This module contains an object that represents a Telegram ReplyKeyboardMarkup.

**class** *telegram.replykeyboardmarkup.ReplyKeyboardMarkup* (*keyboard*, *resize\_keyboard=False*, *one\_time\_keyboard=False*, *selective=False*, *\*\*kwargs*)

Bases: *telegram.replymarkup.ReplyMarkup*

This object represents a Telegram ReplyKeyboardMarkup.

**keyboard**

List[List[*telegram.KeyboardButton*]]

**resize\_keyboard**

*bool*

**one\_time\_keyboard**

*bool*

**selective**

*bool*

**Parameters**

- keyboard** (*List[List[str]*) –
- kwargs** – Arbitrary keyword arguments.

**Keyword Arguments**

- resize\_keyboard** (*Optional[bool]*) –
- one\_time\_keyboard** (*Optional[bool]*) –
- selective** (*Optional[bool]*) –

**static de\_json** (*data*, *bot*)

**Parameters**

- data** (*dict*) –
- bot** (*telegram.Bot*) –

**Returns**

**Return type** *telegram.ReplyKeyboardMarkup*

**to\_dict** ()

## telegram.replymarkup module

Base class for Telegram ReplyMarkup Objects.

```
class telegram.replymarkup.ReplyMarkup
    Bases: telegram.base.TelegramObject

    Base class for Telegram ReplyMarkup Objects

    static de_json (data, bot)
```

## telegram.sticker module

This module contains an object that represents a Telegram Sticker.

```
class telegram.sticker.Sticker (file_id, width, height, thumb=None, emoji=None,
                                file_size=None, **kwargs)
    Bases: telegram.base.TelegramObject
```

This object represents a Telegram Sticker.

```
file_id
    str

width
    int

height
    int

thumb
    telegram.PhotoSize

emoji
    str

file_size
    int
```

### Parameters

- **file\_id** (*str*) –
- **width** (*int*) –
- **height** (*int*) –
- **\*\*kwargs** – Arbitrary keyword arguments.

### Keyword Arguments

- **thumb** (Optional[*telegram.PhotoSize*]) –
- **emoji** (Optional[*str*]) –
- **file\_size** (Optional[*int*]) –

```
static de_json (data, bot)
```

### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

### Returns

Return type *telegram.Sticker*

## telegram.update module

This module contains an object that represents a Telegram Update.

```
class telegram.update.Update (update_id,          message=None,          edited_message=None,
                              inline_query=None,   chosen_inline_result=None,
                              callback_query=None, channel_post=None,
                              edited_channel_post=None, **kwargs)
```

Bases: *telegram.base.TelegramObject*

This object represents a Telegram Update.

### **update\_id**

*int* – The update’s unique identifier.

### **message**

*telegram.Message* – New incoming message of any kind - text, photo, sticker, etc.

### **edited\_message**

*telegram.Message* – New version of a message that is known to the bot and was edited

### **inline\_query**

*telegram.InlineQuery* – New incoming inline query.

### **chosen\_inline\_result**

*telegram.ChosenInlineResult* – The result of an inline query that was chosen by a user and sent to their chat partner.

### **callback\_query**

*telegram.CallbackQuery* – New incoming callback query.

### **channel\_post**

Optional[*telegram.Message*] – New incoming channel post of any kind - text, photo, sticker, etc.

### **edited\_channel\_post**

Optional[*telegram.Message*] – New version of a channel post that is known to the bot and was edited.

### Parameters

- **update\_id** (*int*) –
- **message** (Optional[*telegram.Message*]) –
- **edited\_message** (Optional[*telegram.Message*]) –
- **inline\_query** (Optional[*telegram.InlineQuery*]) –
- **chosen\_inline\_result** (Optional[*telegram.ChosenInlineResult*]) –
- **callback\_query** (Optional[*telegram.CallbackQuery*]) –
- **channel\_post** (Optional[*telegram.Message*]) –
- **edited\_channel\_post** (Optional[*telegram.Message*]) –
- **\*\*kwargs** – Arbitrary keyword arguments.

```
static de_json (data, bot)
```

### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

### Returns

Return type *telegram.Update*

**effective\_chat**

A property that contains the `Chat` that this update was sent in, no matter what kind of update this is. Will be `None` for inline queries and chosen inline results.

**effective\_message**

A property that contains the `Message` included in this update, no matter what kind of update this is. Will be `None` for inline queries, chosen inline results and callback queries from inline messages.

**effective\_user**

A property that contains the `User` that sent this update, no matter what kind of update this is. Will be `None` for channel posts.

## telegram.user module

This module contains an object that represents a Telegram User.

```
class telegram.user.User(id, first_name, type=None, last_name=None, username=None,
                          bot=None, **kwargs)
Bases: telegram.base.TelegramObject
```

This object represents a Telegram User.

**id**

*int*

**first\_name**

*str*

**last\_name**

*str*

**username**

*str*

**type**

*str*

**Parameters**

- **id** (*int*) –
- **first\_name** (*str*) –
- **\*\*kwargs** – Arbitrary keyword arguments.

**Keyword Arguments**

- **type** (*Optional[str]*) –
- **last\_name** (*Optional[str]*) –
- **username** (*Optional[str]*) –
- **bot** (*Optional[Bot]*) – The Bot to use for instance methods

```
static de_json(data, bot)
```

**Parameters**

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

**Returns**

Return type *telegram.User*

```
get_profile_photos (*args, **kwargs)
    Shortcut for bot.getUserProfilePhotos(update.message.from_user.id,
    *args, **kwargs)

name
    str
```

## telegram.userprofilephotos module

This module contains an object that represents a Telegram UserProfilePhotos.

```
class telegram.userprofilephotos.UserProfilePhotos (total_count, photos, **kwargs)
    Bases: telegram.base.TelegramObject
```

This object represents a Telegram UserProfilePhotos.

```
total_count
    int

photos
    List[List[telegram.PhotoSize]]
```

### Parameters

- **total\_count** (*int*) –
- **photos** (List[List[*telegram.PhotoSize*]]) –

```
static de_json (data, bot)
```

### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

### Returns

Return type *telegram.UserProfilePhotos*

```
to_dict ()
```

### Returns

Return type *dict*

## telegram.venue module

This module contains an object that represents a Telegram Venue.

```
class telegram.venue.Venue (location, title, address, foursquare_id=None, **kwargs)
    Bases: telegram.base.TelegramObject
```

This object represents a venue.

### Parameters

- **location** (*telegram.Location*) –
- **title** (*str*) –
- **address** (*str*) –
- **foursquare\_id** (*Optional[str]*) –

```
static de_json (data, bot)
```



## telegram.video module

This module contains an object that represents a Telegram Video.

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

Bases: *telegram.base.TelegramObject*

This object represents a Telegram Video.

**file\_id**

*str*

**width**

*int*

**height**

*int*

**duration**

*int*

**thumb**

*telegram.PhotoSize*

**mime\_type**

*str*

**file\_size**

*int*

### Parameters

- **file\_id** (*str*) –
- **width** (*int*) –
- **height** (*int*) –
- **duration** (*int*) –
- **\*\*kwargs** – Arbitrary keyword arguments.

### Keyword Arguments

- **thumb** (Optional[*telegram.PhotoSize*]) –
- **mime\_type** (Optional[*str*]) –
- **file\_size** (Optional[*int*]) –

```
static de_json(data, bot)
```

### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

### Returns

**Return type** *telegram.Video*

## telegram.voice module

This module contains an object that represents a Telegram Voice.

**class** telegram.voice.Voice (*file\_id, duration, mime\_type=None, file\_size=None, \*\*kwargs*)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram Voice.

**file\_id**

*str*

**duration**

*int*

**mime\_type**

*str*

**file\_size**

*int*

#### Parameters

- **file\_id** (*str*) –
- **duration** (*Optional[int]*) –
- **\*\*kwargs** – Arbitrary keyword arguments.

#### Keyword Arguments

- **mime\_type** (*Optional[str]*) –
- **file\_size** (*Optional[int]*) –

**static de\_json** (*data, bot*)

#### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

#### Returns

Return type *telegram.Voice*

## telegram.webhookinfo module

This module contains an object that represents a Telegram WebhookInfo.

**class** telegram.webhookinfo.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.

**url**

*str* – Webhook URL, may be empty if webhook is not set up.

**has\_custom\_certificate**

*bool*

**pending\_update\_count**

*int*

**last\_error\_date**

*int*

**last\_error\_message**

*str*

**Parameters**

- **url** (*str*) – Webhook URL, may be empty if webhook is not set up.
- **has\_custom\_certificate** (*bool*) –
- **pending\_update\_count** (*int*) –
- **last\_error\_date** (*Optional[int]*) –
- **last\_error\_message** (*Optional[str]*) –

**static de\_json** (*data, bot*)

**Parameters**

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

**Returns**

Return type *telegram.WebhookInfo*

## 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 a Telegram Audio.

**file\_id**  
*str*

**duration**  
*int*

**performer**  
*str*

**title**  
*str*

**mime\_type**  
*str*

**file\_size**  
*int*

**Parameters**

- **file\_id** (*str*) –
- **duration** (*int*) –
- **performer** (*Optional[str]*) –
- **title** (*Optional[str]*) –
- **mime\_type** (*Optional[str]*) –
- **file\_size** (*Optional[int]*) –
- **\*\*kwargs** – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

**Parameters**

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

**Returns****Return type***telegram.Audio***class** `telegram.Bot` (*token, base\_url=None, base\_file\_url=None, request=None*)Bases: *telegram.base.TelegramObject*

This object represents a Telegram Bot.

**id***int* – Unique identifier for this bot.**first\_name***str* – Bot's first name.**last\_name***str* – Bot's last name.**username***str* – Bot's username.**name***str* – Bot's @username.**Parameters**

- **token** (*str*) – Bot's unique authentication.
- **base\_url** (*Optional[str]*) – Telegram Bot API service URL.
- **base\_file\_url** (*Optional[str]*) – Telegram Bot API file URL.
- **request** (*Optional[Request]*) – Pre initialized *Request* class.

**answerCallbackQuery** (*\*args, \*\*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.

**Parameters**

- **callback\_query\_id** (*str*) – Unique identifier for the query to be answered.
- **text** (*Optional[str]*) – Text of the notification. If not specified, nothing will be shown to the user.
- **show\_alert** (*Optional[bool]*) – 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** (*Optional[str]*) – URL that will be opened by the user's client.
- **cache\_time** (*Optional[int]*) – The maximum amount of time in seconds that the result of the callback query may be cached client-side. Telegram apps will support caching starting in version 3.14. Defaults to 0.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns**On success, *True* is returned.**Return type***bool***Raises***telegram.TelegramError*

**answerInlineQuery** (\*args, \*\*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** (*Optional[int]*) – The maximum amount of time the result of the inline query may be cached on the server.
- **is\_personal** (*Optional[bool]*) – 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** (*Optional[str]*) – 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** (*Optional[str]*) – 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** (*Optional[str]*) – Parameter for the start message sent to the bot when user presses the switch button.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type** bool

**Raises** *telegram.TelegramError*

**answer\_callback\_query** (\*args, \*\*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.

**Parameters**

- **callback\_query\_id** (*str*) – Unique identifier for the query to be answered.
- **text** (*Optional[str]*) – Text of the notification. If not specified, nothing will be shown to the user.
- **show\_alert** (*Optional[bool]*) – 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** (*Optional[str]*) – URL that will be opened by the user's client.
- **cache\_time** (*Optional[int]*) – The maximum amount of time in seconds that the result of the callback query may be cached client-side. Telegram apps will support caching starting in version 3.14. Defaults to 0.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type** bool

Raises `telegram.TelegramError`

`answer_inline_query` (\*args, \*\*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** (*Optional[int]*) – The maximum amount of time the result of the inline query may be cached on the server.
- **is\_personal** (*Optional[bool]*) – 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** (*Optional[str]*) – 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** (*Optional[str]*) – 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** (*Optional[str]*) – Parameter for the start message sent to the bot when user presses the switch button.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type** bool

Raises `telegram.TelegramError`

`static de_json` (data, bot)

`deleteMessage` (\*args, \*\*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*) – Unique message identifier.

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

**Return type** bool

Raises `telegram.TelegramError`

`deleteWebhook` (\*args, \*\*kwargs)

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

**Parameters**

- **timeout** (*Optional[float]*) – If this value is specified, use it as the definitive timeout (in seconds) for `urlopen()` operations.
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type** `bool`

**Raises** `telegram.TelegramError`

**delete\_message** (*\*args, \*\*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*) – Unique message identifier.

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

**Return type** `bool`

**Raises** `telegram.TelegramError`

**delete\_webhook** (*\*args, \*\*kwargs*)

Use this method to remove webhook integration if you decide to switch back to `getUpdates`. Returns *True* on success. Requires no parameters.

**Parameters**

- **timeout** (*Optional[float]*) – If this value is specified, use it as the definitive timeout (in seconds) for `urlopen()` operations.
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type** `bool`

**Raises** `telegram.TelegramError`

**editMessageCaption** (*\*args, \*\*kwargs*)

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

**Parameters**

- **chat\_id** (*Optional[int | str]*) – Required if `inline_message_id` is not specified. Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **message\_id** (*Optional[int]*) – Required if `inline_message_id` is not specified. Unique identifier of the sent message.
- **inline\_message\_id** (*Optional[str]*) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.
- **caption** (*Optional[str]*) – New caption of the message.
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) – A JSON-serialized object for an inline keyboard.

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

- **\*\*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`

**editMessageReplyMarkup** (*\*args, \*\*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** (*Optional[int|str]*) – Required if `inline_message_id` is not specified. Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).

- **message\_id** (*Optional[int]*) – Required if `inline_message_id` is not specified. Unique identifier of the sent message.

- **inline\_message\_id** (*Optional[str]*) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.

- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) – A JSON-serialized object for an inline keyboard.

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

- **\*\*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`

**editMessageText** (*\*args, \*\*kwargs*)

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

#### Parameters

- **text** (*str*) – New text of the message.

- **chat\_id** (*Optional[int|str]*) – Required if `inline_message_id` is not specified. Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).

- **message\_id** (*Optional[int]*) – Required if `inline_message_id` is not specified. Unique identifier of the sent message.

- **inline\_message\_id** (*Optional[str]*) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.

- **parse\_mode** (:class:'telegram.ParseMode'*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.

- **disable\_web\_page\_preview** (*bool*) – Disables link previews for links in this message.



- **reply\_markup** (Optional[*telegram.ReplyMarkup*]) – 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** (Optional[*int|float*]) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*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_caption(*args, **kwargs)`

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

**Parameters**

- **chat\_id** (Optional[*int|str*]) – Required if `inline_message_id` is not specified. Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **message\_id** (Optional[*int*]) – Required if `inline_message_id` is not specified. Unique identifier of the sent message.
- **inline\_message\_id** (Optional[*str*]) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.
- **caption** (Optional[*str*]) – New caption of the message.
- **reply\_markup** (Optional[*telegram.InlineKeyboardMarkup*]) – A JSON-serialized object for an inline keyboard.
- **timeout** (Optional[*int|float*]) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*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_reply_markup(*args, **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** (Optional[*int|str*]) – Required if `inline_message_id` is not specified. Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **message\_id** (Optional[*int*]) – Required if `inline_message_id` is not specified. Unique identifier of the sent message.
- **inline\_message\_id** (Optional[*str*]) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.

- **reply\_markup** (Optional[*telegram.InlineKeyboardMarkup*]) – A JSON-serialized object for an inline keyboard.
- **timeout** (Optional[*int|float*]) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*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\_text** (\*args, \*\*kwargs)

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

#### Parameters

- **text** (*str*) – New text of the message.
- **chat\_id** (Optional[*int|str*]) – Required if *inline\_message\_id* is not specified. Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **message\_id** (Optional[*int*]) – Required if *inline\_message\_id* is not specified. Unique identifier of the sent message.
- **inline\_message\_id** (Optional[*str*]) – Required if *chat\_id* and *message\_id* are not specified. Identifier of the inline message.
- **parse\_mode** (:class:'*telegram.ParseMode*'*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.
- **disable\_web\_page\_preview** (*bool*) – Disables link previews for links in this message.
- **reply\_markup** (Optional[*telegram.ReplyMarkup*]) – 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** (Optional[*int|float*]) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*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*

**first\_name**

**forward\_message** (\*args, \*\*kwargs)

Use this method to forward messages of any kind.

#### Parameters

- **chat\_id** (*int|str*) – Unique identifier for the message recipient - Chat id.
- **from\_chat\_id** (*int|str*) – Unique identifier for the chat where the original message was sent - Chat id.

- **message\_id** (*int*) – Unique message identifier.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message forwarded.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**forward\_message** (*\*args, \*\*kwargs*)

Use this method to forward messages of any kind.

#### Parameters

- **chat\_id** (*int|str*) – Unique identifier for the message recipient - Chat id.
- **from\_chat\_id** (*int|str*) – Unique identifier for the chat where the original message was sent - Chat id.
- **message\_id** (*int*) – Unique message identifier.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message forwarded.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

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

**Returns** On success, *telegram.Chat* is returned.

**Return type** *telegram.Chat*

**Raises** *telegram.TelegramError*

**getChatAdministrators** (*\*args, \*\*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** (*Optional[int / float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns**A list of chat member objects.

**Return type**`list[telegram.ChatMember]`

**Raises**`telegram.TelegramError`

**getChatMember** (\*args, \*\*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** (*Optional[int / float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns**On success, chat member object is returned.

**Return type**`telegram.ChatMember`

**Raises**`telegram.TelegramError`

**getChatMembersCount** (\*args, \*\*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** (*Optional[int / float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns**On success, an *int* is returned.

**Return type**`int`

**Raises**`telegram.TelegramError`

**getFile** (\*args, \*\*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.

**Parameters**

- **file\_id** (*str*) – File identifier to get info about.
- **timeout** (*Optional[int / float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, a `telegram.File` object is returned.

**Return type** `telegram.File`

**Raises** `telegram.TelegramError`

**getGameHighScores** (*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.

#### Parameters

- **user\_id** (*int*) – User identifier.
- **chat\_id** (*Optional[int|str]*) – Required if *inline\_message\_id* is not specified. Unique identifier for the target chat (or username of the target channel in the format `@channelusername`)
- **message\_id** (*Optional[int]*) – Required if *inline\_message\_id* is not specified. Identifier of the sent message.
- **inline\_message\_id** (*Optional[str]*) – Required if *chat\_id* and *message\_id* are not specified. Identifier of the inline message.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

#### Returns

Scores of the specified user and several of his neighbors in a game.

**Return type** `list[telegram.GameHighScore]`

**getMe** (*\*args*, *\*\*kwargs*)

A simple method for testing your bot's auth token.

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

#### Returns

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

**Return type** `telegram.User`

**Raises** `telegram.TelegramError`

**getUpdates** (*\*args*, *\*\*kwargs*)

Use this method to receive incoming updates using long polling.

#### Parameters

- **offset** (*Optional[int]*) – 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`.
- **limit** (*Optional[int]*) – Limits the number of updates to be retrieved. Values between 1-100 are accepted. Defaults to 100.
- **allowed\_updates** (*Optional[list[str]]*) – 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 `setWebhook`, so unwanted updates may be received for a short period of time.

- **timeout** (*Optional[int]*) – Timeout in seconds for long polling. Defaults to 0, i.e. usual short polling. Be careful not to set this timeout too high, as the connection might be dropped and there's no way of knowing it immediately (so most likely the failure will be detected after the timeout had passed).
- **network\_delay** – Deprecated. Will be honoured as `read_latency` for a while but will be removed in the future.
- **read\_latency** (*Optional[float|int]*) – 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).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

## Notes

The main problem with long polling is that a connection will be dropped and we won't be getting the notification in time for it. For that, we need to use long polling, but not too long as well read latency which is short, but not too short. Long polling improves performance, but if it's too long and the connection is dropped on many cases we won't know the connection dropped before the long polling timeout and the read latency time had passed. If you experience connection timeouts, you should tune these settings.

**Returns**list[*telegram.Update*]

**Raises***telegram.TelegramError*

**getUserProfilePhotos** (*\*args, \*\*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** (*Optional[int]*) – Sequential number of the first photo to be returned. By default, all photos are returned.
- **limit** (*Optional[int]*) – Limits the number of photos to be retrieved. Values between 1-100 are accepted. Defaults to 100.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

### Returns

A list of user profile photos objects is returned.

**Return type**list[*telegram.UserProfilePhotos*]

**Raises***telegram.TelegramError*

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

Use this method to get current webhook status.

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

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

**Returns**class: *telegram.WebhookInfo*

**get\_chat** (\*args, \*\*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** (*Optional* [*int* / *float*]) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, *telegram.Chat* is returned.

**Return type** *telegram.Chat*

**Raises** *telegram.TelegramError*

**get\_chat\_administrators** (\*args, \*\*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** (*Optional* [*int* / *float*]) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** A list of chat member objects.

**Return type** list [*telegram.ChatMember*]

**Raises** *telegram.TelegramError*

**get\_chat\_member** (\*args, \*\*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** (*Optional* [*int* / *float*]) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, chat member object is returned.

**Return type** *telegram.ChatMember*

**Raises** *telegram.TelegramError*

**get\_chat\_members\_count** (\*args, \*\*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** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

**Returns** On success, an *int* is returned.

**Return type** *int*

**Raises** *telegram.TelegramError*

**get\_file** (*\*args, \*\*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.

**Parameters**

- **file\_id** (*str*) – File identifier to get info about.

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

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

**Returns** On success, a *telegram.File* object is returned.

**Return type** *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.

**Parameters**

- **user\_id** (*int*) – User identifier.

- **chat\_id** (*Optional[int|str]*) – Required if *inline\_message\_id* is not specified. Unique identifier for the target chat (or username of the target channel in the format *@channelusername*)

- **message\_id** (*Optional[int]*) – Required if *inline\_message\_id* is not specified. Identifier of the sent message.

- **inline\_message\_id** (*Optional[str]*) – Required if *chat\_id* and *message\_id* are not specified. Identifier of the inline message.

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

**Returns**

Scores of the specified user and several of his neighbors in a game.

**Return type** *list[telegram.GameHighScore]*

**get\_me** (*\*args, \*\*kwargs*)

A simple method for testing your bot's auth token.

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

**Returns**

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



**Return type** `telegram.User`

**Raises** `telegram.TelegramError`

**get\_updates** (*\*args*, *\*\*kwargs*)

Use this method to receive incoming updates using long polling.

#### Parameters

- **offset** (*Optional[int]*) – 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`.
- **limit** (*Optional[int]*) – Limits the number of updates to be retrieved. Values between 1-100 are accepted. Defaults to 100.
- **allowed\_updates** (*Optional[list[str]]*) – 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 `setWebhook`, so unwanted updates may be received for a short period of time.
- **timeout** (*Optional[int]*) – Timeout in seconds for long polling. Defaults to 0, i.e. usual short polling. Be careful not to set this timeout too high, as the connection might be dropped and there's no way of knowing it immediately (so most likely the failure will be detected after the timeout had passed).
- **network\_delay** – Deprecated. Will be honoured as *read\_latency* for a while but will be removed in the future.
- **read\_latency** (*Optional[float|int]*) – 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).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

#### Notes

The main problem with long polling is that a connection will be dropped and we won't be getting the notification in time for it. For that, we need to use long polling, but not too long as well read latency which is short, but not too short. Long polling improves performance, but if it's too long and the connection is dropped on many cases we won't know the connection dropped before the long polling timeout and the read latency time had passed. If you experience connection timeouts, you should tune these settings.

**Returns** `list[telegram.Update]`

**Raises** `telegram.TelegramError`

**get\_user\_profile\_photos** (*\*args*, *\*\*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** (*Optional[int]*) – Sequential number of the first photo to be returned. By default, all photos are returned.
- **limit** (*Optional[int]*) – Limits the number of photos to be retrieved. Values between 1-100 are accepted. Defaults to 100.

- **timeout** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

**Returns**

A list of user profile photos objects is returned.

**Return type**list[*telegram.UserProfilePhotos*]

**Raises***telegram.TelegramError*

**get\_webhook\_info** (*timeout=None, \*\*kwargs*)

Use this method to get current webhook status.

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

- **parameter timeout** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

**Returns**class: *telegram.WebhookInfo*

**id**

**info** (*func*)

**kickChatMember** (*\*args, \*\*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 group or username of the target supergroup (in the format @supergroupusername).

- **user\_id** (*int/str*) – Unique identifier of the target user.

- **timeout** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

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

**Return type**bool

**Raises***telegram.TelegramError*

**kick\_chat\_member** (*\*args, \*\*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 group or username of the target supergroup (in the format @supergroupusername).

- **user\_id** (*int/str*) – Unique identifier of the target user.

- **timeout** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

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

**Return type**bool

**Raises***telegram.TelegramError*

**last\_name**

**leaveChat** (\*args, \*\*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** (*Optional[int / float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type**bool

**Raises***telegram.TelegramError*

**leave\_chat** (\*args, \*\*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** (*Optional[int / float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type**bool

**Raises***telegram.TelegramError*

**log** (*func*)

**message** (*func*)

**name**

**request**

**sendAudio** (\*args, \*\*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 an .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 backward compatibility, when both fields title and description are empty and mime-type of the sent file is not “audio/mpeg”, file is sent as playable voice message. In this case, your audio must be in an .ogg file encoded with OPUS. This will be removed in the future. You need to use sendVoice method instead.

**Parameters**

- **chat\_id** (*int / str*) – Unique identifier for the message recipient - Chat id.

- **audio** – Audio file to send. You can either pass a `file_id` as `String` to resend an audio that is already on the Telegram servers, or upload a new audio file using `multipart/form-data`.
- **duration** (*Optional[int]*) – Duration of sent audio in seconds.
- **performer** (*Optional[str]*) – Performer of sent audio.
- **title** (*Optional[str]*) – Title of sent audio.
- **caption** (*Optional[str]*) – Audio caption
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – Send file timeout (default: 20 seconds).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

**Raises** `telegram.TelegramError`

**sendChatAction** (*\*args, \*\*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 message recipient - Chat id.
- **action** (:class:`*telegram.ChatAction`|*str*) – Type of action to broadcast. Choose one, depending on what the user is about to receive:
  - `ChatAction.TYPING` for text messages,
  - `ChatAction.UPLOAD_PHOTO` for photos,
  - `ChatAction.UPLOAD_VIDEO` for videos,
  - `ChatAction.UPLOAD_AUDIO` for audio files,
  - `ChatAction.UPLOAD_DOCUMENT` for general files,
  - `ChatAction.FIND_LOCATION` for location data.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**sendContact** (*\*args, \*\*kwargs*)

Use this method to send phone contacts.

#### Parameters

- **chat\_id** (*int|str*) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **phone\_number** (*str*) – Contact's phone number.

- **first\_name** (*str*) – Contact’s first name.
- **last\_name** (*Optional[str]*) – Contact’s last name.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**sendDocument** (*\*args, \*\*kwargs*)

Use this method to send general files.

#### Parameters

- **chat\_id** (*int|str*) – Unique identifier for the message recipient - Chat id.
- **document** – File to send. You can either pass a file\_id as String to resend a file that is already on the Telegram servers, or upload a new file using multipart/form-data.
- **filename** (*Optional[str]*) – File name that shows in telegram message (it is useful when you send file generated by temp module, for example).
- **caption** (*Optional[str]*) – Document caption (may also be used when resending documents by file\_id), 0-200 characters.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – Send file timeout (default: 20 seconds).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**sendGame** (*\*args, \*\*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.

#### Keyword Arguments

- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

**Returns** On success, the sent message is returned.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**sendLocation** (*\*args, \*\*kwargs*)

Use this method to send point on the map.

#### Parameters

- **chat\_id** (*int/str*) – Unique identifier for the message recipient - Chat id.
- **latitude** (*float*) – Latitude of location.
- **longitude** (*float*) – Longitude of location.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**sendMessage** (*\*args, \*\*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. The current maximum length is 4096 UTF-8 characters.

- **parse\_mode** (*Optional[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.
- **disable\_web\_page\_preview** (*Optional[bool]*) – Disables link previews for links in this message.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, the sent message is returned.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**sendPhoto** (*\*args, \*\*kwargs*)

Use this method to send photos.

#### Parameters

- **chat\_id** (*int|str*) – Unique identifier for the message recipient - Chat id.
- **photo** – Photo to send. You can either pass a `file_id` as String to resend a photo that is already on the Telegram servers, or upload a new photo using multipart/form-data.
- **caption** (*Optional[str]*) – Photo caption (may also be used when resending photos by `file_id`).
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – Send file timeout (default: 20 seconds).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**sendSticker** (*\*args, \*\*kwargs*)

Use this method to send .webp stickers.

#### Parameters

- **chat\_id** (*int|str*) – Unique identifier for the message recipient - Chat id.

- **sticker** – Sticker to send. You can either pass a `file_id` as `String` to resend a sticker that is already on the Telegram servers, or upload a new sticker using `multipart/form-data`.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

**Raises** `telegram.TelegramError`

**sendVenue** (*\*args, \*\*kwargs*)

Use this method to send information about a venue.

#### Parameters

- **chat\_id** (*int|str*) – Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).
- **latitude** (*float*) – Latitude of the venue.
- **longitude** (*float*) – Longitude of the venue.
- **title** (*str*) – Name of the venue.
- **address** (*str*) – Address of the venue.
- **foursquare\_id** (*Optional[str]*) – Foursquare identifier of the venue.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

**Raises** `telegram.TelegramError`

**sendVideo** (*\*args, \*\*kwargs*)

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



**Parameters**

- **chat\_id** (*int* / *str*) – Unique identifier for the message recipient - Chat id.
- **video** – Video to send. You can either pass a `file_id` as String to resend a video that is already on the Telegram servers, or upload a new video file using multipart/form-data.
- **duration** (*Optional[int]*) – Duration of sent video in seconds.
- **caption** (*Optional[str]*) – Video caption (may also be used when resending videos by `file_id`).
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int/float]*) – Send file timeout (default: 20 seconds).

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

**Raises** `telegram.TelegramError`

**sendVoice** (*\*args, \*\*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). 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.

**Parameters**

- **chat\_id** (*int* / *str*) – Unique identifier for the message recipient - Chat id.
- **voice** – Audio file to send. You can either pass a `file_id` as String to resend an audio that is already on the Telegram servers, or upload a new audio file using multipart/form-data.
- **duration** (*Optional[int]*) – Duration of sent audio in seconds.
- **caption** (*Optional[str]*) – Voice caption
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int/float]*) – Send file timeout (default: 20 seconds).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

**Raises** `telegram.TelegramError`

**send\_audio** (\*args, \*\*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 an .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 backward compatibility, when both fields title and description are empty and mime-type of the sent file is not "audio/mpeg", file is sent as playable voice message. In this case, your audio must be in an .ogg file encoded with OPUS. This will be removed in the future. You need to use sendVoice method instead.

#### Parameters

- **chat\_id** (*int* / *str*) – Unique identifier for the message recipient - Chat id.
- **audio** – Audio file to send. You can either pass a file\_id as String to resend an audio that is already on the Telegram servers, or upload a new audio file using multipart/form-data.
- **duration** (*Optional[int]*) – Duration of sent audio in seconds.
- **performer** (*Optional[str]*) – Performer of sent audio.
- **title** (*Optional[str]*) – Title of sent audio.
- **caption** (*Optional[str]*) – Audio caption
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int/float]*) – Send file timeout (default: 20 seconds).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**send\_chat\_action** (\*args, \*\*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 message recipient - Chat id.
- **action** (:class:*\*telegram.ChatAction* | *str*) – Type of action to broadcast. Choose one, depending on what the user is about to receive:
  - *ChatAction.TYPING* for text messages,
  - *ChatAction.UPLOAD\_PHOTO* for photos,
  - *ChatAction.UPLOAD\_VIDEO* for videos,
  - *ChatAction.UPLOAD\_AUDIO* for audio files,
  - *ChatAction.UPLOAD\_DOCUMENT* for general files,
  - *ChatAction.FIND\_LOCATION* for location data.

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

**send\_contact** (*\*args, \*\*kwargs*)

Use this method to send phone contacts.

#### Parameters

- **chat\_id** (*int/str*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **phone\_number** (*str*) – Contact’s phone number.
- **first\_name** (*str*) – Contact’s first name.
- **last\_name** (*Optional[str]*) – Contact’s last name.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int/float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**send\_document** (*\*args, \*\*kwargs*)

Use this method to send general files.

#### Parameters

- **chat\_id** (*int/str*) – Unique identifier for the message recipient - Chat id.
- **document** – File to send. You can either pass a file\_id as String to resend a file that is already on the Telegram servers, or upload a new file using multipart/form-data.
- **filename** (*Optional[str]*) – File name that shows in telegram message (it is useful when you send file generated by temp module, for example).
- **caption** (*Optional[str]*) – Document caption (may also be used when resending documents by file\_id), 0-200 characters.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int/float]*) – Send file timeout (default: 20 seconds).

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

**Returns**On success, instance representing the message posted.

**Return type***telegram.Message*

**Raises***telegram.TelegramError*

**send\_game** (*\*args, \*\*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.

#### Keyword Arguments

- disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.

- reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.

- reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int / float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

**Returns**On success, the sent message is returned.

**Return type***telegram.Message*

**Raises***telegram.TelegramError*

**send\_location** (*\*args, \*\*kwargs*)

Use this method to send point on the map.

#### Parameters

- chat\_id** (*int / str*) – Unique identifier for the message recipient - Chat id.

- latitude** (*float*) – Latitude of location.

- longitude** (*float*) – Longitude of location.

- disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.

- reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.

- reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int / float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

**Returns**On success, instance representing the message posted.

**Return type** `telegram.Message`

**Raises** `telegram.TelegramError`

**send\_message** (*\*args*, *\*\*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. The current maximum length is 4096 UTF-8 characters.
- **parse\_mode** (*Optional*[*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.
- **disable\_web\_page\_preview** (*Optional*[*bool*]) – Disables link previews for links in this message.
- **disable\_notification** (*Optional*[*bool*]) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional*[*int*]) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional*[`telegram.ReplyMarkup`]) – 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** (*Optional*[*int* / *float*]) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, the sent message is returned.

**Return type** `telegram.Message`

**Raises** `telegram.TelegramError`

**send\_photo** (*\*args*, *\*\*kwargs*)

Use this method to send photos.

#### Parameters

- **chat\_id** (*int* / *str*) – Unique identifier for the message recipient - Chat id.
- **photo** – Photo to send. You can either pass a `file_id` as String to resend a photo that is already on the Telegram servers, or upload a new photo using multipart/form-data.
- **caption** (*Optional*[*str*]) – Photo caption (may also be used when resending photos by `file_id`).
- **disable\_notification** (*Optional*[*bool*]) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional*[*int*]) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional*[`telegram.ReplyMarkup`]) – 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** (*Optional*[*int* / *float*]) – Send file timeout (default: 20 seconds).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns**On success, instance representing the message posted.

**Return type***telegram.Message*

**Raises***telegram.TelegramError*

**send\_sticker** (\*args, \*\*kwargs)

Use this method to send .webp stickers.

#### Parameters

- **chat\_id** (*int / str*) – Unique identifier for the message recipient - Chat id.
- **sticker** – Sticker to send. You can either pass a *file\_id* as String to resend a sticker that is already on the Telegram servers, or upload a new sticker using multipart/form-data.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int / float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns**On success, instance representing the message posted.

**Return type***telegram.Message*

**Raises***telegram.TelegramError*

**send\_venue** (\*args, \*\*kwargs)

Use this method to send information about a venue.

#### Parameters

- **chat\_id** (*int / str*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **latitude** (*float*) – Latitude of the venue.
- **longitude** (*float*) – Longitude of the venue.
- **title** (*str*) – Name of the venue.
- **address** (*str*) – Address of the venue.
- **foursquare\_id** (*Optional[str]*) – Foursquare identifier of the venue.
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**send\_video** (*\*args, \*\*kwargs*)

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

#### Parameters

- **chat\_id** (*int|str*) – Unique identifier for the message recipient - Chat id.
- **video** – Video to send. You can either pass a file\_id as String to resend a video that is already on the Telegram servers, or upload a new video file using multipart/form-data.
- **duration** (*Optional[int]*) – Duration of sent video in seconds.
- **caption** (*Optional[str]*) – Video caption (may also be used when resending videos by file\_id).
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.
- **reply\_markup** (*Optional[telegram.ReplyMarkup]*) – 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** (*Optional[int|float]*) – Send file timeout (default: 20 seconds).

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**send\_voice** (*\*args, \*\*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). 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.

#### Parameters

- **chat\_id** (*int|str*) – Unique identifier for the message recipient - Chat id.
- **voice** – Audio file to send. You can either pass a file\_id as String to resend an audio that is already on the Telegram servers, or upload a new audio file using multipart/form-data.
- **duration** (*Optional[int]*) – Duration of sent audio in seconds.
- **caption** (*Optional[str]*) – Voice caption
- **disable\_notification** (*Optional[bool]*) – Sends the message silently. iOS users will not receive a notification, Android users will receive a notification with no sound.
- **reply\_to\_message\_id** (*Optional[int]*) – If the message is a reply, ID of the original message.

- **reply\_markup** (Optional[*telegram.ReplyMarkup*]) – 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** (Optional[*int|float*]) – Send file timeout (default: 20 seconds).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**Returns** On success, instance representing the message posted.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**setGameScore** (*user\_id, score, chat\_id=None, message\_id=None, inline\_message\_id=None, edit\_message=None, force=None, disable\_edit\_message=None, timeout=None, \*\*kwargs*)

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

#### Parameters

- **user\_id** (*int*) – User identifier.
- **score** (*int*) – New score, must be non-negative.
- **chat\_id** (Optional[*int|str*]) – Required if *inline\_message\_id* is not specified. Unique identifier for the target chat (or username of the target channel in the format *@channelusername*)
- **message\_id** (Optional[*int*]) – Required if *inline\_message\_id* is not specified. Identifier of the sent message.
- **inline\_message\_id** (Optional[*str*]) – Required if *chat\_id* and *message\_id* are not specified. Identifier of the inline message.
- **force** (Optional[*bool*]) – Pass True, if the high score is allowed to decrease. This can be useful when fixing mistakes or banning cheaters.
- **disable\_edit\_message** (Optional[*bool*]) – Pass True, if the game message should not be automatically edited to include the current scoreboard.
- **edit\_message** (Optional[*bool*]) – Deprecated. Has the opposite logic for *disable\_edit\_message*.
- **timeout** (Optional[*int|float*]) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

#### Returns

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

**Return type** *telegram.Message* or True

**setWebhook** (*\*args, \*\*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.

#### Parameters

- **url** (*str*) – HTTPS url to send updates to. Use an empty string to remove webhook integration.
- **certificate** (*file*) – Upload your public key certificate so that the root certificate in use can be checked.
- **max\_connections** (Optional[*int*]) – 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** (*Optional[list[str]]*) – 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 `setWebhook`, so unwanted updates may be received for a short period of time.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type** `bool`

**Raises** `telegram.TelegramError`

**set\_game\_score** (*user\_id, score, chat\_id=None, message\_id=None, inline\_message\_id=None, edit\_message=None, force=None, disable\_edit\_message=None, timeout=None, \*\*kwargs*)

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

**Parameters**

- **user\_id** (*int*) – User identifier.
- **score** (*int*) – New score, must be non-negative.
- **chat\_id** (*Optional[int|str]*) – Required if `inline_message_id` is not specified. Unique identifier for the target chat (or username of the target channel in the format `@channelusername`)
- **message\_id** (*Optional[int]*) – Required if `inline_message_id` is not specified. Identifier of the sent message.
- **inline\_message\_id** (*Optional[str]*) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.
- **force** (*Optional[bool]*) – Pass `True`, if the high score is allowed to decrease. This can be useful when fixing mistakes or banning cheaters.
- **disable\_edit\_message** (*Optional[bool]*) – Pass `True`, if the game message should not be automatically edited to include the current scoreboard.
- **edit\_message** (*Optional[bool]*) – Deprecated. Has the opposite logic for `disable_edit_message`.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

**Returns**

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

**Return type** `telegram.Message` or `True`

**set\_webhook** (*\*args, \*\*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.

### Parameters

- **url** (*str*) – HTTPS url to send updates to. Use an empty string to remove webhook integration.
- **certificate** (*file*) – Upload your public key certificate so that the root certificate in use can be checked.
- **max\_connections** (*Optional[int]*) – 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** (*Optional[list[str]]*) – 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 `setWebhook`, so unwanted updates may be received for a short period of time.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type** bool

**Raises** `telegram.TelegramError`

`to_dict()`

`unbanChatMember(*args, **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 group or username of the target supergroup (in the format @supergroupusername).
- **user\_id** (*int|str*) – Unique identifier of the target user.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type** bool

**Raises** `telegram.TelegramError`

`unban_chat_member(*args, **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 group or username of the target supergroup (in the format @supergroupusername).

- **user\_id** (*int / str*) – Unique identifier of the target user.
- **timeout** (*Optional[int / float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

**Return type** *bool*

**Raises** *telegram.TelegramError*

**username**

**class** `telegram.Chat` (*id, type, title=None, username=None, first\_name=None, last\_name=None, all\_members\_are\_administrators=None, bot=None, \*\*kwargs*)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram Chat.

**id**

*int*

**type**

*str* – Can be ‘private’, ‘group’, ‘supergroup’ or ‘channel’

**title**

*str* – Title, for channels and group chats

**username**

*str* – Username, for private chats and channels if available

**first\_name**

*str* – First name of the other party in a private chat

**last\_name**

*str* – Last name of the other party in a private chat

**all\_members\_are\_administrators**

*bool* – True if group has ‘All Members Are Administrators’

**Parameters**

- **id** (*int*) –
- **type** (*str*) –
- **title** (*Optional[str]*) –
- **username** (*Optional[str]*) –
- **first\_name** (*Optional[str]*) –
- **last\_name** (*Optional[str]*) –
- **bot** (*Optional[Bot]*) – The Bot to use for instance methods
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**CHANNEL** = ‘channel’

**GROUP** = ‘group’

**PRIVATE** = ‘private’

**SUPERGROUP** = ‘supergroup’

**static de\_json** (*data, bot*)

**Parameters**

- data** (*dict*) –
- bot** (*telegram.Bot*) –

### Returns

Return type *telegram.Chat*

**get\_administrators** (*\*args, \*\*kwargs*)

Shortcut for `bot.getChatAdministrators(update.message.chat.id, *args, **kwargs)`

**get\_member** (*\*args, \*\*kwargs*)

Shortcut for `bot.getChatMember(update.message.chat.id, *args, **kwargs)`

**get\_members\_count** (*\*args, \*\*kwargs*)

Shortcut for `bot.getChatMembersCount(update.message.chat.id, *args, **kwargs)`

**kick\_member** (*\*args, \*\*kwargs*)

Shortcut for `bot.kickChatMember(update.message.chat.id, *args, **kwargs)`

**leave** (*\*args, \*\*kwargs*)

Shortcut for `bot.leaveChat(update.message.chat.id, *args, **kwargs)`

**send\_action** (*\*args, \*\*kwargs*)

Shortcut for `bot.sendChatAction(update.message.chat.id, *args, **kwargs)`

**unban\_member** (*\*args, \*\*kwargs*)

Shortcut for `bot.unbanChatMember(update.message.chat.id, *args, **kwargs)`

**class telegram.ChatMember** (*user, status, \*\*kwargs*)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram ChatMember.

### user

*telegram.User* – Information about the user.

### status

*str* – The member's status in the chat. Can be 'creator', 'administrator', 'member', 'left' or 'kicked'.

### Parameters

- user** (*telegram.User*) –
- status** (*str*) –
- \*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**ADMINISTRATOR** = 'administrator'

**CREATOR** = 'creator'

**KICKED** = 'kicked'

**LEFT** = 'left'

**MEMBER** = 'member'

**static de\_json** (*data, bot*)

### Parameters

- data** (*dict*) –
- bot** (*telegram.Bot*) –

### Returns

Return type *telegram.ChatMember*

**class** telegram.**ChatAction**

Bases: object

This object represents a Telegram ChatAction.

**FIND\_LOCATION** = 'find\_location'

**RECORD\_AUDIO** = 'record\_audio'

**RECORD\_VIDEO** = 'record\_video'

**TYPING** = 'typing'

**UPLOAD\_AUDIO** = 'upload\_audio'

**UPLOAD\_DOCUMENT** = 'upload\_document'

**UPLOAD\_PHOTO** = 'upload\_photo'

**UPLOAD\_VIDEO** = 'upload\_video'

**class** telegram.**ChosenInlineResult** (*result\_id, from\_user, query, location=None, inline\_message\_id=None, \*\*kwargs*)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram ChosenInlineResult.

---

#### Note:

- In Python *from* is a reserved word, use *from\_user* instead.
- 

**result\_id**

*str*

**from\_user**

*telegram.User*

**query**

*str*

**location**

*telegram.Location*

**inline\_message\_id**

*str*

#### Parameters

- **result\_id** (*str*) –
- **from\_user** (*telegram.User*) –
- **query** (*str*) –
- **location** (Optional[*telegram.Location*]) –
- **inline\_message\_id** (Optional[*str*]) –
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static** **de\_json** (*data, bot*)

#### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

#### Returns

Return type *telegram.ChosenInlineResult*

`to_dict()`

**Returns**

**Return typedict**

**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 a Telegram CallbackQuery.

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

Shortcut for `bot.answerCallbackQuery(update.callback_query.id, *args, **kwargs)`

**static de\_json** (*data, bot*)

**Parameters**

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

**Returns**

**Return type** *telegram.CallbackQuery*

**edit\_message\_caption** (*\*args, \*\*kwargs*)

Shortcut for either `bot.editMessageCaption(chat_id=update.callback_query.message.chat_id, message_id=update.callback_query.message.message_id, *args, **kwargs)` or `bot.editMessageCaption(inline_message_id=update.callback_query.inline_message_id, *args, **kwargs)`

**edit\_message\_reply\_markup** (*\*args, \*\*kwargs*)

Shortcut for either `bot.editMessageReplyMarkup(chat_id=update.callback_query.message.chat_id, message_id=update.callback_query.message.message_id, *args, **kwargs)` or `bot.editMessageReplyMarkup(inline_message_id=update.callback_query.inline_message_id, *args, **kwargs)`

**edit\_message\_text** (*\*args, \*\*kwargs*)

Shortcut for either `bot.editMessageText(chat_id=update.callback_query.message.chat_id, message_id=update.callback_query.message.message_id, *args, **kwargs)` or `bot.editMessageText(inline_message_id=update.callback_query.inline_message_id, *args, **kwargs)`

`to_dict()`

**Returns**

**Return typedict**

**class** telegram.**Contact** (*phone\_number, first\_name, last\_name=None, user\_id=None, \*\*kwargs*)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram Contact.

**phone\_number**

*str*

**first\_name**

*str*

**last\_name**

*str*

**user\_id**  
*int*

#### Parameters

- **phone\_number** (*str*) –
- **first\_name** (*str*) –
- **last\_name** (*Optional[str]*) –
- **user\_id** (*Optional[int]*) –
- **\*\*kwargs** – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

#### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

#### Returns

Return type *telegram.Contact*

**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 Telegram Document.

**file\_id**  
*str*

**thumb**  
*telegram.PhotoSize*

**file\_name**  
*str*

**mime\_type**  
*str*

**file\_size**  
*int*

#### Parameters

- **file\_id** (*str*) –
- **thumb** (*Optional[telegram.PhotoSize]*) –
- **file\_name** (*Optional[str]*) –
- **mime\_type** (*Optional[str]*) –
- **file\_size** (*Optional[int]*) –
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

#### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

#### Returns

**Return type***telegram.Document*

**class** telegram.**File** (*file\_id, bot, file\_size=None, file\_path=None, \*\*kwargs*)  
Bases: *telegram.base.TelegramObject*

This object represents a Telegram File.

**file\_id**  
*str*

**file\_size**  
*str*

**file\_path**  
*str*

#### Parameters

- **file\_id** (*str*) –
- **bot** (*telegram.Bot*) –
- **file\_size** (*Optional[int]*) –
- **file\_path** (*Optional[str]*) –
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

#### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

#### Returns

**Return type***telegram.File*

**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. *custom\_path* and *out* are mutually exclusive.

#### Parameters

- **custom\_path** (*Optional[str]*) – Custom path.
- **out** (*Optional[object]*) – A file-like object. Must be opened in binary mode, if applicable.
- **timeout** (*Optional[int|float]*) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

**Raises***ValueError* – If both *custom\_path* and *out* are passed.

**class** telegram.**ForceReply** (*force\_reply=True, selective=False, \*\*kwargs*)  
Bases: *telegram.replymarkup.ReplyMarkup*

This object represents a Telegram ForceReply.

**force\_reply**  
*bool*

**selective**  
*bool*

#### Parameters



- **force\_reply** (*bool*) –
- **selective** (*Optional[bool]*) –
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

#### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

#### Returns

Return type *telegram.ForceReply*

```
class telegram.InlineKeyboardButton (text, url=None, callback_data=None,
                                     switch_inline_query=None,
                                     switch_inline_query_current_chat=None, call-
                                     back_game=None, **kwargs)
```

Bases: *telegram.base.TelegramObject*

This object represents a Telegram InlineKeyboardButton.

#### **text**

*str*

#### **url**

*str*

#### **callback\_data**

*str*

#### **switch\_inline\_query**

*str*

#### **switch\_inline\_query\_current\_chat**

*str*

#### **callback\_game**

*telegram.CallbackGame*

#### Parameters

- **text** (*str*) – Label text on the button.
- **url** (*Optional[str]*) – HTTP url to be opened when button is pressed.
- **callback\_data** (*Optional[str]*) – Data to be sent in a callback query to the bot when button is pressed, 1-64 bytes.
- **switch\_inline\_query** (*Optional[str]*) – 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.
- **switch\_inline\_query\_current\_chat** (*Optional[str]*) – 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.
- **callback\_game** (*Optional[telegram.CallbackGame]*) – Description of the game that will be launched when the user presses the button.
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

#### Parameters

- data** (*dict*) –
- bot** (*telegram.Bot*) –

**Returns****Return type***telegram.InlineKeyboardButton***static de\_list** (*data, bot*)**class** *telegram.InlineKeyboardMarkup* (*inline\_keyboard, \*\*kwargs*)Bases: *telegram.replymarkup.ReplyMarkup*

This object represents a Telegram InlineKeyboardMarkup.

**inline\_keyboard**List[List[*telegram.InlineKeyboardButton*]]**Parameters**

- inline\_keyboard** (List[List[*telegram.InlineKeyboardButton*]]) –
- \*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)**Parameters**

- data** (*dict*) –
- bot** (*telegram.Bot*) –

**Returns****Return type***telegram.InlineKeyboardMarkup***to\_dict** ()**class** *telegram.InlineQuery* (*id, from\_user, query, offset, location=None, bot=None, \*\*kwargs*)Bases: *telegram.base.TelegramObject*

This object represents a Telegram InlineQuery.

---

**Note:**

- In Python *from* is a reserved word, use *from\_user* instead.
- 

**id***str***from\_user***telegram.User***query***str***offset***str***Parameters**

- id** (*int*) –
- from\_user** (*telegram.User*) –
- query** (*str*) –
- offset** (*str*) –
- location** (optional[*telegram.Location*]) –

- bot** (*Optional*[Bot]) – The Bot to use for instance methods
- \*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

Shortcut for `bot.answerInlineQuery(update.inline_query.id, *args, **kwargs)`

**static de\_json** (data, bot)

**Parameters**

- data** (*dict*) –
- bot** (`telegram.Bot`) –

**Returns**

**Return type**`telegram.InlineQuery`

**to\_dict** ()

**Returns**

**Return type**`dict`

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

Bases: `telegram.base.TelegramObject`

This object represents a Telegram InlineQueryResult.

**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.

**static de\_json** (data, bot)

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

Bases: `telegram.base.TelegramObject`

This object represents a Telegram InlineQueryResult.

**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.

**static de\_json** (data, bot)

```
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.inlinequeryresult.InlineQueryResult*

This object represents a Telegram InlineQueryResultArticle.

```
id
    str

title
    str

input_message_content
    telegram.InputMessageContent

reply_markup
    telegram.ReplyMarkup

url
    str

hide_url
    bool

description
    str

thumb_url
    str

thumb_width
    int

thumb_height
    int
```

**Deprecated:** 4.0 `message_text` (str): Use *InputTextMessageContent* instead.

`parse_mode` (str): Use *InputTextMessageContent* instead.

`disable_web_page_preview` (bool): Use *InputTextMessageContent* instead.

#### Parameters

- **id** (*str*) – Unique identifier for this result, 1-64 Bytes
- **title** (*str*) –
- **reply\_markup** (*telegram.ReplyMarkup*) –
- **url** (*Optional[str]*) –
- **hide\_url** (*Optional[bool]*) –
- **description** (*Optional[str]*) –
- **thumb\_url** (*Optional[str]*) –
- **thumb\_width** (*Optional[int]*) –
- **thumb\_height** (*Optional[int]*) –
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

```
static de_json (data, bot)
```

```
class telegram.InlineQueryResultAudio(id, audio_url, title, performer=None, audio_duration=None, caption=None, reply_markup=None, input_message_content=None, **kwargs)
```

Bases: `telegram.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.

```
id
    str

audio_url
    str

title
    str

performer
    Optional[str]

audio_duration
    Optional[str]

caption
    Optional[str]

reply_markup
    Optional[telegram.InlineKeyboardMarkup]

input_message_content
    Optional[telegram.input_message_content]
```

**Deprecated:** 4.0 `message_text` (str): Use `InputTextMessageContent` instead.

`parse_mode` (str): Use `InputTextMessageContent` instead.

`disable_web_page_preview` (bool): Use `InputTextMessageContent` instead.

#### Parameters

- `audio_url` (str) –
- `title` (str) –
- `performer` (Optional[str]) –
- `audio_duration` (Optional[str]) –
- `caption` (Optional[str]) –
- `reply_markup` (Optional[`telegram.InlineKeyboardMarkup`]) –
- `input_message_content` (Optional[`telegram.input_message_content`]) –
- `**kwargs` (dict) – Arbitrary keyword arguments.

```
static de_json(data, bot)
```

```
class telegram.InlineQueryResultCachedAudio(id, audio_file_id, caption=None, reply_markup=None, input_message_content=None, **kwargs)
```

Bases: `telegram.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.

**id**  
*str*

**audio\_file\_id**  
*str*

**caption**  
*Optional[str]*

**reply\_markup**  
*Optional[telegram.InlineKeyboardMarkup]*

**input\_message\_content**  
*Optional[telegram.input\_message\_content]*

**Deprecated:** 4.0 `message_text (str)`: Use *InputTextMessageContent* instead.

`parse_mode (str)`: Use *InputTextMessageContent* instead.

`disable_web_page_preview (bool)`: Use *InputTextMessageContent* instead.

#### Parameters

- **audio\_file\_id** (*str*) –
- **caption** (*Optional[str]*) –
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) –
- **input\_message\_content** (*Optional[telegram.input\_message\_content]*) –
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

**class telegram.InlineQueryResultCachedDocument** (*id, title, document\_file\_id, description=None, caption=None, reply\_markup=None, input\_message\_content=None, \*\*kwargs*)

Bases: *telegram.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. Currently, only pdf-files and zip archives can be sent using this method.

**title**  
*str* – Title for the result.

**document\_file\_id**  
*str* – A valid file identifier for the file.

**description**  
*Optional[str]* – Short description of the result.

**caption**  
*Optional[str]* – Caption of the document to be sent, 0-200 characters.

**reply\_markup**  
*Optional[telegram.InlineKeyboardMarkup]* – Inline keyboard attached to the message.

**input\_message\_content**  
*Optional[telegram.InputMessageContent]* – Content of the message to be sent instead of the file.

#### Parameters

- **id** (*str*) –
- **title** (*str*) –
- **document\_file\_id** (*str*) –
- **description** (*Optional[str]*) –
- **caption** (*Optional[str]*) –
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) –
- **input\_message\_content** (*Optional[telegram.InputMessageContent]*) –
- 
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

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

Bases: *telegram.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.

**gif\_file\_id**

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

**title**

*Optional[str]* – Title for the result.

**caption**

*Optional[str]* – Caption of the GIF file to be sent, 0-200 characters.

**reply\_markup**

*Optional[telegram.InlineKeyboardMarkup]* – Inline keyboard attached to the message.

**input\_message\_content**

*Optional[telegram.InputMessageContent]* – Content of the message to be sent instead of the GIF animation.

#### Parameters

- **id** (*str*) –
- **gif\_file\_id** (*str*) –
- **title** (*Optional[str]*) –
- **caption** (*Optional[str]*) –
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) –
- **input\_message\_content** (*Optional[telegram.InputMessageContent]*) –
- 
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

```
class telegram.InlineQueryResultCachedMpeg4Gif(id, mpeg4_file_id, title=None, caption=None, reply_markup=None, input_message_content=None, **kwargs)
```

Bases: *telegram.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.

**mpeg4\_file\_id**

*str* – A valid file identifier for the MP4 file.

**title**

*Optional[str]* – Title for the result.

**caption**

*Optional[str]* – Caption of the MPEG-4 file to be sent, 0-200 characters.

**reply\_markup**

*Optional[telegram.InlineKeyboardMarkup]* – Inline keyboard attached to the message.

**input\_message\_content**

*Optional[telegram.InputMessageContent]* – Content of the message to be sent instead of the video animation

**Parameters**

- **id** (*str*) –
- **mpeg4\_file\_id** (*str*) –
- **title** (*Optional[str]*) –
- **caption** (*Optional[str]*) –
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) –
- **input\_message\_content** (*Optional[telegram.InputMessageContent]*) –
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

```
class telegram.InlineQueryResultCachedPhoto(id, photo_file_id, title=None, description=None, caption=None, reply_markup=None, input_message_content=None, **kwargs)
```

Bases: `telegram.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.

**photo\_file\_id**

*str* – A valid file identifier of the photo.

**title**

*Optional[str]* – Title for the result.

**description**

*Optional[str]* – Short description of the result.

**caption**

*Optional[str]* – Caption of the photo to be sent, 0-200 characters.

**reply\_markup**

*Optional[telegram.InlineKeyboardMarkup]* – Inline keyboard attached to the message.

**input\_message\_content**

*Optional[telegram.InputMessageContent]* – Content of the message to be sent instead of the photo



**Parameters**

- **id** (*str*) –
- **photo\_file\_id** (*str*) –
- **title** (*Optional[str]*) –
- **description** (*Optional[str]*) –
- **caption** (*Optional[str]*) –
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) –
- **input\_message\_content** (*Optional[telegram.InputMessageContent]*) –
- 
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

```
class telegram.InlineQueryResultCachedSticker(id, sticker_file_id, reply_markup=None,
                                              input_message_content=None,
                                              **kwargs)
```

Bases: *telegram.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.

**sticker\_file\_id**

*str* – A valid file identifier of the sticker.

**reply\_markup**

*Optional[telegram.InlineKeyboardMarkup]* – Inline keyboard attached to the message.

**input\_message\_content**

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

**Parameters**

- **id** (*str*) –
- **sticker\_file\_id** (*str*) –
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) –
- **input\_message\_content** (*Optional[telegram.InputMessageContent]*) –
- 
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

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

Bases: *telegram.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.

**video\_file\_id**

*str* – A valid file identifier for the video file.

**title**

*str* – Title for the result.

**description**

*Optional[str]* – Short description of the result.

**caption**

*Optional[str]* – Caption of the video to be sent, 0-200 characters.

**reply\_markup**

*Optional[telegram.InlineKeyboardMarkup]* – Inline keyboard attached to the message

**input\_message\_content**

*Optional[telegram.InputMessageContent]* – Content of the message to be sent instead of the video

**Parameters**

- **id** (*str*) –
- **video\_file\_id** (*str*) –
- **title** (*str*) –
- **description** (*Optional[str]*) –
- **caption** (*Optional[str]*) –
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) –
- **input\_message\_content** (*Optional[telegram.InputMessageContent]*) –
- 
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

**class telegram.InlineQueryResultCachedVoice** (*id, voice\_file\_id, title, caption=None, reply\_markup=None, input\_message\_content=None, \*\*kwargs*)

Bases: *telegram.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.

**voice\_file\_id**

*str* – A valid file identifier for the voice message.

**title**

*str* – Voice message title.

**caption**

*Optional[str]* – Caption, 0-200 characters.

**reply\_markup**

*Optional[telegram.InlineKeyboardMarkup]* – Inline keyboard attached to the message.

**input\_message\_content**

*Optional[telegram.InputMessageContent]* – Content of the message to be sent instead of the voice message.

**Parameters**

- **id** (*str*) –
- **voice\_file\_id** (*str*) –
- **title** (*str*) –
- **caption** (*Optional[str]*) –
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) –

•**input\_message\_content** (Optional[*telegram.InputMessageContent*])

–

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

**static de\_json** (*data, bot*)

```
class telegram.InlineQueryResultContact (id, phone_number, first_name,
                                         last_name=None, reply_markup=None, in-
                                         put_message_content=None, thumb_url=None,
                                         thumb_width=None, thumb_height=None,
                                         **kwargs)
```

Bases: *telegram.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.

**phone\_number**

*str* – Contact’s phone number.

**first\_name**

*str* – Contact’s first name.

**last\_name**

Optional[*str*] – Contact’s last name.

**reply\_markup**

Optional[*telegram.InlineKeyboardMarkup*] – Inline keyboard attached to the message.

**input\_message\_content**

Optional[*telegram.InputMessageContent*] – Content of the message to be sent instead of the contact.

**thumb\_url**

Optional[*str*] – Url of the thumbnail for the result.

**thumb\_width**

Optional[*int*] – Thumbnail width.

**thumb\_height**

Optional[*int*] – Thumbnail height.

**Parameters**

•**id** (*str*) –

•**phone\_number** (*str*) –

•**first\_name** (*str*) –

•**last\_name** (Optional[*str*]) –

•**reply\_markup** (Optional[*telegram.InlineKeyboardMarkup*]) –

•**input\_message\_content** (Optional[*telegram.InputMessageContent*])

–

•**thumb\_url** (Optional[*str*]) – Url of the thumbnail for the result.

•**thumb\_width** (Optional[*int*]) –

•**thumb\_height** (Optional[*int*]) –

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

**static de\_json** (*data, bot*)

```
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.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.

**title**

*str* – Title for the result.

**caption**

*Optional[str]* – Caption of the document to be sent, 0-200 characters.

**document\_url**

*Optional[str]* – A valid URL for the file.

**mime\_type**

*Optional[str]* – Mime type of the content of the file, either “application/pdf” or “application/zip”.

**description**

*Optional[str]* – Short description of the result.

**reply\_markup**

*Optional[telegram.InlineKeyboardMarkup]* – Inline keyboard attached to the message.

**input\_message\_content**

*Optional[telegram.InputMessageContent]* – Content of the message to be sent instead of the file.

**thumb\_url**

*Optional[str]* – URL of the thumbnail (jpeg only) for the file.

**thumb\_width**

*Optional[int]* – Thumbnail width.

**thumb\_height**

*Optional[int]* – Thumbnail height.

**Parameters**

- **id** (*str*) –
- **document\_url** (*str*) –
- **title** (*str*) –
- **mime\_type** (*str*) –
- **caption** (*Optional[str]*) –
- **description** (*Optional[str]*) –
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) –
- **input\_message\_content** (*Optional[telegram.InputMessageContent]*) –
- 
- **thumb\_url** (*Optional[str]*) –
- **thumb\_width** (*Optional[int]*) –
- **thumb\_height** (*Optional[int]*) –
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

`static de_json (data, bot)`

```
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,
                                     **kwargs)
```

Bases: `telegram.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.

**gif\_url**

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

**thumb\_url**

*str* – URL of the static thumbnail for the result (jpeg or gif).

**gif\_width**

*Optional[int]* – Width of the GIF.

**gif\_height**

*Optional[int]* – Height of the GIF.

**title**

*Optional[str]* – Title for the result.

**caption**

*Optional[str]* – Caption of the GIF file to be sent, 0-200 characters.

**reply\_markup**

*Optional[telegram.InlineKeyboardMarkup]* – Inline keyboard attached to the message.

**input\_message\_content**

*Optional[telegram.InputMessageContent]* – Content of the message to be sent instead of the GIF animation.

#### Parameters

- **id** (*str*) –
- **gif\_url** (*str*) –
- **thumb\_url** (*str*) –
- **gif\_width** (*Optional[int]*) –
- **gif\_height** (*Optional[int]*) –
- **title** (*Optional[str]*) –
- **caption** (*Optional[str]*) –
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) –
- **input\_message\_content** (*Optional[telegram.InputMessageContent]*) –
- 
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

`static de_json (data, bot)`

```
class telegram.InlineQueryResultLocation (id, latitude, longitude, ti-
                                          tle, reply_markup=None, in-
                                          put_message_content=None, thumb_url=None,
                                          thumb_width=None, thumb_height=None,
                                          **kwargs)
```

Bases: `telegram.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.

**latitude**

*float* – Location latitude in degrees.

**longitude**

*float* – Location longitude in degrees.

**title**

*str* – Location title.

**reply\_markup**

Optional[*telegram.InlineKeyboardMarkup*] – Inline keyboard attached to the message.

**input\_message\_content**

Optional[*telegram.InputMessageContent*] – Content of the message to be sent instead of the location.

**thumb\_url**

Optional[*str*] – Url of the thumbnail for the result.

**thumb\_width**

Optional[*int*] – Thumbnail width.

**thumb\_height**

Optional[*int*] – Thumbnail height.

**Parameters**

- **latitude** (*float*) – Location latitude in degrees.
- **longitude** (*float*) – Location longitude in degrees.
- **title** (*str*) – Location title.
- **reply\_markup** (Optional[*telegram.InlineKeyboardMarkup*]) – Inline keyboard attached to the message.
- **input\_message\_content** (Optional[*telegram.InputMessageContent*]) – Content of the message to be sent instead of the location.
- **thumb\_url** (Optional[*str*]) – Url of the thumbnail for the result.
- **thumb\_width** (Optional[*int*]) – Thumbnail width.
- **thumb\_height** (Optional[*int*]) – Thumbnail height.
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

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

```
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, **kwargs)
```

Bases: *telegram.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.

**mpeg4\_url**

*str* – A valid URL for the MP4 file. File size must not exceed 1MB.

**thumb\_url**

*str* – URL of the static thumbnail (jpeg or gif) for the result.

**mpeg4\_width**  
*Optional[int]* – Video width.

**mpeg4\_height**  
*Optional[int]* – Video height.

**title**  
*Optional[str]* – Title for the result.

**caption**  
*Optional[str]* – Caption of the MPEG-4 file to be sent, 0-200 characters.

**reply\_markup**  
*Optional[telegram.InlineKeyboardMarkup]* – Inline keyboard attached to the message.

**input\_message\_content**  
*Optional[telegram.InputMessageContent]* – Content of the message to be sent instead of the video animation.

#### Parameters

- **mpeg4\_url** (*str*) – A valid URL for the MP4 file. File size must not exceed 1MB.
- **thumb\_url** (*str*) – URL of the static thumbnail (jpeg or gif) for the result.
- **mpeg4\_width** (*Optional[int]*) – Video width.
- **mpeg4\_height** (*Optional[int]*) – Video height.
- **title** (*Optional[str]*) – Title for the result.
- **caption** (*Optional[str]*) – Caption of the MPEG-4 file to be sent, 0-200 characters.
- **reply\_markup** (*Optional[telegram.InlineKeyboardMarkup]*) – Inline keyboard attached to the message.
- **input\_message\_content** (*Optional[telegram.InputMessageContent]*) – Content of the message to be sent instead of the video animation.
- **\*\*kwargs** (*dict*) – Arbitrary keyword arguments.

**static de\_json** (*data, bot*)

**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.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.

**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**  
*Optional[int]* – Width of the photo.

**photo\_height**  
*Optional[int]* – Height of the photo.

**title**  
*Optional[str]* – Title for the result.

**description**

Optional[*str*] – Short description of the result.

**caption**

Optional[*str*] – Caption of the photo to be sent, 0-200 characters.

**reply\_markup**

Optional[*telegram.InlineKeyboardMarkup*] – Inline keyboard attached to the message.

**input\_message\_content**

Optional[*telegram.InputMessageContent*] – Content of the message to be sent instead of the photo.

**static de\_json** (*data, bot*)

```
class telegram.InlineQueryResultVenue (id, latitude, longitude, title, address,
                                       foursquare_id=None, reply_markup=None, in-
                                       put_message_content=None, thumb_url=None,
                                       thumb_width=None, thumb_height=None,
                                       **kwargs)
```

Bases: *telegram.inlinequeryresult.InlineQueryResult*

**static de\_json** (*data, bot*)

```
class telegram.InlineQueryResultVideo (id, video_url, mime_type, thumb_url, title, cap-
                                       tion=None, video_width=None, video_height=None,
                                       video_duration=None, description=None, re-
                                       ply_markup=None, input_message_content=None,
                                       **kwargs)
```

Bases: *telegram.inlinequeryresult.InlineQueryResult*

**static de\_json** (*data, bot*)

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

Bases: *telegram.inlinequeryresult.InlineQueryResult*

**static de\_json** (*data, bot*)

```
class telegram.InlineQueryResultGame (id, game_short_name, reply_markup=None,
                                       **kwargs)
```

Bases: *telegram.inlinequeryresult.InlineQueryResult*

**static de\_json** (*data, bot*)

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

Bases: *telegram.inputmessagecontent.InputMessageContent*

Base class for Telegram InputContactMessageContent Objects

**static de\_json** (*data, bot*)

```
class telegram.InputFile (data)
```

Bases: object

This object represents a Telegram InputFile.

**content\_type**

**\*\*Returns\*** – str\*

**headers**

**\*\*Returns\*** – str\*

**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 mimetype of an image.

**Return type**str

**static is\_input\_file** (*data*)

Check if the request is a file request.

**Parameters***data* (*dict*) – A dict of (str, unicode) key/value pairs

**Returns**bool

**to\_form** ()

**Returns**

**Return type**str

**class** telegram. **InputLocationMessageContent** (*latitude, longitude, \*\*kwargs*)

Bases: *telegram.inputmessagecontent.InputMessageContent*

Base class for Telegram InputLocationMessageContent Objects

**static de\_json** (*data, bot*)

**class** telegram. **InputMessageContent**

Bases: *telegram.base.TelegramObject*

Base class for Telegram InputMessageContent Objects

**static de\_json** (*data, bot*)

**class** telegram. **InputTextMessageContent** (*message\_text, parse\_mode=None, disable\_web\_page\_preview=None, \*\*kwargs*)

Bases: *telegram.inputmessagecontent.InputMessageContent*

Base class for Telegram InputTextMessageContent Objects

**static de\_json** (*data, bot*)

**class** telegram. **InputVenueMessageContent** (*latitude, longitude, title, address, foursquare\_id=None, \*\*kwargs*)

Bases: *telegram.inputmessagecontent.InputMessageContent*

Base class for Telegram InputVenueMessageContent Objects

**static de\_json** (*data, bot*)

**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.

**Parameters**

• **text** (*str*) –

• **request\_location** (*Optional[bool]*) –

• **request\_contact** (*Optional[bool]*) –

**static de\_json** (*data, bot*)

**static de\_list** (*data, bot*)

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

Bases: *telegram.base.TelegramObject*

This object represents a Telegram Location.

**longitude**

*float*

**latitude**  
*float*

**Parameters**

- **longitude** (*float*) –
- **latitude** (*float*) –

**static de\_json** (*data, bot*)

**Parameters**

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

**Returns**

**Return type** *telegram.Location*

**class telegram.Message** (*message\_id, from\_user, date, chat, forward\_from=None, forward\_from\_chat=None, forward\_date=None, reply\_to\_message=None, edit\_date=None, text=None, entities=None, audio=None, document=None, photo=None, sticker=None, video=None, voice=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, migrate\_to\_chat\_id=None, migrate\_from\_chat\_id=None, channel\_chat\_created=False, pinned\_message=None, forward\_from\_message\_id=None, bot=None, \*\*kwargs*)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram Message.

---

**Note:**

- In Python *from* is a reserved word, use *from\_user* instead.
- 

**message\_id**  
*int*

**from\_user**  
*telegram.User*

**date**  
*datetime.datetime*

**forward\_from**  
*telegram.User*

**forward\_from\_chat**  
*telegram.Chat*

**forward\_from\_message\_id**  
*int*

**forward\_date**  
*datetime.datetime*

**reply\_to\_message**  
*telegram.Message*

**edit\_date**  
*datetime.datetime*

**text**  
*str*

**audio**  
*telegram.Audio*

**document**  
*telegram.Document*

**game**  
*telegram.Game*

**photo**  
List[*telegram.PhotoSize*]

**sticker**  
*telegram.Sticker*

**video**  
*telegram.Video*

**voice**  
*telegram.Voice*

**caption**  
*str*

**contact**  
*telegram.Contact*

**location**  
*telegram.Location*

**new\_chat\_member**  
*telegram.User*

**left\_chat\_member**  
*telegram.User*

**new\_chat\_title**  
*str*

**new\_chat\_photo**  
List[*telegram.PhotoSize*]

**delete\_chat\_photo**  
*bool*

**group\_chat\_created**  
*bool*

**supergroup\_chat\_created**  
*bool*

**migrate\_to\_chat\_id**  
*int*

**migrate\_from\_chat\_id**  
*int*

**channel\_chat\_created**  
*bool*

**Deprecated: 4.0** `new_chat_participant` (*telegram.User*): Use `new_chat_member` instead.

`left_chat_participant` (*telegram.User*): Use `left_chat_member` instead.

#### Parameters

- **message\_id** (*int*) –
- **from\_user** (*telegram.User*) –
- **date** (*datetime.datetime*) –
- **chat** (*telegram.Chat*) –
- **forward\_from** (*Optional[telegram.User]*) –
- **forward\_from\_chat** (*Optional[telegram.Chat]*) –
- **forward\_from\_message\_id** (*Optional[int]*) –
- **forward\_date** (*Optional[datetime.datetime]*) –
- **reply\_to\_message** (*Optional[telegram.Message]*) –
- **edit\_date** (*Optional[datetime.datetime]*) –
- **text** (*Optional[str]*) –
- **audio** (*Optional[telegram.Audio]*) –
- **document** (*Optional[telegram.Document]*) –
- **game** (*Optional[telegram.Game]*) –
- **photo** (*Optional[List[telegram.PhotoSize]*) –
- **sticker** (*Optional[telegram.Sticker]*) –
- **video** (*Optional[telegram.Video]*) –
- **voice** (*Optional[telegram.Voice]*) –
- **caption** (*Optional[str]*) –
- **contact** (*Optional[telegram.Contact]*) –
- **location** (*Optional[telegram.Location]*) –
- **new\_chat\_member** (*Optional[telegram.User]*) –
- **left\_chat\_member** (*Optional[telegram.User]*) –
- **new\_chat\_title** (*Optional[str]*) –
- **new\_chat\_photo** (*Optional[List[telegram.PhotoSize]*) –
- **delete\_chat\_photo** (*Optional[bool]*) –
- **group\_chat\_created** (*Optional[bool]*) –
- **supergroup\_chat\_created** (*Optional[bool]*) –
- **migrate\_to\_chat\_id** (*Optional[int]*) –
- **migrate\_from\_chat\_id** (*Optional[int]*) –
- **channel\_chat\_created** (*Optional[bool]*) –
- **bot** (*Optional[Bot]*) – The Bot to use for instance methods

**chat\_id**

*int* – Short for `Message.chat.id`

**static de\_json** (*data, bot*)

**Parameters**

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

**Returns**

**Return type***telegram.Message*

**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.editMessageCaption(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.

**edit\_reply\_markup** (\*args, \*\*kwargs)

Shortcut for

```
>>> bot.editReplyMarkup(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.

**edit\_text** (\*args, \*\*kwargs)

Shortcut for

```
>>> bot.editMessageText(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.

**forward** (chat\_id, disable\_notification=False)

Shortcut for

```
>>> bot.forwardMessage(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*

**parse\_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 `entities` attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See `get_entity_text` for more info.

---

**Parameter types** (*Optional[list]*) – 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.)

---

**Parameter 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`

**reply\_audio** (*\*args, \*\*kwargs*)

Shortcut for `bot.sendAudio(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments quote** (*Optional[bool]*) – If set to `True`, the audio 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.sendContact(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments quote** (*Optional[bool]*) – If set to `True`, the contact 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.sendDocument(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments**`quote` (*Optional[bool]*) – If set to `True`, the document 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\_location** (*\*args, \*\*kwargs*)

Shortcut for `bot.sendLocation(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments**`quote` (*Optional[bool]*) – If set to `True`, the location 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\_photo** (*\*args, \*\*kwargs*)

Shortcut for `bot.sendPhoto(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments**`quote` (*Optional[bool]*) – 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.sendSticker(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments**`quote` (*Optional[bool]*) – If set to `True`, the sticker 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.sendMessage(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments**`quote` (*Optional[bool]*) – 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.sendVenue(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments**`quote` (*Optional[bool]*) – If set to `True`, the venue 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.sendVideo(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments**`quote` (*Optional[bool]*) – If set to `True`, the video 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.sendVoice(update.message.chat_id, *args, **kwargs)`

**Keyword Arguments****quote** (*Optional[bool]*) – If set to `True`, the voice 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 (uses `parse_entities`).

Use this if you want to retrieve the original string sent by the bot, as opposed to the plain text with corresponding markup entities.

**Returns**`str`

**text\_markdown**

Creates a markdown-formatted string from the markup entities found in the message (uses `parse_entities`).

Use this if you want to retrieve the original string sent by the bot, as opposed to the plain text with corresponding markup entities.

**Returns**`str`

**to\_dict** ()

**Returns**

**Return type**`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.

**Parameters**

- **type** (*str*) –
- **offset** (*int*) –
- **length** (*int*) –
- **url** (*Optional[str]*) –
- **user** (*Optional[telegram.User]*) –

`ALL_TYPES` = ['mention', 'hashtag', 'bot\_command', 'url', 'email', 'bold', 'italic', 'code', 'pre', 'text\_link', 'text\_mer

`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'
```

```
static de_json (data, bot)
```

```
static de_list (data, bot)
```

```
Parameters data (list) -
```

```
Returns
```

```
Return type List<telegram.MessageEntity>
```

```
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 a Telegram PhotoSize.

```
file_id
```

```
str
```

```
width
```

```
int
```

```
height
```

```
int
```

```
file_size
```

```
int
```

```
Parameters
```

```
•file_id (str) -
```

```
•width (int) -
```

```
•height (int) -
```

```
•**kwargs - Arbitrary keyword arguments.
```

```
Keyword Arguments file_size (Optional[int]) -
```

```
static de_json (data, bot)
```

```
Parameters
```

```
•data (dict) -
```

```
•bot (telegram.Bot) -
```

```
Returns
```

```
Return type telegram.PhotoSize
```

```
static de_list (data, bot)
```

```
Parameters
```

```
•data (list) -
```

```
•bot (telegram.Bot) -
```

### Returns

**Return type** `List<telegram.PhotoSize>`

**class** `telegram.ReplyKeyboardRemove` (*selective=False, \*\*kwargs*)

Bases: `telegram.replymarkup.ReplyMarkup`

This object represents a Telegram ReplyKeyboardRemove.

**remove\_keyboard**

*bool* – Always True.

**selective**

*bool*

### Parameters

• **selective** (*Optional[bool]*) – 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** – Arbitrary keyword arguments.

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

### Parameters

• **data** (*dict*) –

• **bot** (`telegram.Bot`) –

**Return** `telegram.ReplyKeyboardRemove`

**class** `telegram.ReplyKeyboardMarkup` (*keyboard, resize\_keyboard=False,*

*one\_time\_keyboard=False, selective=False, \*\*kwargs*)

Bases: `telegram.replymarkup.ReplyMarkup`

This object represents a Telegram ReplyKeyboardMarkup.

**keyboard**

`List[List[telegram.KeyboardButton]]`

**resize\_keyboard**

*bool*

**one\_time\_keyboard**

*bool*

**selective**

*bool*

### Parameters

• **keyboard** (`List[List[str]]`) –

• **\*\*kwargs** – Arbitrary keyword arguments.

### Keyword Arguments

• **resize\_keyboard** (*Optional[bool]*) –

• **one\_time\_keyboard** (*Optional[bool]*) –

• **selective** (*Optional[bool]*) –

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

**Parameters**

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

**Returns**

**Return type** *telegram.ReplyKeyboardMarkup*

**to\_dict** ()

**class** *telegram.ReplyMarkup*

Bases: *telegram.base.TelegramObject*

Base class for Telegram ReplyMarkup Objects

**static de\_json** (*data, bot*)

**class** *telegram.Sticker* (*file\_id, width, height, thumb=None, emoji=None, file\_size=None, \*\*kwargs*)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram Sticker.

**file\_id**

*str*

**width**

*int*

**height**

*int*

**thumb**

*telegram.PhotoSize*

**emoji**

*str*

**file\_size**

*int*

**Parameters**

- **file\_id** (*str*) –
- **width** (*int*) –
- **height** (*int*) –
- **\*\*kwargs** – Arbitrary keyword arguments.

**Keyword Arguments**

- **thumb** (Optional[*telegram.PhotoSize*]) –
- **emoji** (Optional[*str*]) –
- **file\_size** (Optional[*int*]) –

**static de\_json** (*data, bot*)

**Parameters**

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

**Returns**

**Return type** *telegram.Sticker*

**exception** telegram.**TelegramError** (*message*)

Bases: exceptions.Exception

This object represents a Telegram Error.

**class** telegram.**TelegramObject**

Bases: object

Base class for most telegram objects.

**static de\_json** (*data, bot*)

**Parameters**

• **data** (*dict*) –

• **bot** (*telegram.Bot*) –

**Returns**

**Return type**dict

**to\_dict** ()

**Returns**

**Return type**dict

**to\_json** ()

**Returns**

**Return type**str

**class** telegram.**Update** (*update\_id, message=None, edited\_message=None, inline\_query=None, chosen\_inline\_result=None, callback\_query=None, channel\_post=None, edited\_channel\_post=None, \*\*kwargs*)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram Update.

**update\_id**

*int* – The update’s unique identifier.

**message**

*telegram.Message* – New incoming message of any kind - text, photo, sticker, etc.

**edited\_message**

*telegram.Message* – New version of a message that is known to the bot and was edited

**inline\_query**

*telegram.InlineQuery* – New incoming inline query.

**chosen\_inline\_result**

*telegram.ChosenInlineResult* – The result of an inline query that was chosen by a user and sent to their chat partner.

**callback\_query**

*telegram.CallbackQuery* – New incoming callback query.

**channel\_post**

Optional[*telegram.Message*] – New incoming channel post of any kind - text, photo, sticker, etc.

**edited\_channel\_post**

Optional[*telegram.Message*] – New version of a channel post that is known to the bot and was edited.

**Parameters**

• **update\_id** (*int*) –

• **message** (Optional[*telegram.Message*]) –

- **edited\_message** (Optional[*telegram.Message*]) –
- **inline\_query** (Optional[*telegram.InlineQuery*]) –
- **chosen\_inline\_result** (Optional[*telegram.ChosenInlineResult*]) –
- **callback\_query** (Optional[*telegram.CallbackQuery*]) –
- **channel\_post** (Optional[*telegram.Message*]) –
- **edited\_channel\_post** (Optional[*telegram.Message*]) –
- **\*\*kwargs** – Arbitrary keyword arguments.

**static de\_json** (*data*, *bot*)

#### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

#### Returns

Return type *telegram.Update*

#### **effective\_chat**

A property that contains the *Chat* that this update was sent in, no matter what kind of update this is. Will be *None* for inline queries and chosen inline results.

#### **effective\_message**

A property that contains the *Message* included in this update, no matter what kind of update this is. Will be *None* for inline queries, chosen inline results and callback queries from inline messages.

#### **effective\_user**

A property that contains the *User* that sent this update, no matter what kind of update this is. Will be *None* for channel posts.

**class telegram.User** (*id*, *first\_name*, *type=None*, *last\_name=None*, *username=None*, *bot=None*, *\*\*kwargs*)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram User.

#### **id**

*int*

#### **first\_name**

*str*

#### **last\_name**

*str*

#### **username**

*str*

#### **type**

*str*

#### Parameters

- **id** (*int*) –
- **first\_name** (*str*) –
- **\*\*kwargs** – Arbitrary keyword arguments.

#### Keyword Arguments

- **type** (Optional[*str*]) –
- **last\_name** (Optional[*str*]) –

- username** (*Optional[str]*) –
- bot** (*Optional[Bot]*) – The Bot to use for instance methods

**static de\_json** (*data, bot*)

**Parameters**

- data** (*dict*) –
- bot** (*telegram.Bot*) –

**Returns**

**Return type***telegram.User*

**get\_profile\_photos** (*\*args, \*\*kwargs*)

Shortcut for `bot.getUserProfilePhotos(update.message.from_user.id, *args, **kwargs)`

**name**

*str*

**class telegram.UserProfilePhotos** (*total\_count, photos, \*\*kwargs*)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram UserProfilePhotos.

**total\_count**

*int*

**photos**

List[List[*telegram.PhotoSize*]]

**Parameters**

- total\_count** (*int*) –
- photos** (List[List[*telegram.PhotoSize*]]) –

**static de\_json** (*data, bot*)

**Parameters**

- data** (*dict*) –
- bot** (*telegram.Bot*) –

**Returns**

**Return type***telegram.UserProfilePhotos*

**to\_dict** ()

**Returns**

**Return type***dict*

**class telegram.Venue** (*location, title, address, foursquare\_id=None, \*\*kwargs*)

Bases: *telegram.base.TelegramObject*

This object represents a venue.

**Parameters**

- location** (*telegram.Location*) –
- title** (*str*) –
- address** (*str*) –
- foursquare\_id** (*Optional[str]*) –

```
static 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 Telegram Video.

**file\_id**

*str*

**width**

*int*

**height**

*int*

**duration**

*int*

**thumb**

*telegram.PhotoSize*

**mime\_type**

*str*

**file\_size**

*int*

#### Parameters

- **file\_id** (*str*) –
- **width** (*int*) –
- **height** (*int*) –
- **duration** (*int*) –
- **\*\*kwargs** – Arbitrary keyword arguments.

#### Keyword Arguments

- **thumb** (Optional[*telegram.PhotoSize*]) –
- **mime\_type** (Optional[*str*]) –
- **file\_size** (Optional[*int*]) –

```
static de_json (data, bot)
```

#### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

#### Returns

Return type *telegram.Video*

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

Bases: *telegram.base.TelegramObject*

This object represents a Telegram Voice.

**file\_id**

*str*

**duration**

*int*

**mime\_type**

*str*

**file\_size**

*int*

#### Parameters

- **file\_id** (*str*) –
- **duration** (*Optional[int]*) –
- **\*\*kwargs** – Arbitrary keyword arguments.

#### Keyword Arguments

- **mime\_type** (*Optional[str]*) –
- **file\_size** (*Optional[int]*) –

**static de\_json** (*data, bot*)

#### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

#### Returns

Return type *telegram.Voice*

```
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.

**url**

*str* – Webhook URL, may be empty if webhook is not set up.

**has\_custom\_certificate**

*bool*

**pending\_update\_count**

*int*

**last\_error\_date**

*int*

**last\_error\_message**

*str*

#### Parameters

- **url** (*str*) – Webhook URL, may be empty if webhook is not set up.
- **has\_custom\_certificate** (*bool*) –
- **pending\_update\_count** (*int*) –
- **last\_error\_date** (*Optional[int]*) –
- **last\_error\_message** (*Optional[str]*) –

**static de\_json** (*data, bot*)

#### Parameters

- **data** (*dict*) –



•**bot** (`telegram.Bot`) –

### Returns

**Return type** `telegram.WebhookInfo`

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

This object represents a Telegram Animation.

### file\_id

*str* – Unique file identifier.

### Keyword Arguments

- thumb** (Optional[`telegram.PhotoSize`]) – Animation thumbnail as defined by sender.
- file\_name** (Optional[*str*]) – Original animation filename as defined by sender.
- mime\_type** (Optional[*str*]) – MIME type of the file as defined by sender.
- file\_size** (Optional[*int*]) – File size.

**static de\_json** (*data*, *bot*)

### Parameters

- data** (*dict*) –
- bot** (`telegram.Bot`) –

### Returns

**Return type** `telegram.Game`

```
class telegram.Game (title, description, photo, text=None, text_entities=None, animation=None,
                    **kwargs)
Bases: telegram.base.TelegramObject
```

This object represents a Telegram Game.

### title

*str* – Title of the game.

### description

*str* – Description of the game.

### photo

list[`telegram.PhotoSize`] – List of photos that will be displayed in the game message in chats.

### Keyword Arguments

- text** (Optional[*str*]) – 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 `setGameScore`, or manually edited using `editMessageText`. 0-4096 characters.
- text\_entities** (Optional[list[`telegram.MessageEntity`]]) – Special entities that appear in text, such as usernames, URLs, bot commands, etc.
- animation** (Optional[`telegram.Animation`]) – Animation that will be displayed in the game message in chats. Upload via `BotFather`.

**static de\_json** (*data*, *bot*)

### Parameters

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

**Returns**

**Return type***telegram.Game*

**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 *entities* attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See *get\_entity\_text* for more info.

---

**Parameterstypes** (*Optional[list]*) – List of *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\_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.)

---

**Parametersentity** (*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** ()

**Returns**

**Return type**dict

**class telegram.GameHighScore** (*position, user, score*)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram GameHighScore.

**position**

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

**user**

*telegram.User* – User object.

**score**

*int* – Score.

**static de\_json** (*data, bot*)

**Parameters**

- **data** (*dict*) –
- **bot** (*telegram.Bot*) –

**Returns**

**Return type** *telegram.Game*



## CHAPTER 2

---

### Indices and tables

---

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



### t

telegram, 119

telegram.animation, 38

telegram.audio, 39

telegram.base, 39

telegram.bot, 40

telegram.callbackgame, 71

telegram.callbackquery, 71

telegram.chat, 72

telegram.chataction, 73

telegram.chatmember, 74

telegram.choseninlineresult, 74

telegram.constants, 75

telegram.contact, 76

telegram.contrib, 1

telegram.contrib.botan, 1

telegram.document, 77

telegram.error, 77

telegram.ext, 22

telegram.ext.callbackqueryhandler, 10

telegram.ext.choseninlineresulthandler, 10

telegram.ext.commandhandler, 13

telegram.ext.conversationhandler, 11

telegram.ext.dispatcher, 3

telegram.ext.filters, 18

telegram.ext.handler, 9

telegram.ext.inlinequeryhandler, 14

telegram.ext.jobqueue, 5

telegram.ext.messagehandler, 15

telegram.ext.messagequeue, 15

telegram.ext.regexhandler, 19

telegram.ext.stringcommandhandler, 20

telegram.ext.stringregexhandler, 21

telegram.ext.typehandler, 22

telegram.ext.updater, 1

telegram.file, 78

telegram.forcereply, 79

telegram.game, 79

telegram.gamehighscore, 80

telegram.inlinekeyboardbutton, 81

telegram.inlinekeyboardmarkup, 82

telegram.inlinequery, 82

telegram.inlinequeryresult, 83

telegram.inlinequeryresultarticle, 84

telegram.inlinequeryresultaudio, 85

telegram.inlinequeryresultcachedaudio, 86

telegram.inlinequeryresultcacheddocument, 87

telegram.inlinequeryresultcachedgif, 88

telegram.inlinequeryresultcachedmpeg4gif, 89

telegram.inlinequeryresultcachedphoto, 90

telegram.inlinequeryresultcachedsticker, 91

telegram.inlinequeryresultcachedvideo, 91

telegram.inlinequeryresultcachedvoice, 92

telegram.inlinequeryresultcontact, 93

telegram.inlinequeryresultdocument, 94

telegram.inlinequeryresultgame, 95

telegram.inlinequeryresultgif, 96

telegram.inlinequeryresultlocation, 97

telegram.inlinequeryresultmpeg4gif, 98

telegram.inlinequeryresultphoto, 99

telegram.inlinequeryresultvenue, 100

telegram.inlinequeryresultvideo, 100

telegram.inlinequeryresultvoice, 100

telegram.inputcontactmessagecontent, 101

telegram.inputfile, 101

telegram.inputlocationmessagecontent, 101

telegram.inputmessagecontent, 102

telegram.inputtextmessagecontent, 102

telegram.inputvenuemessagecontent, 102

telegram.keyboardbutton, 102

telegram.location, 103

telegram.message, 103

telegram.messageentity, 109

telegram.parsemode, 110

telegram.photosize, 110

telegram.replykeyboardmarkup, 112

telegram.replykeyboardremove, 111

`telegram.replymarkup`, 113  
`telegram.sticker`, 113  
`telegram.update`, 114  
`telegram.user`, 115  
`telegram.userprofilephotos`, 116  
`telegram.venue`, 116  
`telegram.video`, 117  
`telegram.voice`, 117  
`telegram.webhookinfo`, 118



## Symbols

- `_all_delayq` (telegram.ext.messagequeue.MessageQueue attribute), 17
  - `_group_delayq` (telegram.ext.messagequeue.MessageQueue attribute), 17
  - `_is_messages_queued_default` (telegram.ext.messagequeue.self attribute), 17
  - `_msg_queue` (telegram.ext.messagequeue.self attribute), 17
- ## A
- `add_error_handler()` (telegram.ext.Dispatcher method), 22
  - `add_error_handler()` (telegram.ext.dispatcher.Dispatcher method), 3
  - `add_handler()` (telegram.ext.Dispatcher method), 23
  - `add_handler()` (telegram.ext.dispatcher.Dispatcher method), 4
  - `ADMINISTRATOR` (telegram.ChatMember attribute), 152
  - `ADMINISTRATOR` (telegram.chatmember.ChatMember attribute), 74
  - `all` (telegram.ext.Filters attribute), 34
  - `all` (telegram.ext.filters.Filters attribute), 18
  - `all_members_are_administrators` (telegram.Chat attribute), 151
  - `all_members_are_administrators` (telegram.chat.Chat attribute), 72
  - `ALL_TYPES` (telegram.MessageEntity attribute), 180
  - `ALL_TYPES` (telegram.messageentity.MessageEntity attribute), 110
  - `Animation` (class in telegram), 189
  - `Animation` (class in telegram.animation), 38
  - `answer()` (telegram.CallbackQuery method), 154
  - `answer()` (telegram.callbackquery.CallbackQuery method), 71
  - `answer()` (telegram.InlineQuery method), 159
  - `answer()` (telegram.inlinequery.InlineQuery method), 83
  - `answer_callback_query()` (telegram.Bot method), 121
  - `answer_callback_query()` (telegram.bot.Bot method), 41
  - `answer_inline_query()` (telegram.Bot method), 122
  - `answer_inline_query()` (telegram.bot.Bot method), 42
  - `answerCallbackQuery()` (telegram.Bot method), 120
  - `answerCallbackQuery()` (telegram.bot.Bot method), 40
  - `answerInlineQuery()` (telegram.Bot method), 120
  - `answerInlineQuery()` (telegram.bot.Bot method), 41
  - `Audio` (class in telegram), 119
  - `Audio` (class in telegram.audio), 39
  - `audio` (telegram.ext.Filters attribute), 34
  - `audio` (telegram.ext.filters.Filters attribute), 18
  - `audio` (telegram.Message attribute), 175
  - `audio` (telegram.message.Message attribute), 104
  - `audio_duration` (telegram.InlineQueryResultAudio attribute), 161
  - `audio_duration` (telegram.inlinequeryresultaudio.InlineQueryResultAudio attribute), 85
  - `audio_file_id` (telegram.InlineQueryResultCachedAudio attribute), 162
  - `audio_file_id` (telegram.inlinequeryresultcachedaudio.InlineQueryResultCachedAudio attribute), 86
  - `audio_url` (telegram.InlineQueryResultAudio attribute), 161
  - `audio_url` (telegram.inlinequeryresultaudio.InlineQueryResultAudio attribute), 85
- ## B
- `BadRequest`, 77
  - `BaseFilter` (class in telegram.ext), 33
  - `BaseFilter` (class in telegram.ext.filters), 18
  - `BOLD` (telegram.MessageEntity attribute), 180
  - `BOLD` (telegram.messageentity.MessageEntity attribute), 110
  - `Bot` (class in telegram), 120
  - `Bot` (class in telegram.bot), 40
  - `bot` (telegram.ext.JobQueue attribute), 24
  - `bot` (telegram.ext.jobqueue.JobQueue attribute), 7
  - `BOT_COMMAND` (telegram.MessageEntity attribute), 180
  - `BOT_COMMAND` (telegram.messageentity.MessageEntity attribute), 110
  - `Botan` (class in telegram.contrib.botan), 1

## C

- callback (telegram.ext.Job attribute), 26
- callback (telegram.ext.jobqueue.Job attribute), 5
- callback\_data (telegram.InlineKeyboardButton attribute), 157
- callback\_data (telegram.inlinekeyboardbutton.InlineKeyboardButton attribute), 81
- callback\_game (telegram.InlineKeyboardButton attribute), 157
- callback\_game (telegram.inlinekeyboardbutton.InlineKeyboardButton attribute), 81
- callback\_query (telegram.Update attribute), 184
- callback\_query (telegram.update.Update attribute), 114
- CallbackGame (class in telegram.callbackgame), 71
- CallbackQuery (class in telegram), 154
- CallbackQuery (class in telegram.callbackquery), 71
- CallbackQueryHandler (class in telegram.ext), 28
- CallbackQueryHandler (class in telegram.ext.callbackqueryhandler), 10
- caption (telegram.InlineQueryResultAudio attribute), 161
- caption (telegram.inlinequeryresultaudio.InlineQueryResultAudio attribute), 85
- caption (telegram.InlineQueryResultCachedAudio attribute), 162
- caption (telegram.inlinequeryresultcachedaudio.InlineQueryResultCachedAudio attribute), 86
- caption (telegram.InlineQueryResultCachedDocument attribute), 162
- caption (telegram.inlinequeryresultcacheddocument.InlineQueryResultCachedDocument attribute), 87
- caption (telegram.InlineQueryResultCachedGif attribute), 163
- caption (telegram.inlinequeryresultcachedgif.InlineQueryResultCachedGif attribute), 88
- caption (telegram.InlineQueryResultCachedMpeg4Gif attribute), 164
- caption (telegram.inlinequeryresultcachedmpeg4gif.InlineQueryResultCachedMpeg4Gif attribute), 89
- caption (telegram.InlineQueryResultCachedPhoto attribute), 164
- caption (telegram.inlinequeryresultcachedphoto.InlineQueryResultCachedPhoto attribute), 90
- caption (telegram.InlineQueryResultCachedVideo attribute), 166
- caption (telegram.inlinequeryresultcachedvideo.InlineQueryResultCachedVideo attribute), 92
- caption (telegram.InlineQueryResultCachedVoice attribute), 166
- caption (telegram.inlinequeryresultcachedvoice.InlineQueryResultCachedVoice attribute), 93
- caption (telegram.InlineQueryResultDocument attribute), 168
- caption (telegram.inlinequeryresultdocument.InlineQueryResultDocument attribute), 95
- caption (telegram.InlineQueryResultGif attribute), 169
- caption (telegram.inlinequeryresultgif.InlineQueryResultGif attribute), 96
- caption (telegram.InlineQueryResultMpeg4Gif attribute), 171
- caption (telegram.inlinequeryresultmpeg4gif.InlineQueryResultMpeg4Gif attribute), 98
- caption (telegram.InlineQueryResultPhoto attribute), 172
- caption (telegram.inlinequeryresultphoto.InlineQueryResultPhoto attribute), 99
- caption (telegram.Message attribute), 175
- caption (telegram.message.Message attribute), 104
- CHANNEL (telegram.Chat attribute), 151
- CHANNEL (telegram.chat.Chat attribute), 73
- channel\_chat\_created (telegram.Message attribute), 175
- channel\_chat\_created (telegram.message.Message attribute), 105
- channel\_post (telegram.Update attribute), 184
- channel\_post (telegram.update.Update attribute), 114
- Chat (class in telegram), 151
- Chat (class in telegram.chat), 72
- chat\_data (telegram.ext.dispatcher.Dispatcher attribute), 4
- chat\_id (telegram.Message attribute), 176
- chat\_id (telegram.message.Message attribute), 106
- ChatAction (class in telegram), 153
- ChatAction (class in telegram.chataction), 73
- ChatMember (class in telegram), 152
- ChatMember (class in telegram.chatmember), 74
- ChatMigrated, 77
- check\_update() (telegram.ext.CallbackQueryHandler method), 29
- check\_update() (telegram.ext.callbackqueryhandler.CallbackQueryHandler method), 10
- check\_update() (telegram.ext.ChosenInlineResultHandler method), 30
- check\_update() (telegram.ext.choseninlineresulthandler.ChosenInlineResultHandler method), 11
- check\_update() (telegram.ext.CommandHandler method), 31
- check\_update() (telegram.ext.commandhandler.CommandHandler method), 13
- check\_update() (telegram.ext.ConversationHandler method), 38
- check\_update() (telegram.ext.conversationhandler.ConversationHandler method), 12
- check\_update() (telegram.ext.Handler method), 31
- check\_update() (telegram.ext.handler.Handler method), 9
- check\_update() (telegram.ext.InlineQueryHandler method), 32
- check\_update() (telegram.ext.inlinequeryhandler.InlineQueryHandler method), 14

check\_update() (telegram.ext.MessageHandler method), 33  
 check\_update() (telegram.ext.messagehandler.MessageHandler method), 15  
 check\_update() (telegram.ext.RegexHandler method), 35  
 check\_update() (telegram.ext.regexhandler.RegexHandler method), 20  
 check\_update() (telegram.ext.StringCommandHandler method), 36  
 check\_update() (telegram.ext.stringcommandhandler.StringCommandHandler method), 21  
 check\_update() (telegram.ext.StringRegexHandler method), 36  
 check\_update() (telegram.ext.stringregexhandler.StringRegexHandler method), 21  
 check\_update() (telegram.ext.TypeHandler method), 37  
 check\_update() (telegram.ext.typehandler.TypeHandler method), 22  
 checkUpdate() (telegram.ext.ChosenInlineResultHandler method), 30  
 checkUpdate() (telegram.ext.choseninlineresulthandler.ChosenInlineResultHandler method), 11  
 checkUpdate() (telegram.ext.InlineQueryHandler method), 32  
 checkUpdate() (telegram.ext.inlinequeryhandler.InlineQueryHandler method), 14  
 checkUpdate() (telegram.ext.RegexHandler method), 35  
 checkUpdate() (telegram.ext.regexhandler.RegexHandler method), 20  
 checkUpdate() (telegram.ext.TypeHandler method), 37  
 checkUpdate() (telegram.ext.typehandler.TypeHandler method), 22  
 chosen\_inline\_result (telegram.Update attribute), 184  
 chosen\_inline\_result (telegram.update.Update attribute), 114  
 ChosenInlineResult (class in telegram), 153  
 ChosenInlineResult (class in telegram.choseninlineresult), 74  
 ChosenInlineResultHandler (class in telegram.ext), 29  
 ChosenInlineResultHandler (class in telegram.ext.choseninlineresulthandler), 10  
 CODE (telegram.MessageEntity attribute), 180  
 CODE (telegram.messageentity.MessageEntity attribute), 110  
 collect\_optional\_args() (telegram.ext.Handler method), 31  
 collect\_optional\_args() (telegram.ext.handler.Handler method), 9  
 command (telegram.ext.Filters attribute), 34  
 command (telegram.ext.filters.Filters attribute), 18  
 CommandHandler (class in telegram.ext), 30  
 CommandHandler (class in telegram.ext.commandhandler), 13  
 Contact (class in telegram), 154  
 Contact (class in telegram.contact), 76  
 contact (telegram.ext.Filters attribute), 34  
 contact (telegram.ext.filters.Filters attribute), 18  
 contact (telegram.Message attribute), 175  
 contact (telegram.message.Message attribute), 104  
 content\_type (telegram.InputFile attribute), 172  
 content\_type (telegram.inputfile.InputFile attribute), 101  
 ConversationHandler (class in telegram.ext), 37  
 ConversationHandler (class in telegram.ext.conversationhandler), 11  
 CREATOR (telegram.ChatMember attribute), 152  
 CREATOR (telegram.chatmember.ChatMember attribute), 74  
**D**  
 date (telegram.Message attribute), 174  
 date (telegram.message.Message attribute), 103  
 Days (class in telegram.ext.jobqueue), 5  
 days (telegram.ext.Job attribute), 26  
 days (telegram.ext.jobqueue.Job attribute), 6  
 de\_json() (telegram.Animation static method), 189  
 de\_json() (telegram.animation.Animation static method), 30  
 de\_json() (telegram.Audio static method), 119  
 de\_json() (telegram.audio.Audio static method), 39  
 de\_json() (telegram.base.TelegramObject static method), 39  
 de\_json() (telegram.Bot static method), 122  
 de\_json() (telegram.CallbackQuery static method), 154  
 de\_json() (telegram.callbackquery.CallbackQuery static method), 71  
 de\_json() (telegram.Chat static method), 151  
 de\_json() (telegram.chat.Chat static method), 73  
 de\_json() (telegram.ChatMember static method), 152  
 de\_json() (telegram.chatmember.ChatMember static method), 74  
 de\_json() (telegram.ChosenInlineResult static method), 153  
 de\_json() (telegram.choseninlineresult.ChosenInlineResult static method), 75  
 de\_json() (telegram.Contact static method), 155  
 de\_json() (telegram.contact.Contact static method), 76  
 de\_json() (telegram.Document static method), 155  
 de\_json() (telegram.document.Document static method), 77  
 de\_json() (telegram.File static method), 156  
 de\_json() (telegram.file.File static method), 78  
 de\_json() (telegram.ForceReply static method), 157  
 de\_json() (telegram.forcereply.ForceReply static method), 79  
 de\_json() (telegram.Game static method), 189  
 de\_json() (telegram.game.Game static method), 80  
 de\_json() (telegram.GameHighScore static method), 190

`de_json()` (telegram.gamehighscore.GameHighScore static method), 81  
`de_json()` (telegram.InlineKeyboardButton static method), 157  
`de_json()` (telegram.inlinekeyboardbutton.InlineKeyboardButton static method), 82  
`de_json()` (telegram.InlineKeyboardMarkup static method), 158  
`de_json()` (telegram.inlinekeyboardmarkup.InlineKeyboardMarkup static method), 82  
`de_json()` (telegram.InlineQuery static method), 159  
`de_json()` (telegram.inlinequery.InlineQuery static method), 83  
`de_json()` (telegram.InlineQueryResult static method), 159  
`de_json()` (telegram.inlinequeryresult.InlineQueryResult static method), 84  
`de_json()` (telegram.InlineQueryResultArticle static method), 160  
`de_json()` (telegram.inlinequeryresultarticle.InlineQueryResultArticle static method), 85  
`de_json()` (telegram.InlineQueryResultAudio static method), 161  
`de_json()` (telegram.inlinequeryresultaudio.InlineQueryResultAudio static method), 86  
`de_json()` (telegram.InlineQueryResultCachedAudio static method), 162  
`de_json()` (telegram.inlinequeryresultcachedaudio.InlineQueryResultCachedAudio static method), 87  
`de_json()` (telegram.InlineQueryResultCachedDocument static method), 163  
`de_json()` (telegram.inlinequeryresultcacheddocument.InlineQueryResultCachedDocument static method), 88  
`de_json()` (telegram.InlineQueryResultCachedGif static method), 163  
`de_json()` (telegram.inlinequeryresultcachedgif.InlineQueryResultCachedGif static method), 89  
`de_json()` (telegram.InlineQueryResultCachedMpeg4Gif static method), 164  
`de_json()` (telegram.inlinequeryresultcachedmpeg4gif.InlineQueryResultCachedMpeg4Gif static method), 89  
`de_json()` (telegram.InlineQueryResultCachedPhoto static method), 165  
`de_json()` (telegram.inlinequeryresultcachedphoto.InlineQueryResultCachedPhoto static method), 90  
`de_json()` (telegram.InlineQueryResultCachedSticker static method), 165  
`de_json()` (telegram.inlinequeryresultcachedsticker.InlineQueryResultCachedSticker static method), 91  
`de_json()` (telegram.InlineQueryResultCachedVideo static method), 166  
`de_json()` (telegram.inlinequeryresultcachedvideo.InlineQueryResultCachedVideo static method), 92  
`de_json()` (telegram.InlineQueryResultCachedVoice static method), 167  
`de_json()` (telegram.inlinequeryresultcachedvoice.InlineQueryResultCachedVoice static method), 93  
`de_json()` (telegram.InlineQueryResultContact static method), 167  
`de_json()` (telegram.inlinequeryresultcontact.InlineQueryResultContact static method), 94  
`de_json()` (telegram.InlineQueryResultDocument static method), 168  
`de_json()` (telegram.inlinequeryresultdocument.InlineQueryResultDocument static method), 95  
`de_json()` (telegram.InlineQueryResultGame static method), 172  
`de_json()` (telegram.inlinequeryresultgame.InlineQueryResultGame static method), 96  
`de_json()` (telegram.InlineQueryResultGif static method), 169  
`de_json()` (telegram.inlinequeryresultgif.InlineQueryResultGif static method), 97  
`de_json()` (telegram.InlineQueryResultLocation static method), 170  
`de_json()` (telegram.inlinequeryresultlocation.InlineQueryResultLocation static method), 98  
`de_json()` (telegram.InlineQueryResultMpeg4Gif static method), 171  
`de_json()` (telegram.inlinequeryresultmpeg4gif.InlineQueryResultMpeg4Gif static method), 99  
`de_json()` (telegram.InlineQueryResultPhoto static method), 172  
`de_json()` (telegram.inlinequeryresultphoto.InlineQueryResultPhoto static method), 100  
`de_json()` (telegram.InlineQueryResultVenue static method), 172  
`de_json()` (telegram.inlinequeryresultvenue.InlineQueryResultVenue static method), 100  
`de_json()` (telegram.InlineQueryResultVideo static method), 172  
`de_json()` (telegram.inlinequeryresultvideo.InlineQueryResultVideo static method), 100  
`de_json()` (telegram.InlineQueryResultVoice static method), 172  
`de_json()` (telegram.inlinequeryresultvoice.InlineQueryResultVoice static method), 100  
`de_json()` (telegram.InputContactMessageContent static method), 172  
`de_json()` (telegram.inputcontactmessagecontent.InputContactMessageContent static method), 101  
`de_json()` (telegram.InputLocationMessageContent static method), 173  
`de_json()` (telegram.inputlocationmessagecontent.InputLocationMessageContent static method), 101  
`de_json()` (telegram.InputMessageContent static method), 173  
`de_json()` (telegram.inputmessagecontent.InputMessageContent static method), 102  
`de_json()` (telegram.InputTextMessageContent static method), 173  
`de_json()` (telegram.inputtextmessagecontent.InputTextMessageContent static method), 102  
`de_json()` (telegram.InputVenueMessageContent static method), 173  
`de_json()` (telegram.inputvenuemessagecontent.InputVenueMessageContent static method), 102

- static method), 102
- de\_json() (telegram.KeyboardButton static method), 173
- de\_json() (telegram.keyboardbutton.KeyboardButton static method), 102
- de\_json() (telegram.Location static method), 174
- de\_json() (telegram.location.Location static method), 103
- de\_json() (telegram.Message static method), 176
- de\_json() (telegram.message.Message static method), 106
- de\_json() (telegram.MessageEntity static method), 181
- de\_json() (telegram.messageentity.MessageEntity static method), 110
- de\_json() (telegram.PhotoSize static method), 181
- de\_json() (telegram.photosize.PhotoSize static method), 111
- de\_json() (telegram.ReplyKeyboardMarkup static method), 182
- de\_json() (telegram.replykeyboardmarkup.ReplyKeyboardMarkup static method), 112
- de\_json() (telegram.ReplyKeyboardRemove static method), 182
- de\_json() (telegram.replykeyboardremove.ReplyKeyboardRemove static method), 112
- de\_json() (telegram.ReplyMarkup static method), 183
- de\_json() (telegram.replymarkup.ReplyMarkup static method), 113
- de\_json() (telegram.Sticker static method), 183
- de\_json() (telegram.sticker.Sticker static method), 113
- de\_json() (telegram.TelegramObject static method), 184
- de\_json() (telegram.Update static method), 185
- de\_json() (telegram.update.Update static method), 114
- de\_json() (telegram.User static method), 186
- de\_json() (telegram.user.User static method), 115
- de\_json() (telegram.UserProfilePhotos static method), 186
- de\_json() (telegram.userprofilephotos.UserProfilePhotos static method), 116
- de\_json() (telegram.Venue static method), 186
- de\_json() (telegram.venue.Venue static method), 116
- de\_json() (telegram.Video static method), 187
- de\_json() (telegram.video.Video static method), 117
- de\_json() (telegram.Voice static method), 188
- de\_json() (telegram.voice.Voice static method), 118
- de\_json() (telegram.WebhookInfo static method), 188
- de\_json() (telegram.webhookinfo.WebhookInfo static method), 119
- de\_list() (telegram.InlineKeyboardButton static method), 158
- de\_list() (telegram.inlinekeyboardbutton.InlineKeyboardButton static method), 82
- de\_list() (telegram.KeyboardButton static method), 173
- de\_list() (telegram.keyboardbutton.KeyboardButton static method), 102
- de\_list() (telegram.MessageEntity static method), 181
- de\_list() (telegram.messageentity.MessageEntity static method), 110
- de\_list() (telegram.PhotoSize static method), 181
- de\_list() (telegram.photosize.PhotoSize static method), 111
- DelayQueue (class in telegram.ext.messagequeue), 15
- DelayQueueError, 16
- delete() (telegram.Message method), 177
- delete() (telegram.message.Message method), 106
- delete\_chat\_photo (telegram.Message attribute), 175
- delete\_chat\_photo (telegram.message.Message attribute), 104
- delete\_message() (telegram.Bot method), 123
- delete\_message() (telegram.bot.Bot method), 43
- delete\_webhook() (telegram.Bot method), 123
- delete\_webhook() (telegram.bot.Bot method), 43
- deleteMessage() (telegram.Bot method), 122
- deleteMessage() (telegram.bot.Bot method), 42
- deleteWebhook() (telegram.Bot method), 122
- deleteWebhook() (telegram.bot.Bot method), 43
- description (telegram.Game attribute), 189
- description (telegram.game.Game attribute), 79
- description (telegram.InlineQueryResultArticle attribute), 160
- description (telegram.inlinequeryresultarticle.InlineQueryResultArticle attribute), 84
- description (telegram.InlineQueryResultCachedDocument attribute), 162
- description (telegram.inlinequeryresultcacheddocument.InlineQueryResultCachedDocument attribute), 87
- description (telegram.InlineQueryResultCachedPhoto attribute), 164
- description (telegram.inlinequeryresultcachedphoto.InlineQueryResultCachedPhoto attribute), 90
- description (telegram.InlineQueryResultCachedVideo attribute), 165
- description (telegram.inlinequeryresultcachedvideo.InlineQueryResultCachedVideo attribute), 92
- description (telegram.InlineQueryResultDocument attribute), 168
- description (telegram.inlinequeryresultdocument.InlineQueryResultDocument attribute), 95
- description (telegram.InlineQueryResultPhoto attribute), 171
- description (telegram.inlinequeryresultphoto.InlineQueryResultPhoto attribute), 99
- dispatch\_error() (telegram.ext.Dispatcher method), 23
- dispatch\_error() (telegram.ext.dispatcher.Dispatcher method), 4
- Dispatcher (class in telegram.ext), 22
- Dispatcher (class in telegram.ext.dispatcher), 3
- Document (class in telegram), 155
- Document (class in telegram.document), 77
- document (telegram.ext.Filters attribute), 34
- document (telegram.ext.filters.Filters attribute), 18
- document (telegram.Message attribute), 175
- document (telegram.message.Message attribute), 104
- document\_file\_id (telegram.InlineQueryResultCachedDocument attribute), 160

attribute), 162  
 document\_file\_id (telegram.inlinequeryresultcacheddocument.InlineQueryResultCachedDocument attribute), 87  
 document\_url (telegram.InlineQueryResultDocument attribute), 168  
 document\_url (telegram.inlinequeryresultdocument.InlineQueryResultDocument attribute), 95  
 download() (telegram.File method), 156  
 download() (telegram.file.File method), 78  
 duration (telegram.Audio attribute), 119  
 duration (telegram.audio.Audio attribute), 39  
 duration (telegram.Video attribute), 187  
 duration (telegram.video.Video attribute), 117  
 duration (telegram.Voice attribute), 187  
 duration (telegram.voice.Voice attribute), 118

## E

edit\_caption() (telegram.Message method), 177  
 edit\_caption() (telegram.message.Message method), 106  
 edit\_date (telegram.Message attribute), 174  
 edit\_date (telegram.message.Message attribute), 104  
 edit\_message\_caption() (telegram.Bot method), 125  
 edit\_message\_caption() (telegram.bot.Bot method), 45  
 edit\_message\_caption() (telegram.CallbackQuery method), 154  
 edit\_message\_caption() (telegram.callbackquery.CallbackQuery method), 71  
 edit\_message\_reply\_markup() (telegram.Bot method), 125  
 edit\_message\_reply\_markup() (telegram.bot.Bot method), 46  
 edit\_message\_reply\_markup() (telegram.CallbackQuery method), 154  
 edit\_message\_reply\_markup() (telegram.callbackquery.CallbackQuery method), 72  
 edit\_message\_text() (telegram.Bot method), 126  
 edit\_message\_text() (telegram.bot.Bot method), 46  
 edit\_message\_text() (telegram.CallbackQuery method), 154  
 edit\_message\_text() (telegram.callbackquery.CallbackQuery method), 72  
 edit\_reply\_markup() (telegram.Message method), 177  
 edit\_reply\_markup() (telegram.message.Message method), 106  
 edit\_text() (telegram.Message method), 177  
 edit\_text() (telegram.message.Message method), 106  
 edited\_channel\_post (telegram.Update attribute), 184  
 edited\_channel\_post (telegram.update.Update attribute), 114  
 edited\_message (telegram.Update attribute), 184  
 edited\_message (telegram.update.Update attribute), 114  
 editMessageCaption() (telegram.Bot method), 123

editMessageCaption() (telegram.bot.Bot method), 43  
 editMessageReplyMarkup() (telegram.Bot method), 44  
 editMessageReplyMarkup() (telegram.bot.Bot method), 44  
 editMessageText() (telegram.Bot method), 124  
 editMessageText() (telegram.bot.Bot method), 44  
 effective\_chat (telegram.Update attribute), 185  
 effective\_chat (telegram.update.Update attribute), 114  
 effective\_message (telegram.Update attribute), 185  
 effective\_message (telegram.update.Update attribute), 115  
 effective\_user (telegram.Update attribute), 185  
 effective\_user (telegram.update.Update attribute), 115  
 EMAIL (telegram.MessageEntity attribute), 180  
 EMAIL (telegram.messageentity.MessageEntity attribute), 110  
 emoji (telegram.Sticker attribute), 183  
 emoji (telegram.sticker.Sticker attribute), 113  
 enabled (telegram.ext.Job attribute), 26  
 enabled (telegram.ext.jobqueue.Job attribute), 6  
 END (telegram.ext.ConversationHandler attribute), 38  
 END (telegram.ext.conversationhandler.ConversationHandler attribute), 12  
 entry\_points (telegram.ext.conversationhandler.ConversationHandler attribute), 12  
 EVERY\_DAY (telegram.ext.jobqueue.Days attribute), 5

## F

fallbacks (telegram.ext.conversationhandler.ConversationHandler attribute), 12  
 File (class in telegram), 156  
 File (class in telegram.file), 78  
 file\_id (telegram.Animation attribute), 189  
 file\_id (telegram.animation.Animation attribute), 38  
 file\_id (telegram.Audio attribute), 119  
 file\_id (telegram.audio.Audio attribute), 39  
 file\_id (telegram.Document attribute), 155  
 file\_id (telegram.document.Document attribute), 77  
 file\_id (telegram.File attribute), 156  
 file\_id (telegram.file.File attribute), 78  
 file\_id (telegram.PhotoSize attribute), 181  
 file\_id (telegram.photosize.PhotoSize attribute), 110  
 file\_id (telegram.Sticker attribute), 183  
 file\_id (telegram.sticker.Sticker attribute), 113  
 file\_id (telegram.Video attribute), 187  
 file\_id (telegram.video.Video attribute), 117  
 file\_id (telegram.Voice attribute), 187  
 file\_id (telegram.voice.Voice attribute), 118  
 file\_name (telegram.Document attribute), 155  
 file\_name (telegram.document.Document attribute), 77  
 file\_path (telegram.File attribute), 156  
 file\_path (telegram.file.File attribute), 78  
 file\_size (telegram.Audio attribute), 119  
 file\_size (telegram.audio.Audio attribute), 39  
 file\_size (telegram.Document attribute), 155  
 file\_size (telegram.document.Document attribute), 77

- file\_size (telegram.File attribute), 156
  - file\_size (telegram.file.File attribute), 78
  - file\_size (telegram.PhotoSize attribute), 181
  - file\_size (telegram.photosize.PhotoSize attribute), 111
  - file\_size (telegram.Sticker attribute), 183
  - file\_size (telegram.sticker.Sticker attribute), 113
  - file\_size (telegram.Video attribute), 187
  - file\_size (telegram.video.Video attribute), 117
  - file\_size (telegram.Voice attribute), 188
  - file\_size (telegram.voice.Voice attribute), 118
  - filter() (telegram.ext.BaseFilter method), 34
  - filter() (telegram.ext.filters.BaseFilter method), 18
  - filter() (telegram.ext.Filters.entity method), 34
  - filter() (telegram.ext.filters.Filters.entity method), 18
  - filter() (telegram.ext.filters.InvertedFilter method), 19
  - filter() (telegram.ext.filters.MergedFilter method), 19
  - Filters (class in telegram.ext), 34
  - Filters (class in telegram.ext.filters), 18
  - Filters.entity (class in telegram.ext), 34
  - Filters.entity (class in telegram.ext.filters), 18
  - FIND\_LOCATION (telegram.ChatAction attribute), 153
  - FIND\_LOCATION (telegram.chataction.ChatAction attribute), 73
  - first\_name (telegram.Bot attribute), 120, 126
  - first\_name (telegram.bot.Bot attribute), 40
  - first\_name (telegram.Chat attribute), 151
  - first\_name (telegram.chat.Chat attribute), 72
  - first\_name (telegram.Contact attribute), 154
  - first\_name (telegram.contact.Contact attribute), 76
  - first\_name (telegram.InlineQueryResultContact attribute), 167
  - first\_name (telegram.inlinequeryresultcontact.InlineQueryResultContact attribute), 93
  - first\_name (telegram.User attribute), 185
  - first\_name (telegram.user.User attribute), 115
  - force\_reply (telegram.ForceReply attribute), 156
  - force\_reply (telegram.forcereply.ForceReply attribute), 79
  - ForceReply (class in telegram), 156
  - ForceReply (class in telegram.forcereply), 79
  - forward() (telegram.Message method), 177
  - forward() (telegram.message.Message method), 107
  - forward\_date (telegram.Message attribute), 174
  - forward\_date (telegram.message.Message attribute), 104
  - forward\_from (telegram.Message attribute), 174
  - forward\_from (telegram.message.Message attribute), 104
  - forward\_from\_chat (telegram.Message attribute), 174
  - forward\_from\_chat (telegram.message.Message attribute), 104
  - forward\_from\_message\_id (telegram.Message attribute), 174
  - forward\_from\_message\_id (telegram.message.Message attribute), 104
  - forward\_message() (telegram.Bot method), 127
  - forward\_message() (telegram.bot.Bot method), 47
  - forwarded (telegram.ext.Filters attribute), 34
  - forwarded (telegram.ext.filters.Filters attribute), 18
  - forwardMessage() (telegram.Bot method), 126
  - forwardMessage() (telegram.bot.Bot method), 47
  - FRI (telegram.ext.jobqueue.Days attribute), 5
  - from\_user (telegram.ChosenInlineResult attribute), 153
  - from\_user (telegram.choseninlineresult.ChosenInlineResult attribute), 75
  - from\_user (telegram.InlineQuery attribute), 158
  - from\_user (telegram.inlinequery.InlineQuery attribute), 83
  - from\_user (telegram.Message attribute), 174
  - from\_user (telegram.message.Message attribute), 103
- ## G
- Game (class in telegram), 189
  - Game (class in telegram.game), 79
  - game (telegram.ext.Filters attribute), 34
  - game (telegram.ext.filters.Filters attribute), 18
  - game (telegram.Message attribute), 175
  - game (telegram.message.Message attribute), 104
  - GameHighScore (class in telegram), 190
  - GameHighScore (class in telegram.gamehighscore), 80
  - get\_administrators() (telegram.Chat method), 152
  - get\_administrators() (telegram.chat.Chat method), 73
  - get\_chat() (telegram.Bot method), 130
  - get\_chat() (telegram.bot.Bot method), 51
  - get\_chat\_administrators() (telegram.Bot method), 131
  - get\_chat\_administrators() (telegram.bot.Bot method), 51
  - get\_chat\_member() (telegram.Bot method), 131
  - get\_chat\_member() (telegram.bot.Bot method), 51
  - get\_chat\_members\_count() (telegram.Bot method), 131
  - get\_chat\_members\_count() (telegram.bot.Bot method), 52
  - get\_file() (telegram.Bot method), 132
  - get\_file() (telegram.bot.Bot method), 52
  - get\_game\_high\_scores() (telegram.Bot method), 132
  - get\_game\_high\_scores() (telegram.bot.Bot method), 52
  - get\_instance() (telegram.ext.Dispatcher class method), 23
  - get\_instance() (telegram.ext.dispatcher.Dispatcher class method), 4
  - get\_me() (telegram.Bot method), 132
  - get\_me() (telegram.bot.Bot method), 53
  - get\_member() (telegram.Chat method), 152
  - get\_member() (telegram.chat.Chat method), 73
  - get\_members\_count() (telegram.Chat method), 152
  - get\_members\_count() (telegram.chat.Chat method), 73
  - get\_profile\_photos() (telegram.User method), 186
  - get\_profile\_photos() (telegram.user.User method), 115
  - get\_updates() (telegram.Bot method), 133
  - get\_updates() (telegram.bot.Bot method), 53
  - get\_user\_profile\_photos() (telegram.Bot method), 133
  - get\_user\_profile\_photos() (telegram.bot.Bot method), 54
  - get\_webhook\_info() (telegram.Bot method), 134
  - get\_webhook\_info() (telegram.bot.Bot method), 54

- getChat() (telegram.Bot method), 127
- getChat() (telegram.bot.Bot method), 47
- getChatAdministrators() (telegram.Bot method), 127
- getChatAdministrators() (telegram.bot.Bot method), 48
- getChatMember() (telegram.Bot method), 128
- getChatMember() (telegram.bot.Bot method), 48
- getChatMembersCount() (telegram.Bot method), 128
- getChatMembersCount() (telegram.bot.Bot method), 48
- getFile() (telegram.Bot method), 128
- getFile() (telegram.bot.Bot method), 49
- getGameHighScores() (telegram.Bot method), 129
- getGameHighScores() (telegram.bot.Bot method), 49
- getMe() (telegram.Bot method), 129
- getMe() (telegram.bot.Bot method), 49
- getUpdates() (telegram.Bot method), 129
- getUpdates() (telegram.bot.Bot method), 49
- getUserProfilePhotos() (telegram.Bot method), 130
- getUserProfilePhotos() (telegram.bot.Bot method), 50
- getWebhookInfo() (telegram.Bot method), 130
- getWebhookInfo() (telegram.bot.Bot method), 51
- gif\_file\_id (telegram.InlineQueryResultCachedGif attribute), 163
- gif\_file\_id (telegram.inlinequeryresultcachedgif.InlineQueryResultCachedGif attribute), 88
- gif\_height (telegram.InlineQueryResultGif attribute), 169
- gif\_height (telegram.inlinequeryresultgif.InlineQueryResultGif attribute), 96
- gif\_url (telegram.InlineQueryResultGif attribute), 169
- gif\_url (telegram.inlinequeryresultgif.InlineQueryResultGif attribute), 96
- gif\_width (telegram.InlineQueryResultGif attribute), 169
- gif\_width (telegram.inlinequeryresultgif.InlineQueryResultGif attribute), 96
- GROUP (telegram.Chat attribute), 151
- GROUP (telegram.chat.Chat attribute), 73
- group (telegram.ext.Filters attribute), 34
- group (telegram.ext.filters.Filters attribute), 18
- group\_chat\_created (telegram.Message attribute), 175
- group\_chat\_created (telegram.message.Message attribute), 104
- groups (telegram.ext.dispatcher.Dispatcher attribute), 4
- H**
- handle\_update() (telegram.ext.CallbackQueryHandler method), 29
- handle\_update() (telegram.ext.callbackqueryhandler.CallbackQueryHandler method), 10
- handle\_update() (telegram.ext.ChosenInlineResultHandler method), 30
- handle\_update() (telegram.ext.choseninlineresulthandler.ChosenInlineResultHandler method), 11
- handle\_update() (telegram.ext.CommandHandler method), 31
- handle\_update() (telegram.ext.commandhandler.CommandHandler method), 13
- handle\_update() (telegram.ext.ConversationHandler method), 38
- handle\_update() (telegram.ext.conversationhandler.ConversationHandler method), 12
- handle\_update() (telegram.ext.Handler method), 31
- handle\_update() (telegram.ext.handler.Handler method), 9
- handle\_update() (telegram.ext.InlineQueryHandler method), 32
- handle\_update() (telegram.ext.inlinequeryhandler.InlineQueryHandler method), 14
- handle\_update() (telegram.ext.MessageHandler method), 33
- handle\_update() (telegram.ext.messagehandler.MessageHandler method), 15
- handle\_update() (telegram.ext.RegexHandler method), 35
- handle\_update() (telegram.ext.regexhandler.RegexHandler method), 20
- handle\_update() (telegram.ext.StringCommandHandler method), 36
- handle\_update() (telegram.ext.stringcommandhandler.StringCommandHandler method), 21
- handle\_update() (telegram.ext.StringRegexHandler method), 36
- handle\_update() (telegram.ext.stringregexhandler.StringRegexHandler method), 21
- handle\_update() (telegram.ext.TypeHandler method), 37
- handle\_update() (telegram.ext.typehandler.TypeHandler method), 22
- Handler (class in telegram.ext), 31
- Handler (class in telegram.ext.handler), 9
- handlers (telegram.ext.dispatcher.Dispatcher attribute), 4
- handleUpdate() (telegram.ext.ChosenInlineResultHandler method), 30
- handleUpdate() (telegram.ext.choseninlineresulthandler.ChosenInlineResultHandler method), 11
- handleUpdate() (telegram.ext.InlineQueryHandler method), 32
- handleUpdate() (telegram.ext.inlinequeryhandler.InlineQueryHandler method), 14



- handleUpdate() (telegram.ext.RegexHandler method), 35
- handleUpdate() (telegram.ext.regexhandler.RegexHandler method), 20
- handleUpdate() (telegram.ext.TypeHandler method), 37
- handleUpdate() (telegram.ext.typehandler.TypeHandler method), 22
- has\_custom\_certificate (telegram.WebhookInfo attribute), 188
- has\_custom\_certificate (telegram.webhookinfo.WebhookInfo attribute), 118
- has\_running\_threads (telegram.ext.Dispatcher attribute), 23
- has\_running\_threads (telegram.ext.dispatcher.Dispatcher attribute), 4
- HASHTAG (telegram.MessageEntity attribute), 180
- HASHTAG (telegram.messageentity.MessageEntity attribute), 110
- headers (telegram.InputFile attribute), 172
- headers (telegram.inputfile.InputFile attribute), 101
- height (telegram.PhotoSize attribute), 181
- height (telegram.photosize.PhotoSize attribute), 110
- height (telegram.Sticker attribute), 183
- height (telegram.sticker.Sticker attribute), 113
- height (telegram.Video attribute), 187
- height (telegram.video.Video attribute), 117
- hide\_url (telegram.InlineQueryResultArticle attribute), 160
- hide\_url (telegram.inlinequeryresultarticle.InlineQueryResultArticle attribute), 84
- HTML (telegram.ParseMode attribute), 181
- HTML (telegram.parsemode.ParseMode attribute), 110
- I
- id (telegram.Bot attribute), 120, 134
- id (telegram.bot.Bot attribute), 40
- id (telegram.Chat attribute), 151
- id (telegram.chat.Chat attribute), 72
- id (telegram.InlineQuery attribute), 158
- id (telegram.inlinequery.InlineQuery attribute), 83
- id (telegram.InlineQueryResult attribute), 159
- id (telegram.inlinequeryresult.InlineQueryResult attribute), 84
- id (telegram.InlineQueryResultArticle attribute), 160
- id (telegram.inlinequeryresultarticle.InlineQueryResultArticle attribute), 84
- id (telegram.InlineQueryResultAudio attribute), 161
- id (telegram.inlinequeryresultaudio.InlineQueryResultAudio attribute), 85
- id (telegram.InlineQueryResultCachedAudio attribute), 161
- id (telegram.inlinequeryresultcachedaudio.InlineQueryResultCachedAudio attribute), 86
- id (telegram.User attribute), 185
- id (telegram.user.User attribute), 115
- idle() (telegram.ext.Updater method), 27
- idle() (telegram.ext.updater.Updater method), 2
- info() (telegram.Bot method), 134
- inline\_keyboard (telegram.InlineKeyboardMarkup attribute), 158
- inline\_keyboard (telegram.inlinekeyboardmarkup.InlineKeyboardMarkup attribute), 82
- inline\_message\_id (telegram.ChosenInlineResult attribute), 153
- inline\_message\_id (telegram.choseninlineresult.ChosenInlineResult attribute), 75
- inline\_query (telegram.Update attribute), 184
- inline\_query (telegram.update.Update attribute), 114
- InlineKeyboardButton (class in telegram), 157
- InlineKeyboardButton (class in telegram.inlinekeyboardbutton), 81
- InlineKeyboardMarkup (class in telegram), 158
- InlineKeyboardMarkup (class in telegram.inlinekeyboardmarkup), 82
- InlineQuery (class in telegram), 158
- InlineQuery (class in telegram.inlinequery), 82
- InlineQueryHandler (class in telegram.ext), 32
- InlineQueryHandler (class in telegram.ext.inlinequeryhandler), 14
- InlineQueryResult (class in telegram), 159
- InlineQueryResult (class in telegram.inlinequeryresult), 83
- InlineQueryResultArticle (class in telegram), 159
- InlineQueryResultArticle (class in telegram.inlinequeryresultarticle), 84
- InlineQueryResultAudio (class in telegram), 160
- InlineQueryResultAudio (class in telegram.inlinequeryresultaudio), 85
- InlineQueryResultCachedAudio (class in telegram), 161
- InlineQueryResultCachedAudio (class in telegram.inlinequeryresultcachedaudio), 86
- InlineQueryResultCachedDocument (class in telegram), 162
- InlineQueryResultCachedDocument (class in telegram.inlinequeryresultcacheddocument), 87
- InlineQueryResultCachedGif (class in telegram), 163
- InlineQueryResultCachedGif (class in telegram.inlinequeryresultcachedgif), 88
- InlineQueryResultCachedMpeg4Gif (class in telegram), 163
- InlineQueryResultCachedMpeg4Gif (class in telegram.inlinequeryresultcachedmpeg4gif), 89
- InlineQueryResultCachedPhoto (class in telegram), 164
- InlineQueryResultCachedPhoto (class in telegram.inlinequeryresultcachedphoto), 90
- InlineQueryResultCachedSticker (class in telegram), 165
- InlineQueryResultCachedSticker (class in telegram)

[gram.inlinequeryresultcachedsticker](#)), 91  
[InlineQueryResultCachedVideo](#) (class in telegram), 165  
[InlineQueryResultCachedVideo](#) (class in telegram.inlinequeryresultcachedvideo), 91  
[InlineQueryResultCachedVoice](#) (class in telegram), 166  
[InlineQueryResultCachedVoice](#) (class in telegram.inlinequeryresultcachedvoice), 92  
[InlineQueryResultContact](#) (class in telegram), 167  
[InlineQueryResultContact](#) (class in telegram.inlinequeryresultcontact), 93  
[InlineQueryResultDocument](#) (class in telegram), 167  
[InlineQueryResultDocument](#) (class in telegram.inlinequeryresultdocument), 94  
[InlineQueryResultGame](#) (class in telegram), 172  
[InlineQueryResultGame](#) (class in telegram.inlinequeryresultgame), 95  
[InlineQueryResultGif](#) (class in telegram), 169  
[InlineQueryResultGif](#) (class in telegram.inlinequeryresultgif), 96  
[InlineQueryResultLocation](#) (class in telegram), 169  
[InlineQueryResultLocation](#) (class in telegram.inlinequeryresultlocation), 97  
[InlineQueryResultMpeg4Gif](#) (class in telegram), 170  
[InlineQueryResultMpeg4Gif](#) (class in telegram.inlinequeryresultmpeg4gif), 98  
[InlineQueryResultPhoto](#) (class in telegram), 171  
[InlineQueryResultPhoto](#) (class in telegram.inlinequeryresultphoto), 99  
[InlineQueryResultVenue](#) (class in telegram), 172  
[InlineQueryResultVenue](#) (class in telegram.inlinequeryresultvenue), 100  
[InlineQueryResultVideo](#) (class in telegram), 172  
[InlineQueryResultVideo](#) (class in telegram.inlinequeryresultvideo), 100  
[InlineQueryResultVoice](#) (class in telegram), 172  
[InlineQueryResultVoice](#) (class in telegram.inlinequeryresultvoice), 100  
[input\\_message\\_content](#) (telegram.InlineQueryResultArticle attribute), 160  
[input\\_message\\_content](#) (telegram.inlinequeryresultarticle.InlineQueryResultArticle attribute), 84  
[input\\_message\\_content](#) (telegram.InlineQueryResultAudio attribute), 161  
[input\\_message\\_content](#) (telegram.inlinequeryresultaudio.InlineQueryResultAudio attribute), 85  
[input\\_message\\_content](#) (telegram.InlineQueryResultCachedAudio attribute), 162  
[input\\_message\\_content](#) (telegram.inlinequeryresultcachedaudio.InlineQueryResultCachedAudio attribute), 86  
[input\\_message\\_content](#) (telegram.InlineQueryResultCachedDocument attribute), 162  
[input\\_message\\_content](#) (telegram.inlinequeryresultcacheddocument.InlineQueryResultCachedDocument attribute), 87  
[input\\_message\\_content](#) (telegram.InlineQueryResultCachedGif attribute), 163  
[input\\_message\\_content](#) (telegram.inlinequeryresultcachedgif.InlineQueryResultCachedGif attribute), 88  
[input\\_message\\_content](#) (telegram.InlineQueryResultCachedMpeg4Gif attribute), 164  
[input\\_message\\_content](#) (telegram.inlinequeryresultcachedmpeg4gif.InlineQueryResultCachedMpeg4Gif attribute), 89  
[input\\_message\\_content](#) (telegram.InlineQueryResultCachedPhoto attribute), 164  
[input\\_message\\_content](#) (telegram.inlinequeryresultcachedphoto.InlineQueryResultCachedPhoto attribute), 90  
[input\\_message\\_content](#) (telegram.InlineQueryResultCachedSticker attribute), 165  
[input\\_message\\_content](#) (telegram.inlinequeryresultcachedsticker.InlineQueryResultCachedSticker attribute), 91  
[input\\_message\\_content](#) (telegram.InlineQueryResultCachedVideo attribute), 166  
[input\\_message\\_content](#) (telegram.inlinequeryresultcachedvideo.InlineQueryResultCachedVideo attribute), 92  
[input\\_message\\_content](#) (telegram.InlineQueryResultCachedVoice attribute), 166  
[input\\_message\\_content](#) (telegram.inlinequeryresultcachedvoice.InlineQueryResultCachedVoice attribute), 93  
[input\\_message\\_content](#) (telegram.InlineQueryResultContact attribute), 167  
[input\\_message\\_content](#) (telegram.inlinequeryresultcontact.InlineQueryResultContact attribute), 94  
[input\\_message\\_content](#) (telegram.InlineQueryResultDocument attribute), 168  
[input\\_message\\_content](#) (telegram.inlinequeryresultdocument.InlineQueryResultDocument attribute), 95  
[input\\_message\\_content](#) (telegram.InlineQueryResultGif attribute), 169  
[input\\_message\\_content](#) (telegram.inlinequeryresultgif.InlineQueryResultGif attribute), 96

- input\_message\_content (telegram.InlineQueryResultLocation attribute), 170
  - input\_message\_content (telegram.inlinequeryresultlocation.InlineQueryResultLocation attribute), 97
  - input\_message\_content (telegram.InlineQueryResultMpeg4Gif attribute), 171
  - input\_message\_content (telegram.inlinequeryresultmpeg4gif.InlineQueryResultMpeg4Gif attribute), 98
  - input\_message\_content (telegram.InlineQueryResultPhoto attribute), 172
  - input\_message\_content (telegram.inlinequeryresultphoto.InlineQueryResultPhoto attribute), 99
  - InputContactMessageContent (class in telegram), 172
  - InputContactMessageContent (class in telegram.inputcontactmessagecontent), 101
  - InputFile (class in telegram), 172
  - InputFile (class in telegram.inputfile), 101
  - InputLocationMessageContent (class in telegram), 173
  - InputLocationMessageContent (class in telegram.inputlocationmessagecontent), 101
  - InputMessageContent (class in telegram), 173
  - InputMessageContent (class in telegram.inputmessagecontent), 102
  - InputTextMessageContent (class in telegram), 173
  - InputTextMessageContent (class in telegram.inputtextmessagecontent), 102
  - InputVenueMessageContent (class in telegram), 173
  - InputVenueMessageContent (class in telegram.inputvenuemessagecontent), 102
  - interval (telegram.ext.Job attribute), 26
  - interval (telegram.ext.jobqueue.Job attribute), 5, 6
  - interval\_seconds (telegram.ext.Job attribute), 27
  - interval\_seconds (telegram.ext.jobqueue.Job attribute), 6
  - InvalidToken, 77
  - InvertedFilter (class in telegram.ext.filters), 19
  - is\_image() (telegram.InputFile static method), 172
  - is\_image() (telegram.inputfile.InputFile static method), 101
  - is\_inputfile() (telegram.InputFile static method), 173
  - is\_inputfile() (telegram.inputfile.InputFile static method), 101
  - ITALIC (telegram.MessageEntity attribute), 180
  - ITALIC (telegram.messageentity.MessageEntity attribute), 110
- ## J
- Job (class in telegram.ext), 26
  - Job (class in telegram.ext.jobqueue), 5
  - job\_queue (telegram.ext.Job attribute), 26, 27
  - job\_queue (telegram.ext.jobqueue.Job attribute), 6
  - JobQueue (class in telegram.ext), 24
  - JobQueue (class in telegram.ext.jobqueue), 6
  - jobs() (telegram.ext.JobQueue method), 24
  - jobs() (telegram.ext.jobqueue.JobQueue method), 7
- ## K
- keyboard (telegram.ReplyKeyboardMarkup attribute), 182
  - keyboard (telegram.replykeyboardmarkup.ReplyKeyboardMarkup attribute), 112
  - KeyboardButton (class in telegram), 173
  - KeyboardButton (class in telegram.keyboardbutton), 102
  - kick\_chat\_member() (telegram.Bot method), 134
  - kick\_chat\_member() (telegram.bot.Bot method), 55
  - kick\_member() (telegram.Chat method), 152
  - kick\_member() (telegram.chat.Chat method), 73
  - kickChatMember() (telegram.Bot method), 134
  - kickChatMember() (telegram.bot.Bot method), 54
  - KICKED (telegram.ChatMember attribute), 152
  - KICKED (telegram.chatmember.ChatMember attribute), 74
- ## L
- last\_error\_date (telegram.WebhookInfo attribute), 188
  - last\_error\_date (telegram.webhookinfo.WebhookInfo attribute), 118
  - last\_error\_message (telegram.WebhookInfo attribute), 188
  - last\_error\_message (telegram.webhookinfo.WebhookInfo attribute), 118
  - last\_name (telegram.Bot attribute), 120, 135
  - last\_name (telegram.bot.Bot attribute), 40
  - last\_name (telegram.Chat attribute), 151
  - last\_name (telegram.chat.Chat attribute), 72
  - last\_name (telegram.Contact attribute), 154
  - last\_name (telegram.contact.Contact attribute), 76
  - last\_name (telegram.InlineQueryResultContact attribute), 167
  - last\_name (telegram.inlinequeryresultcontact.InlineQueryResultContact attribute), 93
  - last\_name (telegram.User attribute), 185
  - last\_name (telegram.user.User attribute), 115
  - latitude (telegram.InlineQueryResultLocation attribute), 170
  - latitude (telegram.inlinequeryresultlocation.InlineQueryResultLocation attribute), 97
  - latitude (telegram.Location attribute), 173
  - latitude (telegram.location.Location attribute), 103
  - leave() (telegram.Chat method), 152
  - leave() (telegram.chat.Chat method), 73
  - leave\_chat() (telegram.Bot method), 135
  - leave\_chat() (telegram.bot.Bot method), 55
  - leaveChat() (telegram.Bot method), 135
  - leaveChat() (telegram.bot.Bot method), 55
  - LEFT (telegram.ChatMember attribute), 152
  - LEFT (telegram.chatmember.ChatMember attribute), 74

- left\_chat\_member (telegram.Message attribute), 175
  - left\_chat\_member (telegram.message.Message attribute), 104
  - Location (class in telegram), 173
  - Location (class in telegram.location), 103
  - location (telegram.ChosenInlineResult attribute), 153
  - location (telegram.choseninlineresult.ChosenInlineResult attribute), 75
  - location (telegram.ext.Filters attribute), 34
  - location (telegram.ext.filters.Filters attribute), 18
  - location (telegram.Message attribute), 175
  - location (telegram.message.Message attribute), 104
  - log() (telegram.Bot method), 135
  - logger (telegram.ext.Dispatcher attribute), 23
  - logger (telegram.ext.dispatcher.Dispatcher attribute), 4
  - longitude (telegram.InlineQueryResultLocation attribute), 170
  - longitude (telegram.inlinequeryresultlocation.InlineQueryResultLocation attribute), 97
  - longitude (telegram.Location attribute), 173
  - longitude (telegram.location.Location attribute), 103
- M**
- m (telegram.ext.ChosenInlineResultHandler attribute), 30
  - m (telegram.ext.choseninlineresulthandler.ChosenInlineResultHandler attribute), 11
  - m (telegram.ext.InlineQueryHandler attribute), 32
  - m (telegram.ext.inlinequeryhandler.InlineQueryHandler attribute), 14
  - m (telegram.ext.RegexHandler attribute), 35
  - m (telegram.ext.regexhandler.RegexHandler attribute), 20
  - m (telegram.ext.TypeHandler attribute), 37
  - m (telegram.ext.typehandler.TypeHandler attribute), 22
  - MARKDOWN (telegram.ParseMode attribute), 181
  - MARKDOWN (telegram.parsemode.ParseMode attribute), 110
  - MAX\_CAPTION\_LENGTH (in module telegram.constants), 75
  - MAX\_FILESIZE\_DOWNLOAD (in module telegram.constants), 75
  - MAX\_FILESIZE\_UPLOAD (in module telegram.constants), 75
  - MAX\_INLINE\_QUERY\_RESULTS (in module telegram.constants), 76
  - MAX\_MESSAGE\_ENTITIES (in module telegram.constants), 76
  - MAX\_MESSAGE\_LENGTH (in module telegram.constants), 75
  - MAX\_MESSAGES\_PER\_MINUTE\_PER\_GROUP (in module telegram.constants), 76
  - MAX\_MESSAGES\_PER\_SECOND (in module telegram.constants), 76
  - MAX\_MESSAGES\_PER\_SECOND\_PER\_CHAT (in module telegram.constants), 75
  - MEMBER (telegram.ChatMember attribute), 152
  - MEMBER (telegram.chatmember.ChatMember attribute), 74
  - MENTION (telegram.MessageEntity attribute), 180
  - MENTION (telegram.messageentity.MessageEntity attribute), 110
  - MergedFilter (class in telegram.ext.filters), 19
  - Message (class in telegram), 174
  - Message (class in telegram.message), 103
  - message (telegram.Update attribute), 184
  - message (telegram.update.Update attribute), 114
  - message() (telegram.Bot method), 135
  - message\_id (telegram.Message attribute), 174
  - message\_id (telegram.message.Message attribute), 103
  - MessageEntity (class in telegram), 180
  - MessageEntity (class in telegram.messageentity), 109
  - MessageHandler (class in telegram.ext), 32
  - MessageHandler (class in telegram.ext.messagehandler), 15
  - MessageQueue (class in telegram.ext.messagequeue), 16
  - migrate\_from\_chat\_id (telegram.Message attribute), 175
  - migrate\_from\_chat\_id (telegram.message.Message attribute), 105
  - migrate\_to\_chat\_id (telegram.Message attribute), 175
  - migrate\_to\_chat\_id (telegram.message.Message attribute), 105
  - mime\_type (telegram.Audio attribute), 119
  - mime\_type (telegram.audio.Audio attribute), 39
  - mime\_type (telegram.Document attribute), 155
  - mime\_type (telegram.document.Document attribute), 77
  - mime\_type (telegram.InlineQueryResultDocument attribute), 168
  - mime\_type (telegram.inlinequeryresultdocument.InlineQueryResultDocument attribute), 95
  - mime\_type (telegram.Video attribute), 187
  - mime\_type (telegram.video.Video attribute), 117
  - mime\_type (telegram.Voice attribute), 187
  - mime\_type (telegram.voice.Voice attribute), 118
  - MON (telegram.ext.jobqueue.Days attribute), 5
  - mpeg4\_file\_id (telegram.InlineQueryResultCachedMpeg4Gif attribute), 164
  - mpeg4\_file\_id (telegram.inlinequeryresultcachedmpeg4gif.InlineQueryResultCachedMpeg4Gif attribute), 89
  - mpeg4\_height (telegram.InlineQueryResultMpeg4Gif attribute), 171
  - mpeg4\_height (telegram.inlinequeryresultmpeg4gif.InlineQueryResultMpeg4Gif attribute), 98
  - mpeg4\_url (telegram.InlineQueryResultMpeg4Gif attribute), 170
  - mpeg4\_url (telegram.inlinequeryresultmpeg4gif.InlineQueryResultMpeg4Gif attribute), 98
  - mpeg4\_width (telegram.InlineQueryResultMpeg4Gif attribute), 170
  - mpeg4\_width (telegram.inlinequeryresultmpeg4gif.InlineQueryResultMpeg4Gif attribute), 98

## N

name (telegram.Bot attribute), 120, 135  
 name (telegram.bot.Bot attribute), 40  
 name (telegram.ext.Job attribute), 26  
 name (telegram.ext.jobqueue.Job attribute), 6  
 name (telegram.User attribute), 186  
 name (telegram.user.User attribute), 116  
 NetworkError, 77  
 new\_chat\_member (telegram.Message attribute), 175  
 new\_chat\_member (telegram.message.Message attribute), 104  
 new\_chat\_photo (telegram.Message attribute), 175  
 new\_chat\_photo (telegram.message.Message attribute), 104  
 new\_chat\_title (telegram.Message attribute), 175  
 new\_chat\_title (telegram.message.Message attribute), 104

## O

offset (telegram.InlineQuery attribute), 158  
 offset (telegram.inlinequery.InlineQuery attribute), 83  
 one\_time\_keyboard (telegram.ReplyKeyboardMarkup attribute), 182  
 one\_time\_keyboard (telegram.replykeyboardmarkup.ReplyKeyboardMarkup attribute), 112

## P

parse\_entities() (telegram.Message method), 177  
 parse\_entities() (telegram.message.Message method), 107  
 parse\_entity() (telegram.Message method), 178  
 parse\_entity() (telegram.message.Message method), 107  
 parse\_text\_entities() (telegram.Game method), 190  
 parse\_text\_entities() (telegram.game.Game method), 80  
 parse\_text\_entity() (telegram.Game method), 190  
 parse\_text\_entity() (telegram.game.Game method), 80  
 ParseMode (class in telegram), 181  
 ParseMode (class in telegram.parsemode), 110  
 pending\_update\_count (telegram.WebhookInfo attribute), 188  
 pending\_update\_count (telegram.webhookinfo.WebhookInfo attribute), 118  
 per\_message (telegram.ext.conversationhandler.ConversationHandler attribute), 12  
 performer (telegram.Audio attribute), 119  
 performer (telegram.audio.Audio attribute), 39  
 performer (telegram.InlineQueryResultAudio attribute), 161  
 performer (telegram.inlinequeryresultaudio.InlineQueryResultAudio attribute), 85  
 phone\_number (telegram.Contact attribute), 154  
 phone\_number (telegram.contact.Contact attribute), 76  
 phone\_number (telegram.InlineQueryResultContact attribute), 167

phone\_number (telegram.inlinequeryresultcontact.InlineQueryResultContact attribute), 93  
 photo (telegram.ext.Filters attribute), 34  
 photo (telegram.ext.filters.Filters attribute), 18  
 photo (telegram.Game attribute), 189  
 photo (telegram.game.Game attribute), 79  
 photo (telegram.Message attribute), 175  
 photo (telegram.message.Message attribute), 104  
 photo\_file\_id (telegram.InlineQueryResultCachedPhoto attribute), 164  
 photo\_file\_id (telegram.inlinequeryresultcachedphoto.InlineQueryResultCachedPhoto attribute), 90  
 photo\_height (telegram.InlineQueryResultPhoto attribute), 171  
 photo\_height (telegram.inlinequeryresultphoto.InlineQueryResultPhoto attribute), 99  
 photo\_url (telegram.InlineQueryResultPhoto attribute), 171  
 photo\_url (telegram.inlinequeryresultphoto.InlineQueryResultPhoto attribute), 99  
 photo\_width (telegram.InlineQueryResultPhoto attribute), 171  
 photo\_width (telegram.inlinequeryresultphoto.InlineQueryResultPhoto attribute), 99  
 photos (telegram.UserProfilePhotos attribute), 186  
 photos (telegram.userprofilephotos.UserProfilePhotos attribute), 116  
 PhotoSize (class in telegram), 181  
 PhotoSize (class in telegram.photosize), 110  
 position (telegram.GameHighScore attribute), 190  
 position (telegram.gamehighscore.GameHighScore attribute), 81  
 PRE (telegram.MessageEntity attribute), 180  
 PRE (telegram.messageentity.MessageEntity attribute), 110  
 PRIVATE (telegram.Chat attribute), 151  
 PRIVATE (telegram.chat.Chat attribute), 73  
 private (telegram.ext.Filters attribute), 34  
 private (telegram.ext.filters.Filters attribute), 19  
 process\_update() (telegram.ext.Dispatcher method), 23  
 process\_update() (telegram.ext.dispatcher.Dispatcher method), 4  
 put() (telegram.ext.JobQueue method), 24  
 put() (telegram.ext.jobqueue.JobQueue method), 7

## Q

query (telegram.ChosenInlineResult attribute), 153  
 query (telegram.choseninlineresult.ChosenInlineResult attribute), 75  
 query (telegram.InlineQuery attribute), 158  
 query (telegram.inlinequery.InlineQuery attribute), 83  
 queue (telegram.ext.JobQueue attribute), 24  
 queue (telegram.ext.jobqueue.JobQueue attribute), 7  
 queuedmessage() (in module telegram.ext.messagequeue), 17

## R

RECORD\_AUDIO (telegram.ChatAction attribute),

153

RECORD\_AUDIO (telegram.chataction.ChatAction attribute), 73

RECORD\_VIDEO (telegram.ChatAction attribute), 153

RECORD\_VIDEO (telegram.chataction.ChatAction attribute), 73

RegexHandler (class in telegram.ext), 34

RegexHandler (class in telegram.ext.regexhandler), 19

remove\_error\_handler() (telegram.ext.Dispatcher method), 23

remove\_error\_handler() (telegram.ext.dispatcher.Dispatcher method), 4

remove\_handler() (telegram.ext.Dispatcher method), 23

remove\_handler() (telegram.ext.dispatcher.Dispatcher method), 4

remove\_keyboard (telegram.ReplyKeyboardRemove attribute), 182

remove\_keyboard (telegram.replykeyboardremove.ReplyKeyboardRemove attribute), 111

removed (telegram.ext.Job attribute), 27

removed (telegram.ext.jobqueue.Job attribute), 6

repeat (telegram.ext.Job attribute), 26, 27

repeat (telegram.ext.jobqueue.Job attribute), 6

reply (telegram.ext.Filters attribute), 34

reply (telegram.ext.filters.Filters attribute), 19

reply\_audio() (telegram.Message method), 178

reply\_audio() (telegram.message.Message method), 107

reply\_contact() (telegram.Message method), 178

reply\_contact() (telegram.message.Message method), 108

reply\_document() (telegram.Message method), 178

reply\_document() (telegram.message.Message method), 108

reply\_location() (telegram.Message method), 179

reply\_location() (telegram.message.Message method), 108

reply\_markup (telegram.InlineQueryResultArticle attribute), 160

reply\_markup (telegram.inlinequeryresultarticle.InlineQueryResultArticle attribute), 84

reply\_markup (telegram.InlineQueryResultAudio attribute), 161

reply\_markup (telegram.inlinequeryresultaudio.InlineQueryResultAudio attribute), 85

reply\_markup (telegram.InlineQueryResultCachedAudio attribute), 162

reply\_markup (telegram.inlinequeryresultcachedaudio.InlineQueryResultCachedAudio attribute), 86

reply\_markup (telegram.InlineQueryResultCachedDocument attribute), 162

reply\_markup (telegram.inlinequeryresultcacheddocument.InlineQueryResultCachedDocument attribute), 87

reply\_markup (telegram.InlineQueryResultCachedGif attribute), 163

reply\_markup (telegram.inlinequeryresultcachedgif.InlineQueryResultCachedGif attribute), 88

reply\_markup (telegram.InlineQueryResultCachedMpeg4Gif attribute), 164

reply\_markup (telegram.inlinequeryresultcachedmpeg4gif.InlineQueryResultCachedMpeg4Gif attribute), 89

reply\_markup (telegram.InlineQueryResultCachedPhoto attribute), 164

reply\_markup (telegram.inlinequeryresultcachedphoto.InlineQueryResultCachedPhoto attribute), 90

reply\_markup (telegram.InlineQueryResultCachedSticker attribute), 165

reply\_markup (telegram.inlinequeryresultcachedsticker.InlineQueryResultCachedSticker attribute), 91

reply\_markup (telegram.InlineQueryResultCachedVideo attribute), 166

reply\_markup (telegram.inlinequeryresultcachedvideo.InlineQueryResultCachedVideo attribute), 92

reply\_markup (telegram.InlineQueryResultCachedVoice attribute), 166

reply\_markup (telegram.inlinequeryresultcachedvoice.InlineQueryResultCachedVoice attribute), 93

reply\_markup (telegram.InlineQueryResultContact attribute), 167

reply\_markup (telegram.inlinequeryresultcontact.InlineQueryResultContact attribute), 93

reply\_markup (telegram.InlineQueryResultDocument attribute), 168

reply\_markup (telegram.inlinequeryresultdocument.InlineQueryResultDocument attribute), 95

reply\_markup (telegram.InlineQueryResultGif attribute), 169

reply\_markup (telegram.inlinequeryresultgif.InlineQueryResultGif attribute), 96

reply\_markup (telegram.InlineQueryResultLocation attribute), 170

reply\_markup (telegram.inlinequeryresultlocation.InlineQueryResultLocation attribute), 97

reply\_markup (telegram.InlineQueryResultMpeg4Gif attribute), 171

reply\_markup (telegram.inlinequeryresultmpeg4gif.InlineQueryResultMpeg4Gif attribute), 98

reply\_markup (telegram.InlineQueryResultPhoto attribute), 172

reply\_markup (telegram.inlinequeryresultphoto.InlineQueryResultPhoto attribute), 99

reply\_photo() (telegram.Message method), 179

reply\_photo() (telegram.message.Message method), 108

reply\_sticker() (telegram.Message method), 179

reply\_sticker() (telegram.message.Message method), 108

reply\_text() (telegram.Message method), 179

reply\_text() (telegram.message.Message method), 108

reply\_to\_message (telegram.InlineQueryResultCachedMessage attribute), 174

reply\_to\_message (telegram.message.Message attribute), 104

- reply\_venue() (telegram.Message method), 179
- reply\_venue() (telegram.message.Message method), 108
- reply\_video() (telegram.Message method), 179
- reply\_video() (telegram.message.Message method), 109
- reply\_voice() (telegram.Message method), 180
- reply\_voice() (telegram.message.Message method), 109
- ReplyKeyboardHide (class in telegram.replykeyboardremove), 111
- ReplyKeyboardMarkup (class in telegram), 182
- ReplyKeyboardMarkup (class in telegram.replykeyboardmarkup), 112
- ReplyKeyboardRemove (class in telegram), 182
- ReplyKeyboardRemove (class in telegram.replykeyboardremove), 111
- ReplyMarkup (class in telegram), 183
- ReplyMarkup (class in telegram.replymarkup), 113
- request (telegram.Bot attribute), 135
- resize\_keyboard (telegram.ReplyKeyboardMarkup attribute), 182
- resize\_keyboard (telegram.replykeyboardmarkup.ReplyKeyboardMarkup attribute), 112
- result\_id (telegram.ChosenInlineResult attribute), 153
- result\_id (telegram.choseninlineresult.ChosenInlineResults attribute), 75
- RetryAfter, 77
- run() (telegram.ext.Job method), 27
- run() (telegram.ext.jobqueue.Job method), 6
- run() (telegram.ext.messagequeue.DelayQueue method), 16
- run\_async() (in module telegram.ext.dispatcher), 5
- run\_async() (telegram.ext.Dispatcher method), 23
- run\_async() (telegram.ext.dispatcher.Dispatcher method), 5
- run\_daily() (telegram.ext.JobQueue method), 24
- run\_daily() (telegram.ext.jobqueue.JobQueue method), 7
- run\_once() (telegram.ext.JobQueue method), 24
- run\_once() (telegram.ext.jobqueue.JobQueue method), 7
- run\_repeating() (telegram.ext.JobQueue method), 25
- run\_repeating() (telegram.ext.jobqueue.JobQueue method), 8
- S**
- SAT (telegram.ext.jobqueue.Days attribute), 5
- schedule\_removal() (telegram.ext.Job method), 27
- schedule\_removal() (telegram.ext.jobqueue.Job method), 6
- score (telegram.GameHighScore attribute), 190
- score (telegram.gamehighscore.GameHighScore attribute), 81
- selective (telegram.ForceReply attribute), 156
- selective (telegram.forcereply.ForceReply attribute), 79
- selective (telegram.ReplyKeyboardMarkup attribute), 182
- selective (telegram.replykeyboardmarkup.ReplyKeyboardMarkup attribute), 112
- selective (telegram.ReplyKeyboardRemove attribute), 182
- selective (telegram.replykeyboardremove.ReplyKeyboardRemove attribute), 111
- send\_action() (telegram.Chat method), 152
- send\_action() (telegram.chat.Chat method), 73
- send\_audio() (telegram.Bot method), 141
- send\_audio() (telegram.bot.Bot method), 62
- send\_chat\_action() (telegram.Bot method), 142
- send\_chat\_action() (telegram.bot.Bot method), 62
- send\_contact() (telegram.Bot method), 143
- send\_contact() (telegram.bot.Bot method), 63
- send\_document() (telegram.Bot method), 143
- send\_document() (telegram.bot.Bot method), 63
- send\_game() (telegram.Bot method), 144
- send\_game() (telegram.bot.Bot method), 64
- send\_location() (telegram.Bot method), 144
- send\_location() (telegram.bot.Bot method), 64
- send\_message() (telegram.Bot method), 145
- send\_message() (telegram.bot.Bot method), 65
- send\_photo() (telegram.Bot method), 145
- send\_photo() (telegram.bot.Bot method), 65
- send\_sticker() (telegram.Bot method), 146
- send\_sticker() (telegram.bot.Bot method), 66
- send\_venue() (telegram.Bot method), 146
- send\_venue() (telegram.bot.Bot method), 66
- send\_video() (telegram.Bot method), 147
- send\_video() (telegram.bot.Bot method), 67
- send\_voice() (telegram.Bot method), 147
- send\_voice() (telegram.bot.Bot method), 67
- sendAudio() (telegram.Bot method), 135
- sendAudio() (telegram.bot.Bot method), 55
- sendChatAction() (telegram.Bot method), 136
- sendChatAction() (telegram.bot.Bot method), 56
- sendContact() (telegram.Bot method), 136
- sendContact() (telegram.bot.Bot method), 57
- sendDocument() (telegram.Bot method), 137
- sendDocument() (telegram.bot.Bot method), 57
- sendGame() (telegram.Bot method), 137
- sendGame() (telegram.bot.Bot method), 58
- sendLocation() (telegram.Bot method), 138
- sendLocation() (telegram.bot.Bot method), 58
- sendMessage() (telegram.Bot method), 138
- sendMessage() (telegram.bot.Bot method), 59
- sendPhoto() (telegram.Bot method), 139
- sendPhoto() (telegram.bot.Bot method), 59
- sendSticker() (telegram.Bot method), 139
- sendSticker() (telegram.bot.Bot method), 60
- sendVenue() (telegram.Bot method), 140
- sendVenue() (telegram.bot.Bot method), 60
- sendVideo() (telegram.Bot method), 140
- sendVideo() (telegram.bot.Bot method), 61
- sendVoice() (telegram.Bot method), 141
- sendVoice() (telegram.bot.Bot method), 61

- set\_game\_score() (telegram.Bot method), 149
  - set\_game\_score() (telegram.bot.Bot method), 69
  - set\_webhook() (telegram.Bot method), 149
  - set\_webhook() (telegram.bot.Bot method), 70
  - setGameScore() (telegram.Bot method), 148
  - setGameScore() (telegram.bot.Bot method), 68
  - setWebhook() (telegram.Bot method), 148
  - setWebhook() (telegram.bot.Bot method), 69
  - signal\_handler() (telegram.ext.Updater method), 27
  - signal\_handler() (telegram.ext.updater.Updater method), 2
  - start() (telegram.ext.Dispatcher method), 24
  - start() (telegram.ext.dispatcher.Dispatcher method), 5
  - start() (telegram.ext.JobQueue method), 26
  - start() (telegram.ext.jobqueue.JobQueue method), 8
  - start() (telegram.ext.messagequeue.MessageQueue method), 17
  - start\_polling() (telegram.ext.Updater method), 27
  - start\_polling() (telegram.ext.updater.Updater method), 2
  - start\_webhook() (telegram.ext.Updater method), 28
  - start\_webhook() (telegram.ext.updater.Updater method), 3
  - states (telegram.ext.conversationhandler.ConversationHandler attribute), 12
  - status (telegram.ChatMember attribute), 152
  - status (telegram.chatmember.ChatMember attribute), 74
  - status\_update (telegram.ext.Filters attribute), 34
  - status\_update (telegram.ext.filters.Filters attribute), 19
  - Sticker (class in telegram), 183
  - Sticker (class in telegram.sticker), 113
  - sticker (telegram.ext.Filters attribute), 34
  - sticker (telegram.ext.filters.Filters attribute), 19
  - sticker (telegram.Message attribute), 175
  - sticker (telegram.message.Message attribute), 104
  - sticker\_file\_id (telegram.InlineQueryResultCachedSticker attribute), 165
  - sticker\_file\_id (telegram.inlinequeryresultcachedsticker.InlineQueryResultCachedSticker attribute), 91
  - stop() (telegram.ext.Dispatcher method), 24
  - stop() (telegram.ext.dispatcher.Dispatcher method), 5
  - stop() (telegram.ext.JobQueue method), 26
  - stop() (telegram.ext.jobqueue.JobQueue method), 8
  - stop() (telegram.ext.messagequeue.DelayQueue method), 16
  - stop() (telegram.ext.messagequeue.MessageQueue method), 17
  - stop() (telegram.ext.Updater method), 28
  - stop() (telegram.ext.updater.Updater method), 3
  - StringCommandHandler (class in telegram.ext), 35
  - StringCommandHandler (class in telegram.ext.stringcommandhandler), 20
  - StringRegexHandler (class in telegram.ext), 36
  - StringRegexHandler (class in telegram.ext.stringregexhandler), 21
  - SUN (telegram.ext.jobqueue.Days attribute), 5
  - SUPERGROUP (telegram.Chat attribute), 151
  - SUPERGROUP (telegram.chat.Chat attribute), 73
  - supergroup\_chat\_created (telegram.Message attribute), 175
  - supergroup\_chat\_created (telegram.message.Message attribute), 105
  - SUPPORTED\_WEBHOOK\_PORTS (in module telegram.constants), 75
  - switch\_inline\_query (telegram.InlineKeyboardButton attribute), 157
  - switch\_inline\_query (telegram.inlinekeyboardbutton.InlineKeyboardButton attribute), 81
  - switch\_inline\_query\_current\_chat (telegram.InlineKeyboardButton attribute), 157
  - switch\_inline\_query\_current\_chat (telegram.inlinekeyboardbutton.InlineKeyboardButton attribute), 81
- ## T
- telegram (module), 119
  - telegram.animation (module), 38
  - telegram.audio (module), 39
  - telegram.base (module), 39
  - telegram.bot (module), 40
  - telegram.callbackgame (module), 71
  - telegram.callbackquery (module), 71
  - telegram.chat (module), 72
  - telegram.chataction (module), 73
  - telegram.chatmember (module), 74
  - telegram.choseninlineresult (module), 74
  - telegram.constants (module), 75
  - telegram.contact (module), 76
  - telegram.contrib (module), 1
  - telegram.contrib.botan (module), 1
  - telegram.document (module), 77
  - telegram.error (module), 77
  - telegram.ext (module), 22
  - telegram.ext.callbackqueryhandler (module), 10
  - telegram.ext.choseninlineresulthandler (module), 10
  - telegram.ext.commandhandler (module), 13
  - telegram.ext.conversationhandler (module), 11
  - telegram.ext.dispatcher (module), 3
  - telegram.ext.filters (module), 18
  - telegram.ext.handler (module), 9
  - telegram.ext.inlinequeryhandler (module), 14
  - telegram.ext.jobqueue (module), 5
  - telegram.ext.messagehandler (module), 15
  - telegram.ext.messagequeue (module), 15
  - telegram.ext.regexhandler (module), 19
  - telegram.ext.stringcommandhandler (module), 20
  - telegram.ext.stringregexhandler (module), 21
  - telegram.ext.typehandler (module), 22
  - telegram.ext.updater (module), 1
  - telegram.file (module), 78
  - telegramforcereply (module), 79
  - telegram.game (module), 79
  - telegram.gamehighscore (module), 80



- telegram.inlinekeyboardbutton (module), 81
- telegram.inlinekeyboardmarkup (module), 82
- telegram.inlinequery (module), 82
- telegram.inlinequeryresult (module), 83
- telegram.inlinequeryresultarticle (module), 84
- telegram.inlinequeryresultaudio (module), 85
- telegram.inlinequeryresultcachedaudio (module), 86
- telegram.inlinequeryresultcacheddocument (module), 87
- telegram.inlinequeryresultcachedgif (module), 88
- telegram.inlinequeryresultcachedmpeg4gif (module), 89
- telegram.inlinequeryresultcachedphoto (module), 90
- telegram.inlinequeryresultcachedsticker (module), 91
- telegram.inlinequeryresultcachedvideo (module), 91
- telegram.inlinequeryresultcachedvoice (module), 92
- telegram.inlinequeryresultcontact (module), 93
- telegram.inlinequeryresultdocument (module), 94
- telegram.inlinequeryresultgame (module), 95
- telegram.inlinequeryresultgif (module), 96
- telegram.inlinequeryresultlocation (module), 97
- telegram.inlinequeryresultmpeg4gif (module), 98
- telegram.inlinequeryresultphoto (module), 99
- telegram.inlinequeryresultvenue (module), 100
- telegram.inlinequeryresultvideo (module), 100
- telegram.inlinequeryresultvoice (module), 100
- telegram.inputcontactmessagecontent (module), 101
- telegram.inputfile (module), 101
- telegram.inputlocationmessagecontent (module), 101
- telegram.inputmessagecontent (module), 102
- telegram.inputtextmessagecontent (module), 102
- telegram.inputvenuemessagecontent (module), 102
- telegram.keyboardbutton (module), 102
- telegram.location (module), 103
- telegram.message (module), 103
- telegram.messageentity (module), 109
- telegram.parsemode (module), 110
- telegram.photosize (module), 110
- telegram.replykeyboardmarkup (module), 112
- telegram.replykeyboardremove (module), 111
- telegram.replymarkup (module), 113
- telegram.sticker (module), 113
- telegram.update (module), 114
- telegram.user (module), 115
- telegram.userprofilephotos (module), 116
- telegram.venue (module), 116
- telegram.video (module), 117
- telegram.voice (module), 117
- telegram.webhookinfo (module), 118
- TelegramError, 77, 183
- TelegramObject (class in telegram), 184
- TelegramObject (class in telegram.base), 39
- text (telegram.ext.Filters attribute), 34
- text (telegram.ext.filters.Filters attribute), 19
- text (telegram.InlineKeyboardButton attribute), 157
- text (telegram.inlinekeyboardbutton.InlineKeyboardButton attribute), 81
- text (telegram.Message attribute), 174
- text (telegram.message.Message attribute), 104
- text\_html (telegram.Message attribute), 180
- text\_html (telegram.message.Message attribute), 109
- TEXT\_LINK (telegram.MessageEntity attribute), 180
- TEXT\_LINK (telegram.messageentity.MessageEntity attribute), 110
- text\_markdown (telegram.Message attribute), 180
- text\_markdown (telegram.message.Message attribute), 109
- TEXT\_MENTION (telegram.MessageEntity attribute), 181
- TEXT\_MENTION (telegram.messageentity.MessageEntity attribute), 110
- THU (telegram.ext.jobqueue.Days attribute), 5
- thumb (telegram.Document attribute), 155
- thumb (telegram.document.Document attribute), 77
- thumb (telegram.Sticker attribute), 183
- thumb (telegram.sticker.Sticker attribute), 113
- thumb (telegram.Video attribute), 187
- thumb (telegram.video.Video attribute), 117
- thumb\_height (telegram.InlineQueryResultArticle attribute), 160
- thumb\_height (telegram.inlinequeryresultarticle.InlineQueryResultArticle attribute), 84
- thumb\_height (telegram.InlineQueryResultContact attribute), 167
- thumb\_height (telegram.inlinequeryresultcontact.InlineQueryResultContact attribute), 94
- thumb\_height (telegram.InlineQueryResultDocument attribute), 168
- thumb\_height (telegram.inlinequeryresultdocument.InlineQueryResultDocument attribute), 95
- thumb\_height (telegram.InlineQueryResultLocation attribute), 170
- thumb\_height (telegram.inlinequeryresultlocation.InlineQueryResultLocation attribute), 97
- thumb\_url (telegram.InlineQueryResultArticle attribute), 160
- thumb\_url (telegram.inlinequeryresultarticle.InlineQueryResultArticle attribute), 84
- thumb\_url (telegram.InlineQueryResultContact attribute), 167
- thumb\_url (telegram.inlinequeryresultcontact.InlineQueryResultContact attribute), 94
- thumb\_url (telegram.InlineQueryResultDocument attribute), 168
- thumb\_url (telegram.inlinequeryresultdocument.InlineQueryResultDocument attribute), 95
- thumb\_url (telegram.InlineQueryResultGif attribute), 169
- thumb\_url (telegram.inlinequeryresultgif.InlineQueryResultGif attribute), 96
- thumb\_url (telegram.InlineQueryResultLocation attribute), 170
- thumb\_url (telegram.inlinequeryresultlocation.InlineQueryResultLocation attribute), 97
- thumb\_url (telegram.InlineQueryResultMpeg4Gif attribute), 174

tribute), 170

thumb\_url (telegram.inlinequeryresultmpeg4gif.InlineQueryResultMpeg4Gif attribute), 98

thumb\_url (telegram.InlineQueryResultPhoto attribute), 171

thumb\_url (telegram.inlinequeryresultphoto.InlineQueryResultPhoto attribute), 99

thumb\_width (telegram.InlineQueryResultArticle attribute), 160

thumb\_width (telegram.inlinequeryresultarticle.InlineQueryResultArticle attribute), 84

thumb\_width (telegram.InlineQueryResultContact attribute), 167

thumb\_width (telegram.inlinequeryresultcontact.InlineQueryResultContact attribute), 94

thumb\_width (telegram.InlineQueryResultDocument attribute), 168

thumb\_width (telegram.inlinequeryresultdocument.InlineQueryResultDocument attribute), 95

thumb\_width (telegram.InlineQueryResultLocation attribute), 170

thumb\_width (telegram.inlinequeryresultlocation.InlineQueryResultLocation attribute), 97

thumb\_width (telegram.InlineQueryResultMpeg4Gif attribute), 98

thumb\_width (telegram.inlinequeryresultmpeg4gif.InlineQueryResultMpeg4Gif attribute), 98

tick() (telegram.ext.JobQueue method), 26

tick() (telegram.ext.jobqueue.JobQueue method), 8

timed\_out\_behavior (telegram.ext.conversationhandler.ConversationHandler attribute), 12

TimedOut, 78

title (telegram.Audio attribute), 119

title (telegram.audio.Audio attribute), 39

title (telegram.Chat attribute), 151

title (telegram.chat.Chat attribute), 72

title (telegram.Game attribute), 189

title (telegram.game.Game attribute), 79

title (telegram.InlineQueryResultArticle attribute), 160

title (telegram.inlinequeryresultarticle.InlineQueryResultArticle attribute), 84

title (telegram.InlineQueryResultAudio attribute), 161

title (telegram.inlinequeryresultaudio.InlineQueryResultAudio attribute), 85

title (telegram.InlineQueryResultCachedDocument attribute), 162

title (telegram.inlinequeryresultcacheddocument.InlineQueryResultCachedDocument attribute), 87

title (telegram.InlineQueryResultCachedGif attribute), 163

title (telegram.inlinequeryresultcachedgif.InlineQueryResultCachedGif attribute), 88

title (telegram.InlineQueryResultCachedMpeg4Gif attribute), 164

title (telegram.inlinequeryresultcachedmpeg4gif.InlineQueryResultCachedMpeg4Gif attribute), 89

title (telegram.InlineQueryResultCachedPhoto attribute), 164

title (telegram.inlinequeryresultcachedphoto.InlineQueryResultCachedPhoto attribute), 90

title (telegram.InlineQueryResultCachedVideo attribute), 165

title (telegram.inlinequeryresultcachedvideo.InlineQueryResultCachedVideo attribute), 92

title (telegram.InlineQueryResultCachedVoice attribute), 166

title (telegram.inlinequeryresultcachedvoice.InlineQueryResultCachedVoice attribute), 93

title (telegram.InlineQueryResultDocument attribute), 168

title (telegram.inlinequeryresultdocument.InlineQueryResultDocument attribute), 95

title (telegram.InlineQueryResultGif attribute), 169

title (telegram.inlinequeryresultgif.InlineQueryResultGif attribute), 96

title (telegram.InlineQueryResultLocation attribute), 170

title (telegram.inlinequeryresultlocation.InlineQueryResultLocation attribute), 97

title (telegram.InlineQueryResultMpeg4Gif attribute), 171

title (telegram.inlinequeryresultmpeg4gif.InlineQueryResultMpeg4Gif attribute), 98

title (telegram.InlineQueryResultPhoto attribute), 171

title (telegram.inlinequeryresultphoto.InlineQueryResultPhoto attribute), 99

to\_dict() (telegram.base.TelegramObject method), 40

to\_dict() (telegram.Bot method), 150

to\_dict() (telegram.CallbackQuery method), 154

to\_dict() (telegram.callbackquery.CallbackQuery method), 72

to\_dict() (telegram.ChosenInlineResult method), 153

to\_dict() (telegram.choseninlineresult.ChosenInlineResult method), 75

to\_dict() (telegram.Game method), 190

to\_dict() (telegram.game.Game method), 80

to\_dict() (telegram.InlineKeyboardMarkup method), 158

to\_dict() (telegram.inlinekeyboardmarkup.InlineKeyboardMarkup method), 82

to\_dict() (telegram.InlineQuery method), 159

to\_dict() (telegram.inlinequery.InlineQuery method), 83

to\_dict() (telegram.Message method), 180

to\_dict() (telegram.message.Message method), 109

to\_dict() (telegram.ReplyKeyboardMarkup method), 183

to\_dict() (telegram.replykeyboardmarkup.ReplyKeyboardMarkup method), 112

to\_dict() (telegram.TelegramObject method), 184

to\_dict() (telegram.UserProfilePhotos method), 186

to\_dict() (telegram.userprofilephotos.UserProfilePhotos method), 186

to\_form() (telegram.InputFile method), 173

to\_form() (telegram.inputfile.InputFile method), 101

to\_json() (telegram.base.TelegramObject method), 40

to\_json() (telegram.TelegramObject method), 184

token (telegram.contrib.botan.Botan attribute), 1

total\_count (telegram.UserProfilePhotos attribute), 186

- total\_count (telegram.userprofilephotos.UserProfilePhotosURL (telegram.MessageEntity attribute), 181 attribute), 116  
 URL (telegram.messageentity.MessageEntity attribute), 110  
 track() (telegram.contrib.botan.Botan method), 1  
 TUE (telegram.ext.jobqueue.Days attribute), 5  
 type (telegram.Chat attribute), 151  
 type (telegram.chat.Chat attribute), 72  
 type (telegram.InlineQueryResult attribute), 159  
 type (telegram.inlinequeryresult.InlineQueryResult attribute), 83  
 type (telegram.User attribute), 185  
 type (telegram.user.User attribute), 115  
 TypeHandler (class in telegram.ext), 36  
 TypeHandler (class in telegram.ext.typehandler), 22  
 TYPING (telegram.ChatAction attribute), 153  
 TYPING (telegram.chataction.ChatAction attribute), 73
- ## U
- Unauthorized, 78  
 unban\_chat\_member() (telegram.Bot method), 150  
 unban\_chat\_member() (telegram.bot.Bot method), 71  
 unban\_member() (telegram.Chat method), 152  
 unban\_member() (telegram.chat.Chat method), 73  
 unbanChatMember() (telegram.Bot method), 150  
 unbanChatMember() (telegram.bot.Bot method), 70  
 Update (class in telegram), 184  
 Update (class in telegram.update), 114  
 update\_id (telegram.Update attribute), 184  
 update\_id (telegram.update.Update attribute), 114  
 update\_state() (telegram.ext.ConversationHandler method), 38  
 update\_state() (telegram.ext.conversationhandler.ConversationHandler method), 12  
 Updater (class in telegram.ext), 27  
 Updater (class in telegram.ext.updater), 1  
 UPLOAD\_AUDIO (telegram.ChatAction attribute), 153  
 UPLOAD\_AUDIO (telegram.chataction.ChatAction attribute), 73  
 UPLOAD\_DOCUMENT (telegram.ChatAction attribute), 153  
 UPLOAD\_DOCUMENT (telegram.chataction.ChatAction attribute), 74  
 UPLOAD\_PHOTO (telegram.ChatAction attribute), 153  
 UPLOAD\_PHOTO (telegram.chataction.ChatAction attribute), 74  
 UPLOAD\_VIDEO (telegram.ChatAction attribute), 153  
 UPLOAD\_VIDEO (telegram.chataction.ChatAction attribute), 74  
 url (telegram.InlineKeyboardButton attribute), 157  
 url (telegram.inlinekeyboardbutton.InlineKeyboardButton attribute), 81  
 url (telegram.InlineQueryResultArticle attribute), 160  
 url (telegram.inlinequeryresultarticle.InlineQueryResultArticle attribute), 84  
 url (telegram.WebhookInfo attribute), 188  
 url (telegram.webhookinfo.WebhookInfo attribute), 118  
 url\_template (telegram.contrib.botan.Botan attribute), 1  
 User (class in telegram), 185  
 User (class in telegram.user), 115  
 user (telegram.ChatMember attribute), 152  
 user (telegram.chatmember.ChatMember attribute), 74  
 user (telegram.GameHighScore attribute), 190  
 user (telegram.gamehighscore.GameHighScore attribute), 81  
 user\_data (telegram.ext.dispatcher.Dispatcher attribute), 5  
 user\_id (telegram.Contact attribute), 154  
 user\_id (telegram.contact.Contact attribute), 76  
 username (telegram.Bot attribute), 120, 151  
 username (telegram.bot.Bot attribute), 40  
 username (telegram.Chat attribute), 151  
 username (telegram.chat.Chat attribute), 72  
 username (telegram.User attribute), 185  
 username (telegram.user.User attribute), 115  
 UserProfilePhotos (class in telegram), 186  
 UserProfilePhotos (class in telegram.userprofilephotos), 116
- ## V
- Venue (class in telegram), 186  
 Venue (class in telegram.venue), 116  
 venue (telegram.ext.Filters attribute), 34  
 venue (telegram.ext.filters.Filters attribute), 19  
 Video (class in telegram), 187  
 Video (class in telegram.video), 117  
 video (telegram.ext.Filters attribute), 34  
 video (telegram.ext.filters.Filters attribute), 19  
 video (telegram.Message attribute), 175  
 video (telegram.message.Message attribute), 104  
 video\_file\_id (telegram.InlineQueryResultCachedVideo attribute), 165  
 video\_file\_id (telegram.inlinequeryresultcachedvideo.InlineQueryResultCachedVideo attribute), 92  
 Voice (class in telegram), 187  
 Voice (class in telegram.voice), 117  
 voice (telegram.ext.Filters attribute), 34  
 voice (telegram.ext.filters.Filters attribute), 19  
 voice (telegram.Message attribute), 175  
 voice (telegram.message.Message attribute), 104  
 voice\_file\_id (telegram.InlineQueryResultCachedVoice attribute), 166  
 voice\_file\_id (telegram.inlinequeryresultcachedvoice.InlineQueryResultCachedVoice attribute), 92
- ## W
- WebhookInfo (class in telegram), 188  
 WebhookInfo (class in telegram.webhookinfo), 118  
 WED (telegram.ext.jobqueue.Days attribute), 5

width (telegram.PhotoSize attribute), 181  
width (telegram.photosize.PhotoSize attribute), 110  
width (telegram.Sticker attribute), 183  
width (telegram.sticker.Sticker attribute), 113  
width (telegram.Video attribute), 187  
width (telegram.video.Video attribute), 117