Python Telegram Bot Documentation

Release 6.0.0

Leandro Toledo

May 19, 2017

Contents

CHAPTER 1

telegram package

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).

ReturnsThe update queue that can be filled from the main thread

Return type Queue

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

Starts a small http server to listen for updates via webhook. If cert and key are not provided, the webhook will be started directly on http://listen:port/url_path, so SSL can be handled by another application. Else, the webhook will be started on https://listen:port/url_path

Parameters

- **listen** (*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

ReturnsThe 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.

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

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.

ReturnsDispatcher

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.

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/kwargs) 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*.

ReturnsPromise

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 typefunction

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 it's `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 it's `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 it's `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*, *allow_edited=False*, *pass_update_queue=False*, *pass_job_queue=False*, *pass_user_data=False*, *pass_chat_data=False*, *message_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*)

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.

Parameters **timeout** (float) – indicates maximum time to wait for processor to stop and its thread to exit. If timeout exceeds and processor has not stopped, method silently returns. *is_alive* could be used afterwards to check the actual status. 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

ReturnsNone

stop(*timeout=None*)
Used to gently stop processor and shutdown its thread.

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

ReturnsNone

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 outbond 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.

ReturnsEither 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*.

Parametersentity_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 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*)

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.

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

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.

ReturnsDispatcher

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/kwargs) 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*.

ReturnsPromise

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: 5.2prevent_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 it's 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__

ReturnsThe 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 it's `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 it's `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__`

ReturnsThe 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).

ReturnsThe update queue that can be filled from the main thread

Return type Queue

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

Starts a small http server to listen for updates via webhook. If cert and key are not provided, the webhook will be started directly on http://listen:port/url_path, so SSL can be handled by another application. Else, the webhook will be started on https://listen:port/url_path

Parameters

- **listen** (*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

ReturnsThe 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*, *message_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*.

Parametersentity_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*, *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*)

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 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.
- **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.

ReturnsOn 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.

ReturnsOn 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.

ReturnsOn 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.

Parameters **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 isreturned.

Return typelist[*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.

Parameters**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).

Returnsclass: *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.

ReturnsOn 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.

ReturnsA list of chat member objects.

Return typelist[*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 hisneighbors in a game.

Return typelist[*telegram.GameHighScore*]

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

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

Parameters **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 thecredentials 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.

Returnslist[*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 isreturned.

Return typelist[*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.

Parameters**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).

Returnsclass: *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.

ReturnsOn success, *True* is returned.

Return typebool

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.

ReturnsOn success, *True* is returned.

Return typebool

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.

ReturnsOn success, *True* is returned.

Return typebool

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.

ReturnsOn success, *True* is returned.

Return typebool

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.

ReturnsOn 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.

ReturnsOn 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.

ReturnsOn 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.

ReturnsOn 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.

ReturnsOn 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.

ReturnsOn 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 themessage wasn't sent by the bot, True.

Return typetelegram.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.

ReturnsOn success, *True* is returned.

Return typebool

Raisestelegram.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 themessage 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.

ReturnsOn success, *True* is returned.

Return typebool

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*, *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

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.

telegram.forcereply module

This module contains an object that represents a Telegram ForceReply.

class `telegram.forcereply.`**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*

telegram.game module

This module contains an object that represents a Telegram Game.

class `telegram.game.`**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 thetext that belongs to them, calculated based on UTF-16 codepoints.

Return typedict[*telegram.MessageEntity*, str]

parse_text_entity(*entity*)

Returns the text from a given *telegram.MessageEntity*.

Note: This method is present because Telegram calculates the offset and length in UTF-16 codepoint pairs, which some versions of Python don't handle automatically. (That is, you can't just slice Message.text with the offset and length.)

Parameterentity (telegram.MessageEntity) – The entity to extract the text from. It must be an entity that belongs to this message.

ReturnsThe text of the given entity

Return typestr

to_dict()

Returns

Return typedict

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 typedict

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: 4.0message_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*, *audio_file_id*, *caption=None*, *reply_markup=None*, *input_message_content=N* ***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.input_message_content`]) –
- ****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*, *title*, *document_file_id*, *description=None*, *caption=None*, *reply_markup=No*
input_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*, *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*)

telegram.inlinequeryresultcachedmpeg4gif module

This module contains the classes that represent Telegram InlineQueryResultMpeg4Gif

class telegram.inlinequeryresultcachedmpeg4gif.**InlineQueryResultCachedMpeg4Gif**(*id*, *mpeg4_file_id*, *title=None*, *caption=None*, *reply_markup=No*, *input_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*, *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*)

telegram.inlinequeryresultcachedsticker module

This module contains the classes that represent Telegram InlineQueryResultCachedSticker

class telegram.inlinequeryresultcachedsticker.**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*)

telegram.inlinequeryresultcachedvideo module

This module contains the classes that represent Telegram InlineQueryResultCachedVideo

class telegram.inlinequeryresultcachedvideo.**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*)

telegram.inlinequeryresultcachedvoice module

This module contains the classes that represent Telegram InlineQueryResultCachedVoice

class telegram.inlinequeryresultcachedvoice.**InlineQueryResultCachedVoice**(*id*, *voice_file_id*, *title*, *caption=None*, *reply_markup=None*, *input_message_content=N* ***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*, *reply_markup=None*, *input_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=None*, *reply_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*)

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.

Parameters **stream** (*str*) – A str representing the content of a file.

Returns The str mimetype of an image.

Return type str

static is_inputfile(*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

telegram.inputlocationmessagecontent module

This module contains the classes that represent Telegram InputLocationMessageContent

class telegram.inputlocationmessagecontent.**InputLocationMessageContent**(*latitude*, *longitude*, ***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: 4.0new_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)
```

ReturnsOn success, *True* is returned.

Return typebool

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)
```

ReturnsOn 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.

Parameters**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 thetext that belongs to them, calculated based on UTF-16 codepoints.

Return typedict[*telegram.MessageEntity*, `str`]

parse_entity(*entity*)
Returns the text from a given *telegram.MessageEntity*.

Note: This method is present because Telegram calculates the offset and length in UTF-16 codepoint pairs, which some versions of Python don't handle automatically. (That is, you can't just slice `Message.text` with the offset and length.)

Parameters**entity** (`telegram.MessageEntity`) – The entity to extract the text from. It must be an entity that belongs to this message.

ReturnsThe text of the given entity

Return typestr

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.

ReturnsOn success, instance representing the message posted.

Return typetelegram.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.

ReturnsOn success, instance representing the message posted.

Return typetelegram.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.

ReturnsOn success, instance representing the message posted.

Return typetelegram.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.

ReturnsOn success, instance representing the message posted.

Return typetelegram.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.

ReturnsOn success, instance representing the message posted.

Return typetelegram.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.

ReturnsOn success, instance representing the message posted.

Return typetelegram.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 Argumentsquote (*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.

ReturnsOn 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 Argumentsquote (*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.

ReturnsOn 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 Argumentsquote (*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.

ReturnsOn 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.

Returnsstr

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.

Returnsstr

to_dict()

Returns

Return typedict

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_me

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 type List<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) –

Returnstelegram.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 typedict

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.

ReturnsOn success, *True* is returned.

Return typebool

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.

ReturnsOn success, *True* is returned.

Return typebool

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.

ReturnsOn success, *True* is returned.

Return typebool

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.

ReturnsOn success, *True* is returned.

Return typebool

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.

ReturnsOn success, *True* is returned.

Return typebool

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.

ReturnsOn success, *True* is returned.

Return typebool

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.

ReturnsOn 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*

first_name

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.

ReturnsOn 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 hisneighbors in a game.

Return typelist[*telegram.GameHighScore*]

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

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

Parameters**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 thecredentials 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.

Parameters **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.

ReturnsOn 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.

ReturnsA list of chat member objects.

Return typelist[*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.

ReturnsOn 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.

Parameters **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 typetelegram.User

Raisestelegram.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.

Returnslist[telegram.Update]

Raisestelegram.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 isreturned.

Return typelist[*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.

Parameters**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).

Returnsclass: *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.

ReturnsOn success, *True* is returned.

Return typebool

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.

ReturnsOn success, *True* is returned.

Return typebool

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.

ReturnsOn success, *True* is returned.

Return typebool

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.

ReturnsOn success, *True* is returned.

Return typebool

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.

ReturnsOn 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).

ReturnsOn 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.

ReturnsOn 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).

ReturnsOn 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.

ReturnsOn 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.

ReturnsOn 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.

ReturnsOn 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.

ReturnsOn 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.

ReturnsOn 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.

ReturnsOn success, *True* is returned.

Return typebool

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 themessage 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.

ReturnsOn success, *True* is returned.

Return typebool

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.

ReturnsOn success, *True* is returned.

Return typebool

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).

RaisesValueError – 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*, *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*)

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 typedict

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: 4.0message_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: 4.0message_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*)

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: 4.0message_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.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*, *input_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*, *reply_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*, *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*)

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*, *input_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*, *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*)

class telegram.**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*)

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.

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

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: 4.0new_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)
```

ReturnsOn success, *True* is returned.

Return typebool

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)
```

ReturnsOn 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.

Parameterstypes (*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 thetext that belongs to them, calculated based on UTF-16 codepoints.

Return typedict[*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.)

Parameterentity (`telegram.MessageEntity`) – The entity to extract the text from. It must be an entity that belongs to this message.

ReturnsThe text of the given entity

Return typestr

reply_audio(**args*, ***kwargs*)
Shortcut for `bot.sendAudio(update.message.chat_id, *args, **kwargs)`

Keyword Argumentsquote (*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.

ReturnsOn 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 Argumentsquote (*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.

ReturnsOn 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 Argumentsquote (*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.

ReturnsOn 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 Argumentsquote (*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.

ReturnsOn 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 Argumentsquote (*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.

ReturnsOn 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 Argumentsquote (*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.

ReturnsOn 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 Argumentsquote (*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 Argumentsquote (*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.

ReturnsOn 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 Argumentsquote (*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.

ReturnsOn 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.

ReturnsOn 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.

Returnsstr

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.

Returnsstr

to_dict()

Returns

Return typedict

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_me

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) –

Returnstelegram.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 typedict

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.

Parameters **types** (*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 thetext that belongs to them, calculated based on UTF-16 codepoints.

Return typedict[*telegram.MessageEntity*, str]

parse_text_entity(*entity*)

Returns the text from a given *telegram.MessageEntity*.

Note: This method is present because Telegram calculates the offset and length in UTF-16 codepoint pairs, which some versions of Python don't handle automatically. (That is, you can't just slice Message.text with the offset and length.)

Parameters **entity** (telegram.MessageEntity) – The entity to extract the text from. It must be an entity that belongs to this message.

ReturnsThe text of the given entity

Return typestr

to_dict()

Returns

Return typedict

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

Python Module Index

t

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W