## Contents

1 telegram package ........................................... 3
   1.1 telegram.ext package ........................................... 3
      1.1.1 telegram.ext.Updater ........................................... 3
      1.1.2 telegram.ext.Dispatcher ........................................... 5
      1.1.3 telegram.ext.filters Module ........................................... 8
      1.1.4 telegram.ext.Job ........................................... 13
      1.1.5 telegram.ext.JobQueue ........................................... 14
      1.1.6 telegram.ext.MessageQueue ........................................... 16
      1.1.7 telegram.ext.DelayQueue ........................................... 17
      1.1.8 telegram.ext.CallbackContext ........................................... 19
      1.1.9 Handlers ........................................... 20
      1.1.10 Persistence ........................................... 42
   1.2 telegram.utils package ........................................... 47
      1.2.1 telegram.utils.helpers Module ........................................... 47
      1.2.2 telegram.utils.promise.Promise ........................................... 48
      1.2.3 telegram.utils.request.Request ........................................... 48
   1.3 telegram.Animation ........................................... 50
   1.4 telegram.Audio ........................................... 50
   1.5 telegram.Bot ........................................... 51
   1.6 telegram.CallbackQuery ........................................... 87
   1.7 telegram.Chat ........................................... 89
   1.8 telegram.ChatAction ........................................... 93
   1.9 telegram.ChatMember ........................................... 94
   1.10 telegram.ChatPhoto ........................................... 96
   1.11 telegram.constants Module ........................................... 96
   1.12 telegram.Contact ........................................... 97
   1.13 telegram.Document ........................................... 97
   1.14 telegram.error module ........................................... 98
   1.15 telegram.File ........................................... 99
   1.16 telegram.ForceReply ........................................... 100
   1.17 telegram.InlineKeyboardButton ........................................... 100
   1.18 telegram.InlineKeyboardMarkup ........................................... 101
   1.19 telegram.InputFile ........................................... 102
   1.20 telegram.InputMedia ........................................... 103
   1.21 telegram.InputMediaAnimation ........................................... 103
   1.22 telegram.InputMediaAudio ........................................... 104
   1.23 telegram.InputMediaDocument ........................................... 105
   1.24 telegram.InputMediaPhoto ........................................... 106
   1.25 telegram.InputMediaVideo ........................................... 106
   1.26 telegram.KeyboardButton ........................................... 108
1.27 telegram.Location ........................................ 108
1.28 telegram.Message ........................................ 109
1.29 telegram.MessageEntity ................................ 121
1.30 telegram.ParseMode ...................................... 122
1.31 telegram.PhotoSize ...................................... 122
1.32 telegram.ReplyKeyboardRemove ......................... 123
1.33 telegram.ReplyKeyboardMarkup ......................... 123
1.34 telegram.ReplyMarkup ................................... 126
1.35 telegram.TelegramObject ................................. 126
1.36 telegram.Update .......................................... 126
1.37 telegram.User ............................................ 127
1.38 telegram.UserProfilePhotos ............................ 130
1.39 telegram.Venue .......................................... 130
1.40 telegram.Video .......................................... 131
1.41 telegram.VideoNote ...................................... 132
1.42 telegram.Voice ........................................... 133
1.43 telegram.WebhookInfo ................................... 134
1.44 Stickers .................................................. 134
  1.44.1 telegram.Sticker ..................................... 134
  1.44.2 telegram.StickerSet .................................. 135
  1.44.3 telegram.MaskPosition .............................. 136
1.45 Inline Mode ............................................... 137
  1.45.1 telegram.InlineQuery ............................... 137
  1.45.2 telegram.InlineQueryResult ........................ 138
  1.45.3 telegram.InlineQueryResultArticle ............... 138
  1.45.4 telegram.InlineQueryResultAudio .................. 139
  1.45.5 telegram.InlineQueryResultCachedAudio .......... 140
  1.45.6 telegram.InlineQueryResultCachedDocument ....... 141
  1.45.7 telegram.InlineQueryResultCachedGif ............. 142
  1.45.8 telegram.InlineQueryResultCachedMpeg4Gif ...... 143
  1.45.9 telegram.InlineQueryResultCachedPhoto .......... 144
  1.45.10 telegram.InlineQueryResultCachedSticker ...... 145
  1.45.11 telegram.InlineQueryResultCachedVideo ......... 146
  1.45.12 telegram.InlineQueryResultCachedVoice ......... 147
  1.45.13 telegram.InlineQueryResultContact ............. 148
  1.45.14 telegram.InlineQueryResultDocument .......... 149
  1.45.15 telegram.InlineQueryResultGame ................. 150
  1.45.16 telegram.InlineQueryResultGif .................. 150
  1.45.17 telegram.InlineQueryResultLocation ............. 152
  1.45.18 telegram.InlineQueryResultMpeg4Gif .......... 153
  1.45.19 telegram.InlineQueryResultPhoto ............... 154
  1.45.20 telegram.InlineQueryResultVenue ............... 155
  1.45.21 telegram.InlineQueryResultVideo ............... 157
  1.45.22 telegram.InlineQueryResultVoice ............... 158
  1.45.23 telegram.InputMessageContent .................. 159
  1.45.24 telegram.InputTextMessageContent .............. 159
  1.45.25 telegram.InputLocationMessageContent .......... 160
  1.45.26 telegram.InputVenueMessageContent ............ 160
  1.45.27 telegram.InputContactMessageContent .......... 161
  1.45.28 telegram.ChosenInlineResult ................... 161
1.46 Payments ................................................ 162
  1.46.1 telegram.LabeledPrice ............................ 162
  1.46.2 telegram.Invoice ................................... 162
  1.46.3 telegram.ShippingAddress ......................... 163
  1.46.4 telegram.OrderInfo ................................ 163
  1.46.5 telegram.ShippingOption ........................... 164
  1.46.6 telegram.SuccessfulPayment ....................... 164
  1.46.7 telegram.ShippingQuery ............................ 165
1.46.8  telegram.PreCheckoutQuery ......................................................... 166
1.47 Games ................................................................. 167
  1.47.1  telegram.Game ..................................................... 167
  1.47.2  telegram.Callbackgame .............................................. 169
  1.47.3  telegram.GameHighScore ........................................... 169
1.48 Passport ................................................................. 169
  1.48.1  telegram.PasportElementError ...................................... 169
  1.48.2  telegram.PasportElementErrorFile ................................ 170
  1.48.3  telegram.PasportElementErrorReverseSide ...................... 170
  1.48.4  telegram.PasportElementErrorFrontSide ......................... 170
  1.48.5  telegram.PasportElementErrorFiles ................................ 171
  1.48.6  telegram.PasportElementErrorDataField ......................... 171
  1.48.7  telegram.Credentials ............................................. 172
  1.48.8  telegram.DataCredentials ....................................... 172
  1.48.9  telegram.SecureData .............................................. 173
  1.48.10 telegram.FileCredentials ....................................... 173
  1.48.11 telegram.IdDocumentData ....................................... 174
  1.48.12 telegram.PersonalDetails ....................................... 174
  1.48.13 telegram.ResidentialAddress ................................... 174
  1.48.14 telegram.PassportData ......................................... 175
  1.48.15 telegram.PassportFile .......................................... 176
  1.48.16 telegram.EncryptedPassportElement ......................... 176
  1.48.17 telegram.EncryptedCredentials ............................... 178
1.49 Module contents ....................................................... 179

2 Changelog 307
  2.1 Changes ......................................................... 307

3 Version 12.0.0b1 309
  3.1 Context based callbacks (#1100) .................................. 309
  3.2 Persistence (#1017) .................................................. 310
  3.3 Handler overhaul (#1114) ........................................... 310
  3.4 Filter overhaul (#1221) ............................................. 310
  3.5 ConversationHandler ............................................... 310
  3.6 Bug fixes & improvements ......................................... 310
  3.7 Internal improvements ............................................. 311

4 Pre-2019 (up and including to version 11.1.0) 313

5 Indices and tables 327

Python Module Index 329
Below you can find the documentation for the python-telegram-bot library. except for the .ext package most of the objects in the package reflect the types as defined by the telegram bot api.
1.1 telegram.ext package

1.1.1 telegram.ext.Updater

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

Bases: object

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

- **bot**
  - `telegram.Bot` – The bot used with this Updater.

- **user_sig_handler**
  - `signal` – signals the updater will respond to.

- **update_queue**
  - `Queue` – Queue for the updates.

- **job_queue**
  - `telegram.ext.JobQueue` – Jobqueue for the updater.

- **dispatcher**
  - `telegram.ext.Dispatcher` – Dispatcher that handles the updates and dispatches them to the handlers.

- **running**
  - `bool` – Indicates if the updater is running.

- **persistence**
  - `telegram.ext.BasePersistence` – Optional. The persistence class to store data that should
be persistent over restarts.

```python
use_context
bool, optional – True if using context based callbacks.
```

Parameters

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

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

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

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

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

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

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

Starts polling updates from Telegram.

Parameters

- **poll_interval** (`float`, optional) – Time to wait between polling updates from Telegram in seconds. Default is 0.0.
- **timeout** (`float`, optional) – Passed to `telegram.Bot.get_updates`.
- **clean** (`bool`, optional) – Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is False.
**bootstrap_retries** *(int, optional)* – Whether the bootstrapping phase of the Updater will retry on failures on the Telegram server.

- `< 0` - retry indefinitely (default)
- `0` - no retries
- `> 0` - retry up to X times

**allowed_updates** *(List[str], optional)* – Passed to `telegram.Bot.get_updates`.

**read_latency** *(float|int, optional)* – Grace time in seconds for receiving the reply from server. Will be added to the `timeout` value and used as the read timeout from server (Default: 2).

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

**Return type** `Queue`

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

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

**Parameters**

- **listen** *(str, optional)* – IP-Address to listen on. Default `127.0.0.1`.
- **port** *(int, optional)* – Port the bot should be listening on. Default `80`.
- **url_path** *(str, optional)* – Path inside url.
- **cert** *(str, optional)* – Path to the SSL certificate file.
- **key** *(str, optional)* – Path to the SSL key file.
- **clean** *(bool, optional)* – Whether to clean any pending updates on Telegram servers before actually starting the webhook. Default is `False`.
- **bootstrap_retries** *(int, optional)* – Whether the bootstrapping phase of the Updater will retry on failures on the Telegram server.

- `< 0` - retry indefinitely (default)
- `0` - no retries
- `> 0` - retry up to X times

- **webhook_url** *(str, optional)* – Explicitly specify the webhook url. Useful behind NAT, reverse proxy, etc. Default is derived from `listen`, `port` & `url_path`.

- **allowed_updates** *(List[str], optional)* – Passed to `telegram.Bot.set_webhook`.

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

**Return type** `Queue`

**stop()**

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

### 1.1.2 `telegram.ext.Dispatcher`

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

**Bases:** `object`

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

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


workers

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

user_data

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

chat_data

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

persistence

telegram.ext.BasePersistence – Optional. The persistence class to store data that should be persistent over restarts

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.
- workers (int, optional) – Number of maximum concurrent worker threads for the @run_async decorator. defaults to 4.
- persistence (telegram.ext.BasePersistence, optional) – The persistence class to store data that should be persistent over restarts
- use_context (bool, optional) – If set to True Use the context based callback API. During the deprecation period of the old API the default is False. New users: set this to True.

add_error_handler (callback)

Registers an error handler in the Dispatcher.

Parameters

callback (callable) – The callback function for this error handler. Will be called when an error is raised

Callback signature for context based API:

def callback(update: Update, context: CallbackContext)

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

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

add_handler (handler, group=0)

Register a handler.

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

A handler must be an instance of a subclass of telegram.ext.Handler. All handlers are organized in groups with a numeric value. The default group is 0. All groups will be evaluated for handling an update, but only 0 or 1 handler per group will be used. If telegram.ext.DispatcherHandlerStop is raised from one of the handlers, no further handlers (regardless of the group) will be called.

The priority/order of handlers is determined as follows:
• Priority of the group (lower group number == higher priority)

• The first handler in a group which should handle an update (see telegram.ext.Handler.check_update) will be used. Other handlers from the group will not be used. The order in which handlers were added to the group defines the priority.

Parameters

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

• group (int, optional) – The group identifier. Default is 0.

dispatch_error (update, error)

Dispatches an error.

Parameters

• update (str | telegram.Update | None) – The update that caused the error

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

error_handlers = None

List[callable] – A list of errorHandlers.

classmethod get_instance ()

Get the singleton instance of this class.

Returns telegram.ext.Dispatcher

Raises RuntimeError

groups = None

List[int] – A list with all groups.

handlers = None

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

process_update (update)

Processes a single update.


remove_error_handler (callback)

Removes an error handler.

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

remove_handler (handler, group=0)

Remove a handler from the specified group.

Parameters

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

• group (object, optional) – The group identifier. Default is 0.

run_async (func, *args, **kwargs)

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

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

Parameters

• func (callable) – The function to run in the thread.

• *args (tuple, optional) – Arguments to func.
**kwargs (dict, optional) – Keyword arguments to `func`.

Returns

Returns `Promise`

.. code-block:: python

    running = None
    bool – Indicates if this dispatcher is running.

start (ready=None)

    Thread target of thread ‘dispatcher’.

    Runs in background and processes the update queue.

    Parameters

    ready (threading.Event, optional) – If specified, the event will be set

    once the dispatcher is ready.

stop()

    Stops the thread.

update_persistence()

    Update `user_data` and `chat_data` in `persistence`.

user_data = None

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

1.1.3 telegram.ext.filters Module

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

.. code-block:: python

    class telegram.ext.filters.BaseFilter
        Bases: object

        Base class for all Message Filters.

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

        And:

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

        Or:

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

        Not:

        >>> ~ Filters.command

        Also works with more than two filters:

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

        If you want to create your own filters create a class inheriting from this class and implement a `filter` method

        that returns a boolean: `True` if the message should be handled, `False` otherwise. Note that the filters work

        only as class instances, not actual class objects (so remember to initialize your filter classes).

        By default the filters name (what will get printed when converted to a string for display) will be the class

        name. If you want to overwrite this assign a better name to the `name` class variable.

        name

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

        update_filter

        bool – Whether this filter should work on update. If `False` it will run the filter on `update`

        effective_message`. Default is `False`.
data_filter

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

filter(update)

This method must be overwritten.

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

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

Returns dict or bool

class telegram.ext.filters.Filters

Bases: object

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

Examples

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

all = Filters.all

Filter – All Messages.

animation = Filters.animation

Filter – Messages that contain `telegram.Animation`.

audio = Filters.audio

Filter – Messages that contain `telegram.Audio`.

class caption_entity (entity_type)

Bases: `telegram.ext.filters.BaseFilter`

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

Examples

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

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

class chat (chat_id=None, username=None)

Bases: `telegram.ext.filters.BaseFilter`

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

Examples

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

Parameters

- chat_id (int | List[int], optional) – Which chat ID(s) to allow through.
• `username(str | List[str], optional)` – Which username(s) to allow through. If username start with ‘@’ symbol, it will be ignored.

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

```python
cmd = Filters.command
    Filter – Messages starting with /.

cntct = Filters.contact
    Filter – Messages that contain `telegram.Contact`.

dnmnt = Filters.document
    Filter – Messages that contain `telegram.Document`.

class entity(entity_type)
    Bases: `telegram.ext.filters.BaseFilter`

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

    Examples

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

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

fwrd = Filters.forwarded
    Filter – Messages that are forwarded.

gm = Filters.game
    Filter – Messages that contain `telegram.Game`.

gp = Filters.group
    Filter – Messages sent in a group chat.

invc = Filters.invoice
    Filter – Messages that contain `telegram.Invoice`.

class language(lang)
    Bases: `telegram.ext.filters.BaseFilter`

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

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

    Examples

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

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

locn = Filters.location
    Filter – Messages that contain `telegram.Location`.

pssrt documentation = Filters.passport_data
    Filter – Messages that contain a `telegram.PassportData`

phot = Filters.photo
    Filter – Messages that contain `telegram.PhotoSize`.

privt = Filters.private
    Filter – Messages sent in a private chat.
class regex(pattern)
    Bases: telegram.ext.filters.BaseFilter

    Filters updates by searching for an occurrence of pattern in the message text. The re.search function is used to determine whether an update should be filtered.

    Refer to the documentation of the re module for more information.

    To get the groups and groupdict matched, see telegram.ext.CallbackContext.matches.

Examples

Use MessageHandler(Filters.regex(r'help'), callback) to capture all messages that contain the word help. You can also use MessageHandler(Filters.regex(re.compile(r'help', re.IGNORECASE), callback) if you want your pattern to be case insensitive. This approach is recommended if you need to specify flags on your pattern.

Parameters

    pattern (str|Pattern) -- The regex pattern.

reply = Filters.reply
Filter -- Messages that are a reply to another message.

status_update = Filters.status_update
Subset for messages containing a status update.

Examples

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

chat_created
    Filter -- Messages that contain telegram.Message.group_chat_created, telegram.Message.supergroup_chat_created or telegram.Message.channel_chat_created.

deleat_chat_photo
    Filter -- Messages that contain telegram.Message.delete_chat_photo.

left_chat_member
    Filter -- Messages that contain telegram.Message.left_chat_member.

migrate
    Filter -- Messages that contain telegram.Message.migrate_from_chat_id or :attr: telegram.Message.migrate_from_chat_id.

new_chat_members
    Filter -- Messages that contain telegram.Message.new_chat_members.

new_chat_photo
    Filter -- Messages that contain telegram.Message.new_chat_photo.

new_chat_title
    Filter -- Messages that contain telegram.Message.new_chat_title.

pinned_message
    Filter -- Messages that contain telegram.Message.pinned_message.

sticker = Filters.sticker
Filter -- Messages that contain telegram.Sticker.

successful_payment = Filters.successful_payment
Filter -- Messages that confirm a telegram.SuccessfulPayment.

text = Filters.text
Filter -- Text Messages.
update = _UpdateType
Subset for filtering the type of update.

**Examples**

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

**message**
Filter – Updates with telegram.Update.message

**edited_message**
Filter – Updates with telegram.Update.edited_message

**messages**
Filter – Updates with either telegram.Update.message or telegram.Update.edited_message

**channel_post**
Filter – Updates with telegram.Update.channel_post

**edited_channel_post**
Filter – Updates with telegram.Update.edited_channel_post

**channel_posts**
Filter – Updates with either telegram.Update.channel_post or telegram.Update.edited_channel_post

**class user** (`user_id=None, username=None`)
Bases: telegram.ext.filters.BaseFilter
Filters messages to allow only those which are from specified user ID.

**Examples**

MessageHandler(Filters.user(1234), callback_method)

**Parameters**

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

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

**venue** = Filters.venue
Filter – Messages that contain telegram.Venue.

**video** = Filters.video
Filter – Messages that contain telegram.Video.

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

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

**class** telegram.ext.filters.InvertedFilter(`f`)
Bases: telegram.ext.filters.BaseFilter
Represents a filter that has been inverted.

**Parameters** `f` – The filter to invert.
filter(update)
This method must be overwritten.

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

Parameters update(telegram.Update) – The update that is tested.
Returns dict or bool

class telegram.ext.filters.MergedFilter(base_filter, and_filter=None, or_filter=None)
Bases: telegram.ext.filters.BaseFilter
Represents a filter consisting of two other filters.

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

filter(update)
This method must be overwritten.

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

Parameters update(telegram.Update) – The update that is tested.
Returns dict or bool

1.1.4 telegram.ext.Job

class telegram.ext.Job(callback, interval=None, repeat=True, context=None, days=(0, 1, 2, 3, 4, 5, 6), name=None, job_queue=None)
Bases: object
This class encapsulates a Job.

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

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

name
str – Optional. The name of the new job.

Parameters
• callback(callable) – The callback function that should be executed by the new job. It should take bot, job as parameters, where job is the telegram.ext.Job instance. It can be used to access it’s context or change it to a repeating job.
• interval (int | float | datetime.timedelta, optional) – The interval in which the job will run. If it is an int or a float, it will be interpreted as seconds. If
you don’t set this value, you must set repeat to False and specify next_t when you put the job into the job queue.

- **repeat** (bool, optional) – If this job should be periodically execute its callback function (True) or only once (False). Defaults to True.

- **context** (object, optional) – Additional data needed for the callback function. Can be accessed through job.context in the callback. Defaults to None.

- **name** (str, optional) – The name of the new job. Defaults to `callback.__name__`.

- **days** (Tuple[int], optional) – Defines on which days of the week the job should run. Defaults to `Days.EVERY_DAY`

- **job_queue** (telegram.ext.JobQueue, optional) – The JobQueue this job belongs to. Only optional for backward compatibility with `JobQueue.put()`.

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

```python
enabled
bool – Whether this job is enabled.
```

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

```python
interval_seconds
int – The interval for this job in seconds.
```

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

```python
removed
bool – Whether this job is due to be removed.
```

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

```python
run (dispatcher)
Executes the callback function.
```

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

### 1.1.5 telegram.ext.JobQueue

```python
class telegram.ext.JobQueue (bot=None)
Bases: object
```

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

```python
_queue
PriorityQueue – The queue that holds the Jobs.
```

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

```python
get_jobs_by_name (name)
Returns a tuple of jobs with the given name that are currently in the JobQueue
```

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

```python
run_daily (callback, time, days=(0, 1, 2, 3, 4, 5, 6), context=None, name=None)
Creates a new Job that runs on a daily basis and adds it to the queue.
```
Parameters

- **callback**(callable) – The callback function that should be executed by the new job. It should take `bot`, `job` as parameters, where `job` is the `telegram.ext.Job` instance. It can be used to access its `Job.context` or change it to a repeating job.

- **time**(datetime.time) – Time of day at which the job should run.

- **days**(Tuple[int], optional) – Defines on which days of the week the job should run. Defaults to EVERY_DAY

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

- **name**(str, optional) – The name of the new job. Defaults to `callback.__name__`.

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

**Return type** `telegram.ext.Job`

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

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

Parameters

- **callback**(callable) – The callback function that should be executed by the new job. It should take `bot`, `job` as parameters, where `job` is the `telegram.ext.Job` instance. It can be used to access its `job.context` or change it to a repeating job.

- **when**(int | float | datetime.timedelta | datetime.datetime | datetime.time) – Time in or at which the job should run. This parameter will be interpreted depending on its type.

  - int or float will be interpreted as “seconds from now” in which the job should run.

  - datetime.timedelta will be interpreted as “time from now” in which the job should run.

  - datetime.datetime will be interpreted as a specific date and time at which the job should run.

  - datetime.time will be interpreted as a specific time of day at which the job should run. This could be either today or, if the time has already passed, tomorrow.

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

- **name**(str, optional) – The name of the new job. Defaults to `callback.__name__`.

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

**Return type** `telegram.ext.Job`

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

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

Parameters

- **callback**(callable) – The callback function that should be executed by the new job. It should take `bot`, `job` as parameters, where `job` is the `telegram.ext.Job` instance. It can be used to access its `job.context` or change it to a repeating job.

- **interval**(int | float | datetime.timedelta) – The interval in which the job will run. If it is an int or a float, it will be interpreted as seconds.
• **first** (int | float | datetime.timedelta | datetime.datetime | datetime.time, optional) – Time in or at which the job should run. This parameter will be interpreted depending on its type.
  
  - int or float will be interpreted as “seconds from now” in which the job should run.
  - datetime.timedelta will be interpreted as “time from now” in which the job should run.
  - datetime.datetime will be interpreted as a specific date and time at which the job should run.
  - datetime.time will be interpreted as a specific time of day at which the job should run. This could be either today or, if the time has already passed, tomorrow.

Defaults to **interval**

• **context** (object, optional) – Additional data needed for the callback function. Can be accessed through job.context in the callback. Defaults to None.

• **name** (str, optional) – The name of the new job. Defaults to callback.__name__.

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

**Return type** telegram.ext.Job

**start**()

Starts the job_queue thread.

**stop**()

Stops the thread.

**tick**()

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

### 1.1.6 telegram.ext.MessageQueue

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

**Bases:** object

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

**Parameters**

• **all_burst_limit** (int, optional) – Number of maximum all-type callbacks to process per time-window defined by all_time_limit_ms. Defaults to 30.

• **all_time_limit_ms** (int, optional) – Defines width of all-type time-window used when each processing limit is calculated. Defaults to 1000 ms.

• **group_burst_limit** (int, optional) – Number of maximum group-type callbacks to process per time-window defined by group_time_limit_ms. Defaults to 20.

• **group_time_limit_ms** (int, optional) – Defines width of group-type time-window used when each processing limit is calculated. Defaults to 60000 ms.

• **exc_route** (callable, optional) – A callable, accepting one positional argument; used to route exceptions from processor threads to main thread; is called on Exception subclass exceptions. If not provided, exceptions are routed through dummy handler, which re-raises them.
• **autostart** (bool, optional) – If True, processors are started immediately after object’s creation; if False, should be started manually by `start` method. Defaults to True.

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

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

**Parameters**

- **promise** (callable) – Mainly the `telegram.utils.promise.Promise` (see Notes for other callables), that is processed in delay queues.

- **is_group_msg** (bool, optional) – Defines whether `promise` would be processed in `group` DelayQueue (if set to `True`), or only through `all` DelayQueue (if set to `False`), resulting in needed delays to avoid hitting specified limits. Defaults to `False`.

**Notes**

Method is designed to accept `telegram.utils.promise.Promise` as `promise` argument, but other callables could be used too. For example, lambdas or simple functions could be used to wrap original func to be called with needed args. In that case, be sure that either wrapper func does not raise outside exceptions or the proper `exc_route` handler is provided.

**Returns** Used as `promise` argument.

**Return type** callable

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

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

**__weakref__**

list of weak references to the object (if defined)

**start**()

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

**stop**(timeout=None)

Used to gently stop processor and shutdown its thread.

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

1.1.7 `telegram.ext.DelayQueue`

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

**Bases:** `threading.Thread`

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

**burst_limit**

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

**time_limit**

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

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

**name**

Str – Thread's name.

**Parameters**

- **queue** (Queue, optional) – Used to pass callbacks to thread. Creates Queue implicitly if not provided.
- **burst_limit** (int, optional) – Number of maximum callbacks to process per time-window defined by time_limit_ms. Defaults to 30.
- **time_limit_ms** (int, optional) – Defines width of time-window used when each processing limit is calculated. Defaults to 1000.
- **exc_route** (callable, optional) – A callable, accepting 1 positional argument; used to route exceptions from processor thread to main thread; is called on Exception subclass exceptions. If not provided, exceptions are routed through dummy handler, which re-raises them.
- **autostart** (bool, optional) – If True, processor is started immediately after object’s creation; if False, should be started manually by start method. Defaults to True.
- **name** (str, optional) – Thread’s name. Defaults to 'DelayQueue-N', where N is sequential number of object created.

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

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

**Parameters**

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

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

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

- **group** should be None; reserved for future extension when a ThreadGroup class is implemented.
- **target** is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.
- **name** is the thread name. By default, a unique name is constructed of the form “Thread-N” where N is a small decimal number.
- **args** is the argument tuple for the target invocation. Defaults to ()
- **kwargs** is a dictionary of keyword arguments for the target invocation. Defaults to {}.

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

**run()**

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

**stop**(timeout=None)

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

1.1.8 *telegram.ext.CallbackContext*

**class** `telegram.ext.CallbackContext(dispatcher)`

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

**Note:** `telegram.ext.Dispatcher` will create a single context for an entire update. This means that if you got 2 handlers in different groups and they both get called, they will get passed the same `CallbackContext` object (of course with proper attributes like `.matches` differing). This allows you to add custom attributes in a lower handler group callback, and then subsequently access those attributes in a higher handler group callback. Note that the attributes on `CallbackContext` might change in the future, so make sure to use a fairly unique name for the attributes.

**Warning:** Do not combine custom attributes and `@run_async`. Due to how `@run_async` works, it will almost certainly execute the callbacks for an update out of order, and the attributes that you think you added will not be present.

**chat_data**

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

**user_data**

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

**matches**

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

**args**

List[str], optional – Arguments passed to a command if the associated update is handled by `telegram.ext.CommandHandler`, `telegram.ext.PrefixHandler` or `telegram.ext.StringCommandHandler`. It contains a list of the words in the text after the command, using any whitespace string as a delimiter.

**error**

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

**job**

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

**bot**

`telegram.Bot` – The bot associated with this context.

**job_queue**

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

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

update_queue

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

## 1.1.9 Handlers

telegram.ext.Handler

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

Bases: object

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

callback
callable – The callback function for this handler.

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

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

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

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

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

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

### Parameters

- **callback (callable)** – The callback function for this handler. Will be called when check_update has determined that an update should be processed by this handler. Callback signature for context based API:

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

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

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

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

• **pass_chat_data** (bool, optional) – If set to True, a keyword argument called `chat_data` will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.

**check_update** *(update)*

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

**Parameters**

- `update` *(str | telegram.Update)* – The update to be tested.

**Returns**

Either `None` or `False` if the update should not be handled. Otherwise an object that will be passed to `handle_update` and `collect_additional_context` when the update gets handled.

**collect_additional_context** *(context, update, dispatcher, check_result)*

 Prepares additional arguments for the context. Override if needed.

**Parameters**

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

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

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

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

**Parameters**

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

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

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

**Parameters**

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

telegram.ext.CallbackQueryHandler

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

**Bases:** `telegram.ext.handler.Handler`
Handler class to handle Telegram callback queries. Optionally based on a regex.

Read the documentation of the re module for more information.

**callback**
- callable – The callback function for this handler.

**pass_update_queue**
- bool – Determines whether `update_queue` will be passed to the callback function.

**pass_job_queue**
- bool – Determines whether `job_queue` will be passed to the callback function.

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

**pass_groups**
- bool – Determines whether `groups` will be passed to the callback function.

**pass_groupdict**
- bool – Determines whether `groupdict` will be passed to the callback function.

**pass_user_data**
- bool – Determines whether `user_data` will be passed to the callback function.

**pass_chat_data**
- bool – Determines whether `chat_data` will be passed to the callback function.

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

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

### Parameters

- **callback (callable)** – The callback function for this handler. Will be called when `check_update` has determined that an update should be processed by this handler. Callback signature for context based API:

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

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

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

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

- **pattern (str | Pattern, optional)** – Regex pattern. If not `None`, `re.match` is used on `telegram.CallbackQuery.data` to determine if an update should be handled by this handler.
• **pass_groups** (bool, optional) – If the callback should be passed the result of `re.match(pattern, data).groups()` as a keyword argument called `groups`. Default is False. DEPRECATED: Please switch to context based callbacks.

• **pass_groupdict** (bool, optional) – If the callback should be passed the result of `re.match(pattern, data).groupdict()` as a keyword argument called `groupdict`. Default is False. DEPRECATED: Please switch to context based callbacks.

• **pass_user_data** (bool, optional) – If set to True, a keyword argument called `user_data` will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.

• **pass_chat_data** (bool, optional) – If set to True, a keyword argument called `chat_data` will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.

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

**collect_additional_context** *(context, update, dispatcher, check_result)*
Prepares additional arguments for the context. Override if needed.

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

**collect_optional_args** *(dispatcher, update=None, check_result=None)*
Prepares the optional arguments. If the handler has additional optional args, it should subclass this method, but remember to call this super method.

DEPRECATED: This method is being replaced by new context based callbacks. Please see [https://git.io/fxJuV](https://git.io/fxJuV) for more info.

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

**telegram.ext.ChosenInlineResultHandler**
pass_job_queue
bool – Determines whether job_queue will be passed to the callback function.

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

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

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

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

Parameters

• callback (callable) – The callback function for this handler. Will be called when check_update has determined that an update should be processed by this handler. Callback signature for context based API:

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

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

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

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

• pass_user_data (bool, optional) – If set to True, a keyword argument called user_data will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.

• pass_chat_data (bool, optional) – If set to True, a keyword argument called chat_data will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.

check_update (update)
Determines whether an update should be passed to this handlers callback.


Returns bool

telegram.ext.ConversationHandler

class telegram.ext.ConversationHandler (entry_points, states, fallbacks, allow_reentry=False, per_chat=True, per_user=True, per_message=False, conversation_timeout=None, name=None, persistent=False)
A handler to hold a conversation with a single user by managing four collections of other handlers. Note that neither posts in Telegram Channels, nor group interactions with multiple users are managed by instances of this class.

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

The second collection, a dict named `states`, contains the different conversation steps and one or more associated handlers that should be used if the user sends a message when the conversation with them is currently in that state. Here you can also define a state for `TIMEOUT` to define the behavior when `conversation_timeout` is exceeded, and a state for `WAITING` to define behavior when a new update is received while the previous @run_async decorated handler is not finished.

The third collection, a list named `fallbacks`, is used if the user is currently in a conversation but the state has either no associated handler or the handler that is associated to the state is inappropriate for the update, for example if the update contains a command, but a regular text message is expected. You could use this for a /cancel command or to let the user know their message was not recognized.

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

To change the state of conversation, the callback function of a handler must return the new state after responding to the user. If it does not return anything (returning `None` by default), the state will not change. If an entry point callback function returns None, the conversation ends immediately after the execution of this callback function. To end the conversation, the callback function must return `END` or `-1`. To handle the conversation timeout, use handler `TIMEOUT` or `-2`.

**entry_points**

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

**states**

- `Dict[object, List[telegram.ext.Handler]]` – A dict that defines the different states of conversation a user can be in and one or more associated `Handler` objects that should be used in that state.

**fallbacks**

- `List[telegram.ext.Handler]` – A list of handlers that might be used if the user is in a conversation, but every handler for their current state returned `False` on `check_update`.

**allow_reentry**

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

**per_chat**

- `bool` – If the conversation key should contain the Chat’s ID.

**per_user**

- `bool` – If the conversation key should contain the User’s ID.

**per_message**

- `bool` – If the conversation key should contain the Message’s ID.

**conversation_timeout**

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

**name**

- `str` – Optional. The name for this conversation handler. Required for persistence

**persistent**

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

- **entry_points** (List[telegram.ext.Handler]) – A list of Handler objects that can trigger the start of the conversation. The first handler which \( \text{check\_update} \) method returns True will be used. If all return False, the update is not handled.

- **states** (Dict[object, List[telegram.ext.Handler]]) – A dict that defines the different states of conversation a user can be in and one or more associated Handler objects that should be used in that state. The first handler which \( \text{check\_update} \) method returns True will be used.

- **fallbacks** (List[telegram.ext.Handler]) – A list of handlers that might be used if the user is in a conversation, but every handler for their current state returned False on \( \text{check\_update} \). The first handler which \( \text{check\_update} \) method returns True will be used. If all return False, the update is not handled.

- **allow_reentry** (bool, optional) – If set to True, a user that is currently in a conversation can restart the conversation by triggering one of the entry points.

- **per_chat** (bool, optional) – If the conversation key should contain the Chat’s ID. Default is True.

- **per_user** (bool, optional) – If the conversation key should contain the User’s ID. Default is True.

- **per_message** (bool, optional) – If the conversation key should contain the Message’s ID. Default is False.

- **conversation_timeout** (float | datetime.timedelta, optional) – When this handler is inactive more than this timeout (in seconds), it will be automatically ended. If this value is 0 or None (default), there will be no timeout. The last received update will be handled by ALL the handler’s who’s \( \text{check\_update} \) method returns True that are in the state ConversationHandler.TIMEOUT.

- **name** (str, optional) – The name for this conversation handler. Required for persistence.

- **persistent** (bool, optional) – If the conversations dict for this handler should be saved. Name is required and persistence has to be set in telegram.ext.Updater.

Raises ValueError

END = -1
int – Used as a constant to return when a conversation is ended.

TIMEOUT = -2
int – Used as a constant to handle state when a conversation is timed out.

WAITING = -3
int – Used as a constant to handle state when a conversation is still waiting on the previous @run_sync decorated running handler to finish.

**check_update** \( \text{(update)} \)

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

Parameters

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

Returns bool

**handle_update** \( \text{(update, dispatcher, check\_result, context=None)} \)

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

Parameters

- **check\_result** – The result from check_update. For this handler it’s a tuple of key, handler, and the handler’s check result.
• `update` (:class:`telegram.Update`) – Incoming telegram update.

• `dispatcher` (:class:`telegram.ext.Dispatcher`) – Dispatcher that originated the Update.

```python
persistence = None
```

.. code-block:: python

    telegram.ext.BasePersistance – The persistence used to store conversations. Set by dispatcher

**telegram.ext.CommandHandler**

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

.. code-block:: python

    Bases: telegram.ext.handler.Handler

Handler class to handle Telegram commands.

Commands are Telegram messages that start with `/`, optionally followed by an `@` and the bot’s name and/or some additional text. The handler will add a list to the `CallbackContext` named `CallbackContext.args`. It will contain a list of strings, which is the text following the command split on single or consecutive whitespace characters.

By default the handler listens to messages as well as edited messages. To change this behavior use `~Filters.update.edited_message` in the filter argument.

**command**

.. code-block:: python

    str | List[str] – The command or list of commands this handler should listen for. Limitations are the same as described here https://core.telegram.org/bots#commands

**callback**

.. code-block:: python

    callable – The callback function for this handler.

**filters**

.. code-block:: python

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

**allow_edited**

.. code-block:: python

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

**pass_args**

.. code-block:: python

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

**pass_update_queue**

.. code-block:: python

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

**pass_job_queue**

.. code-block:: python

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

**pass_user_data**

.. code-block:: python

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

**pass_chat_data**

.. code-block:: python

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

---

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

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

---

**Parameters**

---

1.1. **telegram.ext package**
• **command** *(str | List[str])* – The command or list of commands this handler should listen for. Limitations are the same as described here https://core.telegram.org/bots#commands

• **callback** *(callable)* – The callback function for this handler. Will be called when *check_update* has determined that an update should be processed by this handler. Callback signature for context based API:

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

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

• **filters** *(telegram.ext.BaseFilter, optional)* – A filter inheriting from *telegram.ext.filters.BaseFilter*. Standard filters can be found in *telegram.ext.filters.Filters*. Filters can be combined using bitwise operators (& for and, | for or, ~ for not).

• **allow_edited** *(bool, optional)* – Determines whether the handler should also accept edited messages. Default is False. DEPRECATED: Edited is allowed by default. To change this behavior use ~Filters.update.edited_message.

• **pass_args** *(bool, optional)* – Determines whether the handler should be passed the arguments passed to the command as a keyword argument called *args*. It will contain a list of strings, which is the text following the command split on single or consecutive whitespace characters. Default is False DEPRECATED: Please switch to context based callbacks.

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

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

• **pass_user_data** *(bool, optional)* – If set to True, a keyword argument called *user_data* will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.

• **pass_chat_data** *(bool, optional)* – If set to True, a keyword argument called *chat_data* will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.

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

**check_update**(update)

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

**Parameters**

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

**Returns** The list of args for the handler

**Return type** *list*

**collect_additional_context**(context, update, dispatcher, check_result)

Prepares additional arguments for the context. Override if needed.

**Parameters**

- **context** *(telegram.ext.CallbackContext)* – The context object.
- **update** *(telegram.Update)* – The update to gather chat/user id from.
collect_optional_args (dispatcher, update=None, check_result=None)

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

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

Parameters

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

**telegram.ext.InlineQueryHandler**

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

Bases: telegram.ext.handler.Handler

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

**callback**

callable – The callback function for this handler.

**pass_update_queue**

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

**pass_job_queue**

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

**pattern**

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

**pass_groups**

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

**pass_groupdict**

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

**pass_user_data**

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

**pass_chat_data**

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

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

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

**Parameters**
• **callback** *(callable)* – The callback function for this handler. Will be called when `check_update` has determined that an update should be processed by this handler. 

Callback signature for context based API:

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

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

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

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

• **pattern** *(str | Pattern, optional)* – Regex pattern. If not None, `re.match` is used on `telegram.InlineQuery.query` to determine if an update should be handled by this handler.

• **pass_groups** *(bool, optional)* – If the callback should be passed the result of `re.match(pattern, data).groups()` as a keyword argument called `groups`. Default is False. DEPRECATED: Please switch to context based callbacks.

• **pass_groupdict** *(bool, optional)* – If the callback should be passed the result of `re.match(pattern, data).groupdict()` as a keyword argument called `groupdict`. Default is False. DEPRECATED: Please switch to context based callbacks.

• **pass_user_data** *(bool, optional)* – If set to True, a keyword argument called `user_data` will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.

• **pass_chat_data** *(bool, optional)* – If set to True, a keyword argument called `chat_data` will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.

**check_update** *(update)*

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

**Parameters**


**Returns**

bool

**collect_additional_context** *(context, update, dispatcher, check_result)*

Prepares additional arguments for the context. Override if needed.

**Parameters**

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

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

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

DEPRECATED: This method is being replaced by new context based callbacks. Please see [https://git.io/fxJuV](https://git.io/fxJuV) for more info.
Parameters

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

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

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

**filters**

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

**callback**

callable – The callback function for this handler.

**pass_update_queue**

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

**pass_job_queue**

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

**pass_user_data**

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

**pass_chat_data**

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

**message_updates**

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

**channel_post_updates**

bool – Should channel posts updates be handled? Default is None.

**edited_updates**

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

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

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

**Parameters**

- **filters** (`telegram.ext.BaseFilter`, optional) – A filter inheriting from `telegram.ext.filters.BaseFilter`. Standard filters can be found in `telegram.ext.filters.Filters`. Filters can be combined using bitwise operators (& for and, | for or, ~ for not). Default is `telegram.ext.filters.Filters.update`. This defaults to all message_type updates being: message, edited_message, channel_post and edited_channel_post. If you don’t want or need any of those pass ~Filters.update.* in the filter argument.
• **callback** *(callable)* – The callback function for this handler. Will be called when `check_update` has determined that an update should be processed by this handler. Callback signature for context based API:

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

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

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

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

• **pass_user_data** *(bool, optional)* – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. Default is `False`. DEPRECATED: Please switch to context based callbacks.

• **pass_chat_data** *(bool, optional)* – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. Default is `False`. DEPRECATED: Please switch to context based callbacks.

• **message_updates** *(bool, optional)* – Should “normal” message updates be handled? Default is `None`. DEPRECATED: Please switch to filters for update filtering.

• **channel_post_updates** *(bool, optional)* – Should channel posts updates be handled? Default is `None`. DEPRECATED: Please switch to filters for update filtering.

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

**Raises** *ValueError*

• **check_update** *(update)*

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

Parameters


Returns *bool*

• **collect_additional_context** *(context, update, dispatcher, check_result)*

Prepares additional arguments for the context. Override if needed.

Parameters


- `update` *(telegram.Update)* – The update to gather chat/user id from.

- `dispatcher` *(telegram.ext.Dispatcher)* – The calling dispatcher.

- `check_result` – The result (return value) from `check_update`.

**telegram.ext.PreCheckoutQueryHandler**

```
class telegram.ext.PreCheckoutQueryHandler:
```

Bases: `telegram.ext.handler.Handler`

```python

callback=callback,  pass_update_queue=False,  
pass_job_queue=False,  pass_user_data=False,  
pass_chat_data=False)
```

```
Handler class to handle Telegram PreCheckout callback queries.

**callback**

Callable – The callback function for this handler.

**pass_update_queue**

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

**pass_job_queue**

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

**pass_user_data**

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

**pass_chat_data**

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

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

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

**Parameters**

- **callback** (Callable) – The callback function for this handler. Will be called when check_update has determined that an update should be processed by this handler. Callback signature for context based API:

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

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

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

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

- **pass_user_data** (bool, optional) – If set to True, a keyword argument called user_data will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.

- **pass_chat_data** (bool, optional) – If set to True, a keyword argument called chat_data will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.

**check_update** (update)

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

**Parameters**

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

**Returns** bool
telegram.ext.PrefixHandler

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

Bases: telegram.ext.commandhandler.CommandHandler

Handler class to handle custom prefix commands

This is a intermediate handler between MessageHandler and CommandHandler. It supports configurable commands with the same options as CommandHandler. It will respond to every combination of prefix and command. It will add a list to the CallbackContext named CallbackContext.args. It will contain a list of strings, which is the text following the command split on single or consecutive whitespace characters.

Examples:

Single prefix and command:

PrefixHandler('!', 'test', callback) will respond to '!test'.

Multiple prefixes, single command:

PrefixHandler(['!', '#'], 'test', callback) will respond to '!test' and '#test'.

Multiple prefixes and commands:

PrefixHandler(['!', '#'], ['test', 'help'], callback) will respond to '!test', '#test', '!help' and '#help'.

By default the handler listens to messages as well as edited messages. To change this behavior use ~"Filters.update.edited_message".

prefix  
str | List[str] – The prefix(es) that will precede command.

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

callback
callable – The callback function for this handler.

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

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

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

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

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

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

Note: pass_user_data and pass_chat_data determine whether a dict you can use to keep any data in will be sent to the callback function. Related to either the user or the chat that the update was
sent in. For each update from the same user or in the same chat, it will be the same dict.
Note that this is DEPRECATED, and you should use context based callbacks. See https://git.io/fxJuV for more info.

Parameters

- prefix (str | List[str]) – The prefix(es) that will precede command.
- command (str | List[str]) – The command or list of commands this handler should listen for.
- callback (callable) – The callback function for this handler. Will be called when check_update has determined that an update should be processed by this handler. Callback signature for context based API:
  
  ```python
def callback(update: Update, context: CallbackContext)
  ```

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

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

check_update (update)

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

Parameters


Returns

The list of args for the handler

Return type

list

collect_additional_context (context, update, dispatcher, check_result)

Prepares additional arguments for the context. Override if needed.

Parameters

1.1. telegram.ext package

35
• **context** (*telegram.ext.CallbackContext*) – The context object.
• **update** (*telegram.Update*) – The update to gather chat/user id from.
• **dispatcher** (*telegram.ext.Dispatcher*) – The calling dispatcher.
• **check_result** – The result (return value) from `check_update`.

### `telegram.ext.RegexHandler`

class `telegram.ext.RegexHandler`

```python
class telegram.ext.RegexHandler(pattern, callback, **kwargs):
    ...```

Bases: `telegram.ext.messagehandler.MessageHandler`

Handler class to handle Telegram updates based on a regex.

It uses a regular expression to check text messages. Read the documentation of the `re` module for more information. The `re.match` function is used to determine if an update should be handled by this handler.

**pattern**

str | Pattern – The regex pattern.

**callback**

callable – The callback function for this handler.

**pass_groups**

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

**pass_groupdict**

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

**pass_update_queue**

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

**pass_job_queue**

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

**pass_user_data**

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

**pass_chat_data**

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

**allow_edited**

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

**message_updates**

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

**channel_post_updates**

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

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

**Parameters**

- **pattern** (str | Pattern) – The regex pattern.
- **callback** (callable) – The callback function for this handler. Will be called when `check_update` has determined that an update should be processed by this handler.

Callback signature for context based API:

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

The return value of the callback is usually ignored except for the special case of `telegram.ext.ConversationHandler`.
• **pass_groups** (bool, optional) – If the callback should be passed the result of `re.match(pattern, data).groups()` as a keyword argument called `groups`. Default is `False`.

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

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

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

• **pass_user_data** (bool, optional) – If set to `True`, a keyword argument called `user_data` will be passed to the callback function. Default is `False`.

• **pass_chat_data** (bool, optional) – If set to `True`, a keyword argument called `chat_data` will be passed to the callback function. Default is `False`.

• **message_updates** (bool, optional) – Should “normal” message updates be handled? Default is `True`.

• **channel_post_updates** (bool, optional) – Should channel posts updates be handled? Default is `True`.

• **edited_updates** (bool, optional) – Should “edited” message updates be handled? Default is `False`.

Raises: `ValueError`

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

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

DEPRECATED: This method is being replaced by new context based callbacks. Please see [https://git.io/fxJuV](https://git.io/fxJuV) for more info.

Parameters:

• **dispatcher** *(`telegram.ext.Dispatcher`)* – The dispatcher.

• **update** *(`telegram.Update`)* – The update to gather chat/user id from.

• **check_result** – The result from check_update

---

**telegram.ext.ShippingQueryHandler**

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

Bases: `telegram.ext.handler.Handler`

Handler class to handle Telegram shipping callback queries.

**callback**

callable – The callback function for this handler.

**pass_update_queue**

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

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

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

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

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

Parameters
- callback (callable) – The callback function for this handler. Will be called when check_update has determined that an update should be processed by this handler. Callback signature for context based API:

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

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

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

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

- pass_user_data (bool, optional) – If set to True, a keyword argument called user_data will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.

- pass_chat_data (bool, optional) – If set to True, a keyword argument called chat_data will be passed to the callback function. Default is False. DEPRECATED: Please switch to context based callbacks.

check_update (update)
Determines whether an update should be passed to this handlers callback.


Returns bool

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

Bases: telegram.ext.handler.Handler

Handler class to handle string commands. Commands are string updates that start with /.
Note: This handler is not used to handle Telegram `telegram.Update`, but strings manually put in the queue. For example to send messages with the bot using command line or API.

**command**
```
str – The command this handler should listen for.
```

**callback**
```
callable – The callback function for this handler.
```

**pass_args**
```
bool – Determines whether the handler should be passed `args`.
```

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

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

**Parameters**

- **callback** *(callable)*: The callback function for this handler. Will be called when `check_update` has determined that an update should be processed by this handler. Callback signature for context based API:

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

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

- **pass_args** *(bool, optional)*: Determines whether the handler should be passed the arguments passed to the command as a keyword argument called `args`. It will contain a list of strings, which is the text following the command split on single or consecutive whitespace characters. Default is False. DEPRECATED: Please switch to context based callbacks.

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

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

**check_update** *(update)*

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

**Parameters**

- **update** *(str)*: An incoming command.

**Returns** *(bool)*

**collect_additional_context** *(context, update, dispatcher, check_result)*

Prepares additional arguments for the context. Override if needed.

**Parameters**

- **context** *(telegram.ext.CallbackContext)*: The context object.

- **update** *(telegram.Update)*: The update to gather chat/user id from.

- **dispatcher** *(telegram.ext.Dispatcher)*: The calling dispatcher.

- **check_result** – The result (return value) from `check_update`.

1.1. `telegram.ext` package
**collect_optional_args** *(dispatcher, update=None, check_result=None)*

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

 DEPRECATED: This method is being replaced by new context based callbacks. Please see [https://git.io/fxJuV](https://git.io/fxJuV) for more info.

 **Parameters**

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

---

**telegram.ext.StringRegexHandler**

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

 Bases: **telegram.ext.handler.Handler**

 Handler class to handle string updates based on a regex which checks the update content.

 Read the documentation of the `re` module for more information. The `re.match` function is used to determine if an update should be handled by this handler.

 **Note:** This handler is not used to handle Telegram `telegram.Update`, but strings manually put in the queue. For example to send messages with the bot using command line or API.

 **pattern**

 str|Pattern – The regex pattern.

 **callback**

 callable – The callback function for this handler.

 **pass_groups**

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

 **pass_groupdict**

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

 **pass_update_queue**

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

 **pass_job_queue**

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

 **Parameters**

 - **pattern** *(str|Pattern)* – The regex pattern.
 - **callback** *(callable)* – The callback function for this handler. Will be called when `check_update` has determined that an update should be processed by this handler. Callback signature for context based API:

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

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

 - **pass_groups** *(bool, optional)* – If the callback should be passed the result of `re.match(pattern, data).groups()` as a keyword argument called `groups`. Default is `False` DEPRECATED: Please switch to context based callbacks.
• **pass_groupdict** (bool, optional) – If the callback should be passed the result of `re.match(pattern, data).groupdict()` as a keyword argument called `groupdict`. Default is `False`. DEPRECATED: Please switch to context based callbacks.

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

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

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

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

Returns `bool`

`collect_additional_context(context, update, dispatcher, check_result)`
Prepares additional arguments for the context. Override if needed.

Parameters
• `context` (`telegram.ext.CallbackContext`) – The context object.
• `update` (`telegram.Update`) – The update to gather chat/user id from.
• `dispatcher` (`telegram.ext.Dispatcher`) – The calling dispatcher.
• `check_result` – The result (return value) from `check_update`.

`collect_optional_args(dispatcher, update=None, check_result=None)`
Prepares the optional arguments. If the handler has additional optional args, it should subclass this method, but remember to call this super method.

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

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

telegram.ext.TypeHandler

class telegram.ext.TypeHandler(type, callback, strict=False, pass_update_queue=False, pass_job_queue=False)
Bases: telegram.ext.handler.Handler
Handler class to handle updates of custom types.

`type`
• `type` – The type of updates this handler should process.

`callback`
• `callable` – The callback function for this handler.

`strict`
• `bool` – Use `type` instead of `isinstance`. Default is `False`.

1.1. telegram.ext package
pass_update_queue
bool – Determines whether update_queue will be passed to the callback function.

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

Parameters

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

• callback (callable) – The callback function for this handler. Will be called when check_update has determined that an update should be processed by this handler. Callback signature for context based API:
def callback(update: Update, context: CallbackContext)
The return value of the callback is usually ignored except for the special case of telegram.ext.ConversationHandler.

• strict (bool, optional) – Use type instead of isinstance. Default is False

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

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

check_update(update)
Determines whether an update should be passed to this handlers callback.

Parameters


Returns bool

1.1.10 Persistence

telegram.ext.BasePersistence
class telegram.ext.BasePersistence (store_user_data=True, store_chat_data=True)
Bases: object

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

All relevant methods must be overwritten. This means:

• If store_chat_data is True you must overwrite get_chat_data() and update_chat_data().

• If store_user_data is True you must overwrite get_user_data() and update_user_data().

• If you want to store conversation data with telegram.ext.ConversationHandler, you must overwrite get_conversations() and update_conversation().

• flush() will be called when the bot is shutdown.

store_user_data
bool – Optional. Whether user_data should be saved by this persistence class.
store_chat_data
bool – Optional. Whether chat_data should be saved by this persistence class.

Parameters

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

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

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

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

Returns The restored chat data.
Return type defaultdict

get_conversations(name)
“Will be called by telegram.ext.Dispatcher when a telegram.ext.ConversationHandler is added if telegram.ext.ConversationHandler.persistent is True. It should return the conversations for the handler with name or an empty dict

Parameters name (str) – The handlers name.

Returns The restored conversations for the handler.
Return type dict

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

Returns The restored user data.
Return type defaultdict

update_chat_data(chat_id, data)
Will be called by the telegram.ext.Dispatcher after a handler has handled an update.

Parameters

• chat_id (int) – The chat the data might have been changed for.

• data (dict) – The :attr:`telegram.ext.dispatcher.chat_data'[user_id].

update_conversation(name, key, new_state)
Will be called when a telegram.ext.ConversationHandler.update_state is called. this allows the storage of the new state in the persistence.

Parameters

• name (str) – The handlers name.

• key (tuple) – The key the state is changed for.

• new_state (tuple|any) – The new state for the given key.

update_user_data(user_id, data)
Will be called by the telegram.ext.Dispatcher after a handler has handled an update.

Parameters
• **user_id**(int) – The user the data might have been changed for.

• **data**(dict) – The :attr:`telegram.ext.dispatcher.user_data`[user_id].

telegarm.ext.PicklePersistence

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

Bases: telegram.ext.basepersistence.BasePersistence

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

`filename`

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

`store_user_data`

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

`store_chat_data`

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

`single_file`

bool – Optional. When False will store 3 separate files of `filename_user_data`, `filename_chat_data` and `filename_conversations`. Default is True.

`on_flush`

bool, optional – When True will only save to file when `flush()` is called and keep data in memory until that happens. When False will store data on any transaction and on call for `flush()`. Default is False.

Parameters

• **filename**(str) – The filename for storing the pickle files. When `single_file` is false this will be used as a prefix.

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

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

• **single_file**(bool, optional) – When False will store 3 separate files of `filename_user_data`, `filename_chat_data` and `filename_conversations`. Default is True.

• **on_flush**(bool, optional) – When True will only save to file when `flush()` is called and keep data in memory until that happens. When False will store data on any transaction and on call for `flush()`. Default is False.

`flush()`

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

`get_chat_data()`

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

Returns The restored chat data.

Return type defaultdict

`get_conversations(name)`

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

Parameters name (str) – The handlers name.

Returns The restored conversations for the handler.
Return type: `dict`

**get_user_data()**

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

Returns: The restored user data.

**Return type: `defaultdict`**

**update_chat_data(chat_id, data)**

Will update the chat_data (if changed) and depending on `on_flush` save the pickle file.

**Parameters**

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

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

Will update the conversations for the given handler and depending on `on_flush` save the pickle file.

**Parameters**

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

**update_user_data(user_id, data)**

Will update the user_data (if changed) and depending on `on_flush` save the pickle file.

**Parameters**

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

### telegram.ext.DictPersistence

**class telegram.ext.DictPersistence (store_user_data=True, store_chat_data=True, user_data_json="", chat_data_json="", conversations_json="")**

**Bases:** `telegram.ext.basepersistence.BasePersistence`

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

**store_user_data**

bool – Whether user_data should be saved by this persistence class.

**store_chat_data**

bool – Whether chat_data should be saved by this persistence class.

**Parameters**

- `store_user_data (bool, optional)` – Whether user_data should be saved by this persistence class. Default is True.
- `store_chat_data (bool, optional)` – Whether user_data should be saved by this persistence class. Default is True.
- `user_data_json (str, optional)` – Json string that will be used to reconstruct user_data on creating this persistence. Default is ".
- `chat_data_json (str, optional)` – Json string that will be used to reconstruct chat_data on creating this persistence. Default is ".
- `conversations_json (str, optional)` – Json string that will be used to reconstruct conversation on creating this persistence. Default is ".
chat_data
dict – The chat_data as a dict

chat_data_json
str – The chat_data serialized as a JSON-string.

conversations
dict – The conversations as a dict

conversations_json
str – The conversations serialized as a JSON-string.

get_chat_data()
Returns the chat_data created from the chat_data_json or an empty defaultdict.

Returns The restored user data.

Return type defaultdict

get_conversations(name)
Returns the conversations created from the conversations_json or an empty defaultdict.

Returns The restored user data.

Return type defaultdict

get_user_data()
Returns the user_data created from the user_data_json or an empty defaultdict.

Returns The restored user data.

Return type defaultdict

update_chat_data(chat_id, data)
Will update the chat_data (if changed).

Parameters

• chat_id (int) – The chat the data might have been changed for.

• data (dict) – The :attr:`telegram.ext.dispatcher.chat_data`[chat_id].

update_conversation(name, key, new_state)
Will update the conversations for the given handler.

Parameters

• name (str) – The handlers name.

• key (tuple) – The key the state is changed for.

• new_state (tuple|any) – The new state for the given key.

update_user_data(user_id, data)
Will update the user_data (if changed).

Parameters

• user_id (int) – The user the data might have been changed for.

• data (dict) – The :attr:`telegram.ext.dispatcher.user_data`[user_id].

user_data
dict – The user_data as a dict

user_data_json
str – The user_data serialized as a JSON-string.
1.2 telegram.utils package

1.2.1 telegram.utils.helpers Module

This module contains helper functions.

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

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

**Parameters**

- **json_string (str)** – The conversations dict as JSON string.

**Returns**
The conversations dict after decoding

**Return type**
dict

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

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

**Parameters**

- **data (str)** – The user/chat_data dict as JSON string.

**Returns**
The user/chat_data defaultdict after decoding

**Return type**
dict

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

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

**Parameters**

- **entity (Union[telegram.Message, telegram.Update])** –

**Returns**
One of Message.MESSAGE_TYPES

**Return type**
str

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

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

**Parameters**

- **conversations (dict)** – The conversations dict to transform to JSON.

**Returns**
The JSON-serialized conversations dict

**Return type**
str

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

Helper function to escape telegram markup symbols.

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

Helper function to get a datetime.datetime from an int.

**Parameters**

- **unixtime (int)** –

**Returns**
datetime.datetime

**Return type**
datetime.datetime

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

Returns the signal name of the given signal number.

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

**Parameters**

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

**Returns**
The inline mention for the user as html.

**Return type**
str

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

**Parameters**

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

**Returns**
The inline mention for the user as markdown.

**Return type**
str
Parameters

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

Returns The inline mention for the user as markdown.

Return type str
telegram.utils.helpers.to_timestamp(dt_obj)

Parameters

- **dt_obj** (datetime.datetime) –

Returns

Return type int

### 1.2.2 telegram.utils.promise.Promise

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

Bases: object

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

Parameters

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

pooled_function

callable – The callable that will be called concurrently.

args

list|tuple – Positional arguments for *pooled_function*.

kwargs
dict – Keyword arguments for *pooled_function*.

done

threading.Event – Is set when the result is available.

exception

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

result (timeout=None)

Return the result of the Promise.

Parameters

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

Returns

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

Raises

Any exception raised by *pooled_function*.

run()

Calls the *pooled_function* callable.

### 1.2.3 telegram.utils.request.Request

class telegram.utils.request.Request(con_pool_size=1, proxy_url=None, urlllib3_proxy_kwargs=None, connect_timeout=5.0, read_timeout=5.0)

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

**Parameters**

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

**con_pool_size**
The size of the connection pool used.

**download (url, filename, timeout=None)**
Download a file by its URL.

**Parameters**

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

**get (url, timeout=None)**
Request an URL.

**Parameters**

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

**Returns**
A JSON object.

**post (url, data, timeout=None)**
Request an URL.

**Parameters**

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

**Returns**
A JSON object.

**retrieve (url, timeout=None)**
Retrieve the contents of a file by its URL.

**Parameters**

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

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

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

```python
file_id
str – Unique file identifier.
```

```python
width
int – Video width as defined by sender.
```

```python
height
int – Video height as defined by sender.
```

```python
duration
int – Duration of the video in seconds as defined by sender.
```

```python
thumb
telegram.PhotoSize – Optional. Animation thumbnail as defined by sender.
```

```python
file_name
str – Optional. Original animation filename as defined by sender.
```

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

```python
file_size
int – Optional. File size.
```

### Parameters

- `file_id` (str) – Unique file identifier.
- `width` (int) – Video width as defined by sender.
- `height` (int) – Video height as defined by sender.
- `duration` (int) – Duration of the video in seconds as defined by sender.
- `thumb` (telegram.PhotoSize, optional) – Animation thumbnail as defined by sender.
- `file_name` (str, optional) – Original animation filename as defined by sender.
- `mime_type` (str, optional) – MIME type of the file as defined by sender.
- `file_size` (int, optional) – File size.

### 1.4 telegram.Audio

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

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

```python
file_id
str – Unique identifier for this file.
```

```python
duration
int – Duration of the audio in seconds.
```

```python
performer
str – Optional. Performer of the audio as defined by sender or by audio tags.
```
title
   str – Optional. Title of the audio as defined by sender or by audio tags.

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

file_size
   int – Optional. File size.

thumb
   telegram.PhotoSize – Optional. Thumbnail of the album cover to which the music file belongs

bot

Parameters

* file_id (str) – Unique identifier for this file.
* duration (int) – Duration of the audio in seconds as defined by sender.
* performer (str, optional) – Performer of the audio as defined by sender or by audio tags.
* title (str, optional) – Title of the audio as defined by sender or by audio tags.
* mime_type (str, optional) – MIME type of the file as defined by sender.
* file_size (int, optional) – File size.
* thumb (telegram.PhotoSize, optional) – Thumbnail of the album cover to which the music file belongs
* bot (telegram.Bot, optional) – The Bot to use for instance methods.
* **kwargs (dict) – Arbitrary keyword arguments.

get_file (timeout=None, **kwargs)
Convenience wrapper over telegram.Bot.get_file

Parameters

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

Returns telegram.File

Raises telegram.TelegramError

1.5 telegram.Bot

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

Bases: telegram.base.TelegramObject

This object represents a Telegram Bot.

Parameters

* token (str) – Bot’s unique authentication.
* base_url (str, optional) – Telegram Bot API service URL.
* base_file_url (str, optional) – Telegram Bot API file URL.
• **request** (telegram.utils.request.Request, optional) – Pre initialized telegram.utils.request.Request.

• **private_key** (bytes, optional) – Private key for decryption of telegram passport data.

• **private_key_password** (bytes, optional) – Password for above private key.

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

Alias for add_sticker_to_set

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

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

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

Parameters

• **user_id** (int) – User identifier of created sticker set owner.

• **name** (str) – Sticker set name.

• **png_sticker** (str | filelike object) – Png image with the sticker, must be up to 512 kilobytes in size, dimensions must not exceed 512px, and either width or height must be exactly 512px. Pass a file_id as a String to send a file that already exists on the Telegram servers, pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data.

• **emojis** (str) – One or more emoji corresponding to the sticker.

• **mask_position** (telegram.MaskPosition, optional) – Position where the mask should be placed on faces.

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

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

Returns On success, True is returned.

Return type bool

Raises telegram.TelegramError

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

Alias for answer_callback_query

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

Alias for answer_inline_query

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

Alias for answer_pre_checkout_query

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

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

Use this method to send answers to callback queries sent from inline keyboards. The answer will be displayed to the user as a notification at the top of the chat screen or as an alert. Alternatively, the user can be redirected to the specified Game URL. For this option to work, you must first create a game for your bot via BotFather and accept the terms. Otherwise, you may use links like t.me/your_bot?start=XXXX that open your bot with a parameter.

Parameters

- `callback_query_id` (str) – Unique identifier for the query to be answered.
- `text` (str, optional) – Text of the notification. If not specified, nothing will be shown to the user, 0-200 characters.
- `show_alert` (bool, optional) – If true, an alert will be shown by the client instead of a notification at the top of the chat screen. Defaults to false.
- `url` (str, optional) – URL that will be opened by the user’s client. If you have created a Game and accepted the conditions via @Botfather, specify the URL that opens your game - note that this will only work if the query comes from a callback game button. Otherwise, you may use links like t.me/your_bot?start=XXXX that open your bot with a parameter.
- `cache_time` (int, optional) – The maximum amount of time in seconds that the result of the callback query may be cached client-side. Defaults to 0.
- `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- `**kwargs` (dict) – Arbitrary keyword arguments.

Returns bool On success, True is returned.

Raises telegram.TelegramError

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

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

Parameters

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

**kwargs (dict) – Arbitrary keyword arguments.

Example

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

Returns bool On success, True is returned.

Raises telegram.TelegramError

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

Once the user has confirmed their payment and shipping details, the Bot API sends the final confirmation in the form of an Update with the field pre_checkout_query. Use this method to respond to such pre-checkout queries.

Parameters

- pre_checkout_query_id (str) – Unique identifier for the query to be answered.
- ok (bool) – Specify True if everything is alright (goods are available, etc.) and the bot is ready to proceed with the order. Use False if there are any problems.
- error_message (str, optional) – Required if ok is False. Error message in human readable form that explains the reason for failure to proceed with the checkout (e.g. “Sorry, somebody just bought the last of our amazing black T-shirts while you were busy filling out your payment details. Please choose a different color or garment!”). Telegram will display this message to the user.
- timeout (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **kwargs (dict) – Arbitrary keyword arguments.

Returns On success, True is returned.

Return type bool

Raises telegram.TelegramError

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

If you sent an invoice requesting a shipping address and the parameter is_flexible was specified, the Bot API will send an Update with a shipping_query field to the bot. Use this method to reply to shipping queries.

Parameters

- shipping_query_id (str) – Unique identifier for the query to be answered.
• **ok** *(bool)* – Specify True if delivery to the specified address is possible and False if there are any problems (for example, if delivery to the specified address is not possible).

• **shipping_options** *(List[telegram.ShippingOption])* – Required if ok is True. A JSON-serialized array of available shipping options.

• **error_message** *(str, optional)* – Required if ok is False. Error message in human readable form that explains why it is impossible to complete the order (e.g. "Sorry, delivery to your desired address is unavailable"). Telegram will display this message to the user.

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

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

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

**Raises**  telegram.TelegramError

### createNewStickerSet

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

**Alias for** create_new_sticker_set

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

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

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

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

**Parameters**

• **user_id** *(int)* – User identifier of created sticker set owner.

• **name** *(str)* – Short name of sticker set, to be used in t.me/addstickers/ URLs (e.g., animals). Can contain only english letters, digits and underscores. Must begin with a letter, can't contain consecutive underscores and must end in "_<bot_username>". <bot_username> is case insensitive. 1-64 characters.

• **title** *(str)* – Sticker set title, 1-64 characters.

• **png_sticker** *(str | filelike object)* – Png image with the sticker, must be up to 512 kilobytes in size, dimensions must not exceed 512px, and either width or height must be exactly 512px. Pass a file_id as a String to send a file that already exists on the Telegram servers, pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data.

• **emojis** *(str)* – One or more emoji corresponding to the sticker.

• **contains_masks** *(bool, optional)* – Pass True, if a set of mask stickers should be created.

• **mask_position** *(telegram.MaskPosition, optional)* – Position where the mask should be placed on faces.

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

• ****kwargs** *(dict)* – Arbitrary keyword arguments.
Returns On success, True is returned.

Return type bool

Raises telegram.TelegramError

deleteChatPhoto(chat_id, timeout=None, **kwargs)
Alias for delete_chat_photo

deleteChatStickerSet(chat_id, timeout=None, **kwargs)
Alias for delete_chat_sticker_set

deleteMessage(chat_id, message_id, timeout=None, **kwargs)
Alias for delete_message

deleteStickerFromSet(sticker, timeout=None, **kwargs)
Alias for delete_sticker_from_set

deleteWebhook(timeout=None, **kwargs)
Alias for delete_webhook

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

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

Parameters

• chat_id (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

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

• **kwargs (dict) – Arbitrary keyword arguments

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

Returns Returns True on success.

Return type bool

Raises telegram.TelegramError

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

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

Parameters

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

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

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

Returns True on success.

Return type bool
**delete_message**(chat_id, message_id, timeout=None, **kwargs)

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

**Parameters**

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

**Returns**

On success, True is returned.

**Return type**

bool

**Raises**

telegram.TelegramError

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

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

**Parameters**

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

**Returns**

On success, True is returned.

**Return type**

bool

**Raises**

telegram.TelegramError

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

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

**Parameters**

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

**Returns**

bool On success, True is returned.

**Raise**

telegram.TelegramError

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

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

Alias for edit_message_live_location

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

Alias for edit_message_media

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

Alias for edit_message_reply_markup

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

Alias for edit_message_text

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

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

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **message_id** (int, optional) – Required if inline_message_id is not specified. Identifier of the sent message.
- **inline_message_id** (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.
- **caption** (str, optional) – New caption of the message.
- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
- **reply_markup** (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs (dict) – Arbitrary keyword arguments.

Returns

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

Return type

telegram.Message

Raises

telegram.TelegramError

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

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

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

- `chat_id (int | str)`: Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- `message_id (int, optional)`: Required if inline_message_id is not specified. Identifier of the sent message.
- `inline_message_id (str, optional)`: Required if chat_id and message_id are not specified. Identifier of the inline message.
- `latitude (float, optional)`: Latitude of location.
- `longitude (float, optional)`: Longitude of location.
- `location (telegram.Location, optional)`: The location to send.
- `reply_markup (telegram.ReplyMarkup, optional)`: Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- `timeout (int | float, optional)`: If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

**Returns**  On success the edited message.

**Return type**  `telegram.Message`

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

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

Parameters

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

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

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

Parameters
• chat_id (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• message_id (int, optional) – Required if inline_message_id is not specified. Identifier of the sent message.

• inline_message_id (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.

• reply_markup (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

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

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

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

Return type telegram.Message

Raises telegram.TelegramError

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

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

Parameters

• chat_id (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• message_id (int, optional) – Required if inline_message_id is not specified. Identifier of the sent message.

• inline_message_id (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.

• text (str) – New text of the message.

• parse_mode (str) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message. See the constants in telegram.ParseMode for the available modes.

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

• reply_markup (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

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

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

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

Return type telegram.Message

Raises telegram.TelegramError

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

Alias for export_chat_invite_link
**export_chat_invite_link** (*chat_id, timeout=None, **kwargs*)

Use this method to export an invite link to a supergroup or a channel. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

**Parameters**

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

**Returns**

Exported invite link on success.

**Return type**

str

**Raises**

telegram.TelegramError

**first_name**

str – Bot’s first name.

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

Alias for **forward_message**

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

Use this method to forward messages of any kind.

**Parameters**

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **from_chat_id** (int | str) – Unique identifier for the chat where the original message was sent (or channel username in the format @channelusername).
- **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- **message_id** (int) – Message identifier in the chat specified in from_chat_id.
- **timeout** (int | float, optional) – If this value is specified, use it as
- **read timeout** (the) – from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments

**Returns**

On success, the sent Message is returned.

**Return type**

telegram.Message

**Raises**

telegram.TelegramError

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

Alias for **get_chat**

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

Alias for **get_chat_administrators**

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

Alias for **get_chat_member**

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

Alias for **get_chat_members_count**
getFile(file_id, timeout=None, **kwargs)
   Alias for get_file

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

getMe(timeout=None, **kwargs)
   Alias for get_me

getStickerSet(name, timeout=None, **kwargs)
   Alias for get_sticker_set

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

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

getWebhookInfo(timeout=None, **kwargs)
   Alias for get_webhook_info

get_chat(chat_id, timeout=None, **kwargs)
   Use this method to get up to date information about the chat (current name of the user for one-on-one conversations, current username of a user, group or channel, etc.).

   Parameters

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

   Returns telegram.Chat

   Raises telegram.TelegramError

get_chat_administrators(chat_id, timeout=None, **kwargs)
   Use this method to get a list of administrators in a chat. On success, returns an Array of ChatMember objects that contains information about all chat administrators except other bots. If the chat is a group or a supergroup and no administrators were appointed, only the creator will be returned.

   Parameters

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

   Returns List[telegram.ChatMember]

   Raises telegram.TelegramError

get_chat_member(chat_id, user_id, timeout=None, **kwargs)
   Use this method to get information about a member of a chat.

   Parameters

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

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

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

Returns telegram.ChatMember

Raises telegram.TelegramError

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

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

Parameters

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

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

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

Returns Number of members in the chat.

Return type int

Raises telegram.TelegramError

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

Use this method to get basic info about a file and prepare it for downloading. For the moment, bots can download files of up to 20MB in size. The file can then be downloaded with telegram.File.download. It is guaranteed that the link will be valid for at least 1 hour. When the link expires, a new one can be requested by calling get_file again

Parameters


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

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

Returns telegram.File

Raises telegram.TelegramError

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

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

Parameters

• **user_id** (int) – User identifier.

• **chat_id** (int | str, optional) – Required if inline_message_id is not specified. Unique identifier for the target chat.

• **message_id** (int, optional) – Required if inline_message_id is not specified. Identifier of the sent message.
• **inline_message_id** (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.

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

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

Returns List[telegram.GameHighScore]

Raises telegram.TelegramError

get_me (timeout=None, **kwargs)

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

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

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

Return type telegram.User

Raises telegram.TelegramError

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

Use this method to get a sticker set.

Parameters

• **name** (str) – Short name of the sticker set that is used in t.me/addstickers/ URLs (e.g., animals)

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

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

Returns telegram.StickerSet

Raises telegram.TelegramError

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

Use this method to receive incoming updates using long polling.

Parameters

• **offset** (int, optional) – Identifier of the first update to be returned. Must be greater by one than the highest among the identifiers of previously received updates. By default, updates starting with the earliest unconfirmed update are returned. An update is considered confirmed as soon as getUpdates is called with an offset higher than its update_id. The negative offset can be specified to retrieve updates starting from -offset update from the end of the updates queue. All previous updates will forgotten.

• **limit** (int, optional) – Limits the number of updates to be retrieved. Values between 1-100 are accepted. Defaults to 100.

• **timeout** (int, optional) – Timeout in seconds for long polling. Defaults to 0, i.e. usual short polling. Should be positive, short polling should be used for testing purposes only.

• **allowed_updates** (List[str], optional) – List the types of updates you want your bot to receive. For example, specify ["message", “edited_channel_post”, “callback_query"] to only receive updates of these types. See telegram.Update for
a complete list of available update types. Specify an empty list to receive all updates regardless of type (default). If not specified, the previous setting will be used. Please note that this parameter doesn’t affect updates created before the call to the get_updates, so unwanted updates may be received for a short period of time.

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

Notes

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

Returns List[telegram.Update]

Raises telegram.TelegramError

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

Use this method to get a list of profile pictures for a user.

Parameters

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

Returns telegram.UserProfilePhotos

Raises telegram.TelegramError

get_webhook_info (timeout=None, **kwargs)

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

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

Parameters

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

Returns telegram.WebhookInfo

id

int – Unique identifier for this bot.

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

Alias for kick_chat_member

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

Use this method to kick a user from a group or a supergroup. In the case of supergroups, the user will not be able to return to the group on their own using invite links, etc., unless unbanned first. The bot must be an administrator in the group for this to work.

1.5. telegram.Bot

65
Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **user_id** (int) – Unique identifier of the target user.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **until_date** (int | datetime.datetime, optional) – Date when the user will be unbanned, unix time. If user is banned for more than 366 days or less than 30 seconds from the current time they are considered to be banned forever.
- **kwargs** (dict) – Arbitrary keyword arguments.

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

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

**Raises** telegram.TelegramError

---

**last_name**

str – Optional. Bot’s last name.

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

Alias for **leave_chat**

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

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

**Parameters**

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

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

**Raises** telegram.TelegramError

---

**name**

str – Bot’s @username.

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

Alias for **pin_chat_message**

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

Use this method to pin a message in a supergroup. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

**Parameters**

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

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

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

Returns: Returns **True** on success.

Return type: **bool**

Raises: **telegram.TelegramError**

**promoteChatMember** *(chat_id, user_id, can_change_info=None, can_post_messages=None, can_edit_messages=None, can_delete_messages=None, can_restrict_members=None, can_invite_users=None, can_pin_messages=None, can_promote_members=None, timeout=None, **kwargs)*

Alias for **promote_chat_member**

**promote_chat_member** *(chat_id, user_id, can_change_info=None, can_post_messages=None, can_edit_messages=None, can_delete_messages=None, can_invite_users=None, can_restrict_members=None, can_pin_messages=None, can_promote_members=None, timeout=None, **kwargs)*

Use this method to promote or demote a user in a supergroup or a channel. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights. Pass **False** for all boolean parameters to demote a user

Parameters:

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

• **user_id** *(int)* – Unique identifier of the target user.

• **can_change_info** *(bool, optional)* – Pass **True**, if the administrator can change chat title, photo and other settings.

• **can_post_messages** *(bool, optional)* – Pass **True**, if the administrator can create channel posts, channels only.

• **can_edit_messages** *(bool, optional)* – Pass **True**, if the administrator can edit messages of other users, channels only.

• **can_delete_messages** *(bool, optional)* – Pass **True**, if the administrator can delete messages of other users.

• **can_invite_users** *(bool, optional)* – Pass **True**, if the administrator can invite new users to the chat.

• **can_restrict_members** *(bool, optional)* – Pass **True**, if the administrator can restrict, ban or unban chat members.

• **can_pin_messages** *(bool, optional)* – Pass **True**, if the administrator can pin messages, supergroups only.

• **can_promote_members** *(bool, optional)* – Pass **True**, if the administrator can add new administrators with a subset of his own privileges or demote administrators that he has promoted, directly or indirectly (promoted by administrators that were appointed by him).

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

Returns
Returns True on success.

Return type
bool

Raises
telegram.TelegramError

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

Alias for **restrict_chat_member**

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

Use this method to restrict a user in a supergroup. The bot must be an administrator in the supergroup for this to work and must have the appropriate admin rights. Pass True for all boolean parameters to lift restrictions from a user.

Parameters

- **chat_id** (int \| str) – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername).
- **user_id** (int) – Unique identifier of the target user.
- **until_date** (int \| datetime.datetime, optional) – Date when restrictions will be lifted for the user, unix time. If user is restricted for more than 366 days or less than 30 seconds from the current time, they are considered to be restricted forever.
- **can_send_messages** (bool, optional) – Pass True, if the user can send text messages, contacts, locations and venues.
- **can_send_media_messages** (bool, optional) – Pass True, if the user can send audios, documents, photos, videos, video notes and voice notes, implies can_send_messages.
- **can_send_other_messages** (bool, optional) – Pass True, if the user can send animations, games, stickers and use inline bots, implies can_send_media_messages.
- **can_add_web_page_previews** (bool, optional) – Pass True, if the user may add web page previews to their messages, implies can_send_media_messages.
- **timeout** (int \| float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments

Returns
Returns True on success.

Return type
bool

Raises
telegram.TelegramError

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

Alias for send_animation

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

Alias for send_audio

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

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

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

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

sendInvoice (chat_id, title, description, payload, provider_token, start_parameter, currency, prices, photo_url=None, photo_size=None, photo_width=None, photo_height=None, need_name=None, need_phone_number=None, need_email=None, need_shipping_address=None, is_flexible=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, provider_data=None, send_phone_number_to_provider=None, send_email_to_provider=None, need_shipping_address=None, **kwargs)
Alias for send_invoice

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

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

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

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

sendSticker (chat_id, sticker, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, **kwargs)
Alias for send_sticker

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

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

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

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

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

**Parameters**

- **chat_id** *(int|str)* – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **animation** *(str|filelike object|telegram.Animation)* – Animation to send. Pass a file_id as String to send an animation that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an animation from the Internet, or upload a new animation using multipart/form-data. Lastly you can pass an existing telegram.Animation object to send.
- **duration** *(int, optional)* – Duration of sent animation in seconds.
- **width** *(int, optional)* – Animation width.
- **height** *(int, optional)* – Animation height.
- **thumb** *(filelike object, optional)* – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.
- **caption** *(str, optional)* – Animation caption (may also be used when resending animations by file_id), 0-1024 characters.
- **parse_mode** *(str, optional)* – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
- **disable_notification** *(bool, optional)* – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** *(int, optional)* – If the message is a reply, ID of the original message.
- **reply_markup** *(telegram.ReplyMarkup, optional)* – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** *(int|float, optional)* – Send file timeout (default: 20 seconds).
- ****kwargs**(dict)** – Arbitrary keyword arguments.

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

**Return type** telegram.Message

**Raises** telegram.TelegramError

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

Use this method to send audio files, if you want Telegram clients to display them in the music player. Your audio must be in the .mp3 format. On success, the sent Message is returned. Bots can currently send audio files of up to 50 MB in size, this limit may be changed in the future.

For sending voice messages, use the sendVoice method instead.

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

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

• `audio (str | filelike object | telegram.Audio)` – Audio file to send. Pass a file_id as a String to send an audio file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an audio file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing `telegram.Audio` object to send.

• `caption (str, optional)` – Audio caption, 0-1024 characters.

• `parse_mode (str, optional)` – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

• `duration (int, optional)` – Duration of sent audio in seconds.

• `performer (str, optional)` – Performer.

• `title (str, optional)` – Track name.

• `disable_notification (bool, optional)` – Sends the message silently. Users will receive a notification with no sound.

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

• `reply_markup (telegram.ReplyMarkup, optional)` – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

• `thumb (filelike object, optional)` – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not passed as a string or file_id.

• `timeout (int | float, optional)` – Send file timeout (default: 20 seconds).

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

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

### send_chat_action

Use this method when you need to tell the user that something is happening on the bot’s side. The status is set for 5 seconds or less (when a message arrives from your bot, Telegram clients clear its typing status).

Parameters

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

• `action (telegram.ChatAction | str)` – Type of action to broadcast. Choose one, depending on what the user is about to receive. For convenience look at the constants in `telegram.ChatAction`

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

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

Returns True on success.

Return type `bool`

Raises `telegram.TelegramError`
send_contact (chat_id, phone_number=None, first_name=None, last_name=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=None, contact=None, vcard=None, **kwargs)

Use this method to send phone contacts.

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

Parameters

- chat_id (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- phone_number (str, optional) – Contact’s phone number.
- first_name (str, optional) – Contact’s first name.
- last_name (str, optional) – Contact’s last name.
- vcard (str, optional) – Additional data about the contact in the form of a vCard, 0-2048 bytes.
- contact (telegram.Contact, optional) – The contact to send.
- disable_notification (bool, optional) – Sends the message silently. Users will receive a notification without sound.
- reply_to_message_id (int, optional) – If the message is a reply, ID of the original message.
- reply_markup (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- timeout (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **kwargs (dict) – Arbitrary keyword arguments.

Returns

On success, the sent Message is returned.

Return type telegram.Message

Raises telegram.TelegramError

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

Use this method to send general files.

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

Parameters

- chat_id (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- document (str | filelike object | telegram.Document) – File to send. Pass a file_id as String to send a file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload
a new one using multipart/form-data. Lastly you can pass an existing telegram.
Document object to send.

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

Returns  On success, the sent Message is returned.

Return type  telegram.Message

 Raises  telegram.TelegramError

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

Use this method to send a game.

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **game_short_name** (str) – Short name of the game, serves as the unique identifier for the game. Set up your games via Botfather.
- **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.
- **reply_markup** (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs** (dict) – Arbitrary keyword arguments.

Returns  On success, the sent Message is returned.

Return type  telegram.Message

 Raises  telegram.TelegramError
send_invoice(chat_id, title, description, payload, provider_token, start_parameter, currency, prices, photo_url=None, photo_size=None, photo_width=None, photo_height=None, need_name=None, need_phone_number=None, need_email=None, need_shipping_address=None, is_flexible=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, provider_data=None, send_phone_number_to_provider=None, send_email_to_provider=None, timeout=None, **kwargs)

Use this method to send invoices.

**Parameters**

- **chat_id** (`int` | `str`) – Unique identifier for the target private chat.
- **title** (`str`) – Product name.
- **description** (`str`) – Product description.
- **payload** (`str`) – Bot-defined invoice payload, 1-128 bytes. This will not be displayed to the user, use for your internal processes.
- **provider_token** (`str`) – Payments provider token, obtained via Botfather.
- **start_parameter** (`str`) – Unique deep-linking parameter that can be used to generate this invoice when used as a start parameter.
- **currency** (`str`) – Three-letter ISO 4217 currency code.
- **prices** (`List[telegram.LabeledPrice]`) – Price breakdown, a list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.).
- **provider_data** (`str` | `object`, optional) – JSON-encoded data about the invoice, which will be shared with the payment provider. A detailed description of required fields should be provided by the payment provider. When an object is passed, it will be encoded as JSON.
- **photo_url** (`str`, optional) – URL of the product photo for the invoice. Can be a photo of the goods or a marketing image for a service. People like it better when they see what they are paying for.
- **photo_size** (`str`, optional) – Photo size.
- **photo_width** (`int`, optional) – Photo width.
- **photo_height** (`int`, optional) – Photo height.
- **need_name** (`bool`, optional) – Pass True, if you require the user’s full name to complete the order.
- **need_phone_number** (`bool`, optional) – Pass True, if you require the user’s phone number to complete the order.
- **need_email** (`bool`, optional) – Pass True, if you require the user’s email to complete the order.
- **need_shipping_address** (`bool`, optional) – Pass True, if you require the user’s shipping address to complete the order.
- **send_phone_number_to_provider** (`bool`, optional) – Pass True, if user’s phone number should be sent to provider.
- **send_email_to_provider** (`bool`, optional) – Pass True, if user’s email address should be sent to provider.
- **is_flexible** (`bool`, optional) – Pass True, if the final price depends on the shipping method.
- **disable_notification** (`bool`, optional) – Sends the message silently. Users will receive a notification with no sound.
• reply_to_message_id (int, optional) – If the message is a reply, ID of the original message.

• reply_markup (telegram.ReplyMarkup, optional) – Additional interface options. An inline keyboard. If empty, one 'Pay total price' button will be shown. If not empty, the first button must be a Pay button.

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

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

Returns On success, the sent Message is returned.

Return type telegram.Message

Raises telegram.TelegramError

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

Use this method to send point on the map.

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

Parameters

• chat_id (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• latitude (float, optional) – Latitude of location.

• longitude (float, optional) – Longitude of location.

• location (telegram.Location, optional) – The location to send.

• live_period (int, optional) – Period in seconds for which the location will be updated, should be between 60 and 86400.

• disable_notification (bool, optional) – Sends the message silently. Users will receive a notification with no sound.

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

• reply_markup (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

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

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

Returns On success, the sent Message is returned.

Return type telegram.Message

Raises telegram.TelegramError

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

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

Parameters
• chat_id (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• media (List[telegram.InputMedia]) – An array describing photos and videos to be sent, must include 2–10 items.

• disable_notification (bool, optional) – Sends the message silently. Users will receive a notification with no sound.

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

• timeout (int | float, optional) – Send file timeout (default: 20 seconds).

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

Returns An array of the sent Messages.

Return type List[telegram.Message]

Raises telegram.TelegramError

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

Use this method to send text messages.

Parameters

• chat_id (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• text (str) – Text of the message to be sent. Max 4096 characters. Also found as telegram.constants.MAX_MESSAGE_LENGTH.

• parse_mode (str) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot’s message. See the constants in telegram.ParseMode for the available modes.

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

• disable_notification (bool, optional) – Sends the message silently. Users will receive a notification with no sound.

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

• reply_markup (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

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

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

Returns On success, the sent message is returned.

Return type telegram.Message

Raises telegram.TelegramError

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

Use this method to send photos.
**Note:** The photo argument can be either a file_id, an URL or a file from disk `open(filename, 'rb')`

**Parameters**

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

- **photo** (str | filelike object | telegram.PhotoSize) – Photo to send. Pass a file_id as String to send a photo that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a photo from the Internet, or upload a new photo using multipart/form-data. Lastly you can pass an existing telegram.PhotoSize object to send.

- **caption** (str, optional) – Photo caption (may also be used when resending photos by file_id), 0-1024 characters.

- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

- **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.

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

- **reply_markup** (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

- **timeout** (int | float, optional) – Send file timeout (default: 20 seconds).

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

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

**Return type** telegram.Message

**Raises** telegram.TelegramError

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

Use this method to send .webp stickers.

**Note:** The sticker argument can be either a file_id, an URL or a file from disk `open(filename, 'rb')`
• reply_to_message_id (int, optional) – If the message is a reply, ID of the original message.

• reply_markup (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

• timeout (int | float, optional) – Send file timeout (default: 20 seconds).

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

Returns On success, the sent Message is returned.

Return type telegram.Message

Raises telegram.TelegramError

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

Use this method to send information about a venue.

Note: you can either supply venue, or latitude, longitude, title and address and optionally foursquare_id and optionally foursquare_type.

Parameters

• chat_id (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• latitude (float, optional) – Latitude of venue.

• longitude (float, optional) – Longitude of venue.

• title (str, optional) – Name of the venue.

• address (str, optional) – Address of the venue.

• foursquare_id (str, optional) – Foursquare identifier of the venue.

• foursquare_type (str, optional) – Foursquare type of the venue, if known. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”.)

• venue (telegram.Venue, optional) – The venue to send.

• disable_notification (bool, optional) – Sends the message silently. Users will receive a notification with no sound.

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

• reply_markup (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

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

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

Returns On success, the sent Message is returned.

Return type telegram.Message

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

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

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

### Parameters

- **chat_id** *(int | str)* – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **video** *(str | filelike object | telegram.Video)* – Video file to send. Pass a file_id as String to send an video file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an video file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing `telegram.Video` object to send.
- **duration** *(int, optional)* – Duration of sent video in seconds.
- **width** *(int, optional)* – Video width.
- **height** *(int, optional)* – Video height.
- **caption** *(str, optional)* – Video caption (may also be used when resending videos by file_id), 0-1024 characters.
- **parse_mode** *(str, optional)* – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **supports_streaming** *(bool, optional)* – Pass True, if the uploaded video is suitable for streaming.
- **disable_notification** *(bool, optional)* – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id** *(int, optional)* – If the message is a reply, ID of the original message.
- **reply_markup** *(telegram.ReplyMarkup, optional)* – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **thumb** *(filelike object, optional)* – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.
- **timeout** *(int | float, optional)* – Send file timeout (default: 20 seconds).
- ****kwargs** *(dict)* – Arbitrary keyword arguments.

### Returns

On success, the sent Message is returned.

### Return type

`telegram.Message`

### Raises

`telegram.TelegramError`

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

Use this method to send video messages.
Note: The video_note argument can be either a file_id or a file from disk open(filename, 'rb')

Parameters

- **chat_id**(int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **video_note**(str | filelike object | telegram.VideoNote) – Video note to send. Pass a file_id as String to send a video note that exists on the Telegram servers (recommended) or upload a new video using multipart/form-data. Or you can pass an existing telegram.VideoNote object to send. Sending video notes by a URL is currently unsupported.
- **duration**(int, optional) – Duration of sent video in seconds.
- **length**(int, optional) – Video width and height
- **disable_notification**(bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- **reply_to_message_id**(int, optional) – If the message is a reply, ID of the original message.
- **reply_markup**(telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **thumb**(filelike object, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.
- **timeout**(int | float, optional) – Send file timeout (default: 20 seconds).
- ****kwargs**(dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type telegram.Message

Raises telegram.TelegramError

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

Use this method to send audio files, if you want Telegram clients to display the file as a playable voice message. For this to work, your audio must be in an .ogg file encoded with OPUS (other formats may be sent as Audio or Document).

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

Parameters

- **chat_id**(int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **voice**(str | filelike object | telegram.Voice) – Voice file to send. Pass a file_id as String to send an voice file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an voice file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing telegram.Voice object to send.
• **caption** (*str*, optional) – Voice message caption, 0-1024 characters.
• **parse_mode** (*str*, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.
• **duration** (*int*, optional) – Duration of the voice message in seconds.
• **disable_notification** (*bool*, optional) – Sends the message silently. Users will receive a notification with no sound.
• **reply_to_message_id** (*int*, optional) – If the message is a reply, ID of the original message.
• **reply_markup** (*telegram.ReplyMarkup*, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
• **timeout** (*int | float*, optional) – Send file timeout (default: 20 seconds).
• ****kwargs** (*dict*) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type *telegram.Message*

Raises *telegram.TelegramError*

**setChatDescription** (*chat_id*, *description*, *timeout=None, **kwargs*)
   
   Alias for *set_chat_description*

**setChatPhoto** (*chat_id*, *photo*, *timeout=20, **kwargs*)
   
   Alias for *set_chat_photo*

**setChatStickerSet** (*chat_id*, *sticker_set_name*, *timeout=None, **kwargs*)
   
   Alias for *set_chat_sticker_set*

**setChatTitle** (*chat_id*, *title*, *timeout=None, **kwargs*)
   
   Alias for *set_chat_title*

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

**setPassportDataErrors** (*user_id*, *errors*, *timeout=None, **kwargs*)
   
   Alias for *set_passport_data_errors*

**setStickerPositionInSet** (*sticker*, *position*, *timeout=None, **kwargs*)
   
   Alias for *set_sticker_position_in_set*

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

**set_chat_description** (*chat_id*, *description*, *timeout=None, **kwargs*)
   
   Use this method to change the description of a supergroup or a channel. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Parameters

• **chat_id** (*int | str*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
• **description** (*str*) – New chat description, 1-255 characters.
• **timeout** (*int | float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
• ****kwargs** (*dict*) – Arbitrary keyword arguments
Returns  Returns True on success.
Return type  bool
Raises  telegram.TelegramError

set_chat_photo (chat_id, photo, timeout=20, **kwargs)
Use this method to set a new profile photo for the chat.
Photos can’t be changed for private chats. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

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

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

Returns  Returns True on success.
Return type  bool
Raises  telegram.TelegramError

set_chat_sticker_set (chat_id, sticker_set_name, timeout=None, **kwargs)
Use this method to set a new group sticker set for a supergroup. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights. Use the field telegram.Chat.can_set_sticker_set optionally returned in get_chat requests to check if the bot can use this method.

Parameters
• chat_id (int | str) – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername).
• sticker_set_name (str) – Name of the sticker set to be set as the group sticker set.
• timeout (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
• **kwargs (dict) – Arbitrary keyword arguments.

Returns  True on success.
Return type  bool

set_chat_title (chat_id, title, timeout=None, **kwargs)
Use this method to change the title of a chat. Titles can’t be changed for private chats. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Parameters
• chat_id (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
• title (str) – New chat title, 1-255 characters.
• `timeout` *(int | float, optional)* – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

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

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

**Returns**  Returns `True` on success.

**Return type**  `bool`

**Raises**  `telegram.TelegramError`

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

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

**Parameters**

• `user_id` *(int)* – User identifier.

• `score` *(int)* – New score, must be non-negative.

• `force` *(bool, optional)* – Pass `True`, if the high score is allowed to decrease. This can be useful when fixing mistakes or banning cheaters.

• `disable_edit_message` *(bool, optional)* – Pass `True`, if the game message should not be automatically edited to include the current scoreboard.

• `chat_id` *(int|str, optional)* – Required if inline_message_id is not specified. Unique identifier for the target chat.

• `message_id` *(int, optional)* – Required if inline_message_id is not specified. Identifier of the sent message.

• `inline_message_id` *(str, optional)* – Required if chat_id and message_id are not specified. Identifier of the inline message.

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

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

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

**Return type**  `telegram.Message`

**Raises**

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

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

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

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

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

Returns On success, True is returned.

Return type bool

Raises telegram.TelegramError

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

Use this method to move a sticker in a set created by the bot to a specific position.

Parameters

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

Returns On success, True is returned.

Return type bool

Raises telegram.TelegramError

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

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

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

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

Parameters

- **url**(str) – HTTPS url to send updates to. Use an empty string to remove webhook integration.
- **certificate**(filelike) – Upload your public key certificate so that the root certificate in use can be checked. See our self-signed guide for details. (https://goo.gl/rw7w6Y)
- **max_connections**(int, optional) – Maximum allowed number of simultaneous HTTPS connections to the webhook for update delivery, 1-100. Defaults to 40. Use lower values to limit the load on your bot’s server, and higher values to increase your bot’s throughput.
• **allowed_updates** (List[str], optional) – List the types of updates you want your bot to receive. For example, specify "message", "edited_message", "call-back_query") to only receive updates of these types. See telegram.Update for a complete list of available update types. Specify an empty list to receive all updates regardless of type (default). If not specified, the previous setting will be used. Please note that this parameter doesn’t affect updates created before the call to the set_webhook, so unwanted updates may be received for a short period of time.

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

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

---

**Note:**

1. You will not be able to receive updates using get_updates for as long as an outgoing webhook is set up.

2. To use a self-signed certificate, you need to upload your public key certificate using certificate parameter. Please upload as InputFile, sending a String will not work.

3. Ports currently supported for Webhooks: 443, 80, 88, 8443.

---

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

**Raises**  telegram.TelegramError

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

Alias for **stop_message_live_location**

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

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

**Parameters**

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

• **message_id** (int, optional) – Required if inline_message_id is not specified. Identifier of the sent message.

• **inline_message_id** (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.

• **reply_markup** (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

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

**Returns**  On success the edited message.

**Return type**  telegram.Message

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

Alias for **unban_chat_member**
**unban_chat_member** (chat_id, user_id, timeout=None, **kwargs)

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

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

**Parameters**

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

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

**Raises**  telegram.TelegramError

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

Alias for **unpin_chat_message**

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

Use this method to unpin a message in a supergroup. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

**Parameters**

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

**Returns**  Returns True on success.

**Return type**  bool

**Raises**  telegram.TelegramError

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

Alias for **upload_sticker_file**

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

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

**Parameters**

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

__Note:__ The **png_sticker** argument can be either a file_id, an URL or a file from disk open(filename, 'rb')
**kwargs (dict) – Arbitrary keyword arguments.

Returns The uploaded File

Return type telegram.File

Raises telegram.TelegramError

username
str – Bot’s username.

1.6 telegram.CallbackQuery

class telegram.CallbackQuery(id, from_user, chat_instance, message=None, data=None, inline_message_id=None, game_short_name=None, bot=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents an incoming callback query from a callback button in an inline keyboard.

If the button that originated the query was attached to a message sent by the bot, the field message will be present. If the button was attached to a message sent via the bot (in inline mode), the field inline_message_id will be present.

Note:

- In Python `from` is a reserved word, use `from_user` instead.
- Exactly one of the fields `data` or `game_short_name` will be present.

id
str – Unique identifier for this query.

from_user
telegram.User – Sender.

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

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

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

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

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

Parameters

- id (str) – Unique identifier for this query.
- from_user (telegram.User) – Sender.
- message (telegram.Message, optional) – Message with the callback button that originated the query. Note that message content and message date will not be available if the message is too old.
- inline_message_id (str, optional) – Identifier of the message sent via the bot in inline mode, that originated the query.
• **chat_instance** *(str, optional) – Global identifier, uniquely corresponding to the chat to which the message with the callback button was sent. Useful for high scores in games.*

• **data** *(str, optional) – Data associated with the callback button. Be aware that a bad client can send arbitrary data in this field.*

• **game_short_name** *(str, optional) – Short name of a Game to be returned, serves as the unique identifier for the game*

**Note:** After the user presses an inline button, Telegram clients will display a progress bar until you call `answer`. It is, therefore, necessary to react by calling `telegram.Bot.answer_callback_query` even if no notification to the user is needed (e.g., without specifying any of the optional parameters).

```python
answer(*args, **kwargs)
```
Shortcut for:
```python
bot.answer_callback_query(update.callback_query.id, *args, **kwargs)
```

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

**Return type** `bool`

```python
edit_message_caption(caption, *args, **kwargs)
```
Shortcut for either:
```python
bot.edit_message_caption(caption=caption,
                         chat_id=update.callback_query.message.chat_id,
                         message_id=update.callback_query.message.message_id,
                         *args, **kwargs)
```

or:
```python
bot.edit_message_caption(caption=caption,
                         inline_message_id=update.callback_query.inline_message_id,
                         *args, **kwargs)
```

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

**Return type** `telegram.Message`

```python
edit_message_reply_markup(reply_markup, *args, **kwargs)
```
Shortcut for either:
```python
bot.edit_message_replyMarkup(chat_id=update.callback_query.message.chat_id,
                             message_id=update.callback_query.message.message_id,
                             reply_markup=reply_markup,
                             *args, **kwargs)
```

or:
```python
bot.edit_message_replyMarkup(inline_message_id=update.callback_query.inline_message_id,
                             reply_markup=reply_markup,
                             *args, **kwargs)
```
Returns On success, if edited message is sent by the bot, the edited Message is returned, otherwise True is returned.

Return type `telegram.Message`

`edit_message_text` (*text*, *args*, **kwargs)
Shortcut for either:

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

or:

```python
bot.edit_message_text(text, inline_message_id=update.callback_query.inline_message_id,
                      *args, **kwargs)
```

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

Return type `telegram.Message`

### 1.7 `telegram.Chat`

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

Bases: `telegram.base.TelegramObject`

This object represents a chat.

- **id**
  int – Unique identifier for this chat.

- **type**
  str – Type of chat.

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

- **username**
  str – Optional. Username.

- **first_name**
  str – Optional. First name of the other party in a private chat.

- **last_name**
  str – Optional. Last name of the other party in a private chat.

- **all_members_are_administrators**
  bool – Optional.

- **photo**

- **description**
  str – Optional. Description, for supergroups and channel chats.

- **invite_link**
  str – Optional. Chat invite link, for supergroups and channel chats.
pinned_message

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

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

Parameters

- **id** (int) – Unique identifier for this chat. This number may be greater than 32 bits and some programming languages may have difficulty/silent defects in interpreting it. But it is smaller than 52 bits, so a signed 64 bit integer or double-precision float type are safe for storing this identifier.
- **type** (str) – Type of chat, can be either ‘private’, ‘group’, ‘supergroup’ or ‘channel’.
- **title** (str, optional) – Title, for supergroups, channels and group chats.
- **username** (str, optional) – Username, for private chats, supergroups and channels if available.
- **first_name** (str, optional) – First name of the other party in a private chat.
- **last_name** (str, optional) – Last name of the other party in a private chat.
- **all_members_are_administrators** (bool, optional) – True if a group has All Members Are Admins enabled.
- **photo** (telegram.ChatPhoto, optional) – Chat photo. Returned only in getChat.
- **description** (str, optional) – Description, for supergroups and channel chats. Returned only in get_chat.
- **invite_link** (str, optional) – Chat invite link, for supergroups and channel chats. Returned only in get_chat.
- **pinned_message** (telegram.Message, optional) – Pinned message, for supergroups. Returned only in get_chat.
- **bot** (telegram.Bot, optional) – The Bot to use for instance methods.
- **sticker_set_name** (str, optional) – For supergroups, name of Group sticker set. Returned only in get_chat.
- **can_set_sticker_set** (bool, optional) – True, if the bot can change group the sticker set. Returned only in get_chat.
- ****kwargs** (dict) – Arbitrary keyword arguments.

 CHANNEL = 'channel'
 str – ‘channel’

GROUP = 'group'
 str – ‘group’

PRIVATE = 'private'
 str – ‘private’

SUPERGROUP = 'supergroup'
 str – ‘supergroup’

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

bot.get_chat_administrators(update.message.chat.id, *args, **kwargs)
Returns  A list of administrators in a chat. An Array of `telegram.ChatMember` objects that contains information about all chat administrators except other bots. If the chat is a group or a supergroup and no administrators were appointed, only the creator will be returned.

Return type  List[telegram.ChatMember]

```python
get_member(*args, **kwargs)
```

Shortcut for:

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

Returns  `telegram.ChatMember`

```python
get_members_count(*args, **kwargs)
```

Shortcut for:

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

Returns  `int`

```python
kick_member(*args, **kwargs)
```

Shortcut for:

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

Returns  If the action was sent succesfully.

Return type  `bool`

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

```python
leave(*args, **kwargs)
```

Shortcut for:

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

Returns  `bool` If the action was sent successfully.

```python
link
```

str – Convenience property. If the chat has a `username`, returns a t.me link of the chat.

```python
send_action(*args, **kwargs)
```

Shortcut for:

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

Returns  If the action was sent successfully.

Return type  `bool`

```python
send_animation(*args, **kwargs)
```

Shortcut for:

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

Where Chat is the current instance.
Returns On success, instance representing the message posted.

Return type telegram.Message

`send_audio`(*args, **kwargs)
Shortcut for:

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

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type telegram.Message

`send_document`(*args, **kwargs)
Shortcut for:

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

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type telegram.Message

`send_message`(*args, **kwargs)
Shortcut for:

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

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type telegram.Message

`send_photo`(*args, **kwargs)
Shortcut for:

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

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type telegram.Message

`send_sticker`(*args, **kwargs)
Shortcut for:

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

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type telegram.Message

`send_video`(*args, **kwargs)
Shortcut for:

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

Where Chat is the current instance.

Returns On success, instance representing the message posted.

Return type telegram.Message
**send_video_note**(*args, **kwargs*)

Shortcut for:

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

Where Chat is the current instance.

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

**Return type** `telegram.Message`

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

Shortcut for:

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

Where Chat is the current instance.

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

**Return type** `telegram.Message`

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

Shortcut for:

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

**Returns** If the action was sent successfully.

**Return type** `bool`

### 1.8 `telegram.ChatAction`

**class** `telegram.ChatAction`

**Bases:** object

Helper class to provide constants for different chatactions.

- **FIND_LOCATION** = 'find_location'
  
  str – 'find_location'

- **RECORD_AUDIO** = 'record_audio'
  
  str – 'record_audio'

- **RECORD_VIDEO** = 'record_video'
  
  str – 'record_video'

- **RECORD_VIDEO_NOTE** = 'record_video_note'
  
  str – 'record_video_note'

- **TYPING** = 'typing'
  
  str – 'typing'

- **UPLOAD_AUDIO** = 'upload_audio'
  
  str – 'upload_audio'

- **UPLOAD_DOCUMENT** = 'upload_document'
  
  str – 'upload_document'

- **UPLOAD_PHOTO** = 'upload_photo'
  
  str – 'upload_photo'

- **UPLOAD_VIDEO** = 'upload_video'
  
  str – 'upload_video'
UPLOAD_VIDEO_NOTE = 'upload_video_note'
str - 'upload_video_note'

1.9 telegram.ChatMember

class telegram.ChatMember(user, status, until_date=None, can_be_edited=None, can_change_info=None, can_post_messages=None, can_edit_messages=None, can_delete_messages=None, can_invite_users=None, can_restrict_members=None, can_pin_messages=None, can_promote_members=None, can_send_messages=None, can_send_media_messages=None, can_send_other_messages=None, can_add_web_page_previews=None, **kwargs)

Bases: telegram.base.TelegramObject

This object contains information about one member of the chat.

user
telegram.User – Information about the user.

status
str – The member's status in the chat.

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

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

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

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

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

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

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

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

can_pin_messages
bool – Optional. If the administrator can pin messages.

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

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

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

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

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

Parameters

- **user** (*telegram.User*) – Information about the user.
- **until_date** (*datetime.datetime*, optional) – Restricted and kicked only. Date when restrictions will be lifted for this user.
- **can_be_edited** (*bool*, optional) – Administrators only. True, if the bot is allowed to edit administrator privileges of that user.
- **can_change_info** (*bool*, optional) – Administrators only. True, if the administrator can change the chat title, photo and other settings.
- **can_post_messages** (*bool*, optional) – Administrators only. True, if the administrator can post in the channel, channels only.
- **can_edit_messages** (*bool*, optional) – Administrators only. True, if the administrator can edit messages of other users, channels only.
- **can_delete_messages** (*bool*, optional) – Administrators only. True, if the administrator can delete messages of other user.
- **can_invite_users** (*bool*, optional) – Administrators only. True, if the administrator can invite new users to the chat.
- **can_restrict_members** (*bool*, optional) – Administrators only. True, if the administrator can restrict, ban or unban chat members.
- **can_pin_messages** (*bool*, optional) – Administrators only. True, if the administrator can pin messages, supergroups only.
- **can_promote_members** (*bool*, optional) – Administrators only. True, if the administrator can add new administrators with a subset of his own privileges or demote administrators that he has promoted, directly or indirectly (promoted by administrators that were appointed by the user).
- **can_send_messages** (*bool*, optional) – Restricted only. True, if the user can send text messages, contacts, locations and venues.
- **can_send_media_messages** (*bool*, optional) – Restricted only. True, if the user can send audios, documents, photos, videos, video notes and voice notes, implies can_send_messages.
- **can_send_other_messages** (*bool*, optional) – Restricted only. True, if the user can send animations, games, stickers and use inline bots, implies can_send_media_messages.
- **can_add_web_page_previews** (*bool*, optional) – Restricted only. True, if user may add web page previews to his messages, implies can_send_media_messages.

```
ADMINISTRATOR = 'administrator'
str – 'administrator'

CREATOR = 'creator'
str – 'creator'

KICKED = 'kicked'
str – 'kicked'

LEFT = 'left'
str – 'left'
```
MEMBER = 'member'
str = 'member'

RESTRICTED = 'restricted'
str = 'restricted'

1.10 telegram.ChatPhoto

class telegram.ChatPhoto(small_file_id, big_file_id, bot=None, **kwargs)
    Bases: telegram.base.TelegramObject

This object represents a chat photo.

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

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

Parameters

    • small_file_id(str) – Unique file identifier of small (160x160) chat photo. This file_id can be used only for photo download.

    • big_file_id(str) – Unique file identifier of big (640x640) chat photo. This file_id can be used only for photo download.

    • bot(telegram.Bot, optional) – The Bot to use for instance methods

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

1.11 telegram.constants Module

Constants in the Telegram network.

The following constants were extracted from the Telegram Bots FAQ.

telegram.constants.MAX_MESSAGE_LENGTH
    int – 4096

telegram.constants.MAX.Caption_LENGTH
    int – 1024

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

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

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

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

telegram.constants.MAX_MESSAGES_PER_SECOND
    int – 30

telegram.constants.MAX_MESSAGES_PER_MINUTE_PER_GROUP
    int – 20

telegram.constants.MAX_INLINE_QUERY_RESULTS
    int – 50
The following constant have been found by experimentation:

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

1.12 telegram.Contact

class telegram.Contact (phone_number, first_name, last_name=None, user_id=None, vcard=None, **kwargs)
   Bases: telegram.base.TelegramObject

This object represents a phone contact.

phone_number
   str – Contact’s phone number.

first_name
   str – Contact’s first name.

last_name
   str – Optional. Contact’s last name.

user_id
   int – Optional. Contact’s user identifier in Telegram.

vcard
   str – Optional. Additional data about the contact in the form of a vCard.

Parameters

   • phone_number (str) – Contact’s phone number.
   • first_name (str) – Contact’s first name.
   • last_name (str, optional) – Contact’s last name.
   • user_id (int, optional) – Contact’s user identifier in Telegram.
   • vcard (str, optional) – Additional data about the contact in the form of a vCard.
   • **kwargs (dict) – Arbitrary keyword arguments.

1.13 telegram.Document

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

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

file_id
   str – Unique file identifier.

thumb

file_name
   str – Original filename.

mime_type
   str – Optional. MIME type of the file.

file_size
   int – Optional. File size.
bot


Parameters

• file_id (str) – Unique file identifier
• thumb (telegram.PhotoSize, optional) – Document thumbnail as defined by sender.
• file_name (str, optional) – Original filename as defined by sender.
• mime_type (str, optional) – MIME type of the file as defined by sender.
• file_size (int, optional) – File size.
• bot (telegram.Bot, optional) – The Bot to use for instance methods.
• **kwargs (dict) – Arbitrary keyword arguments.

get_file (timeout=None, **kwargs)

Convenience wrapper over telegram.Bot.get_file

Parameters

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

Returns telegram.File

Raises telegram.TelegramError

1.14 telegram.error module

This module contains an object that represents Telegram errors.

exception telegram.error.BadRequest (message)

Bases: telegram.error.NetworkError

exception telegram.error.ChatMigrated (new_chat_id)

Bases: telegram.error.TelegramError

Parameters new_chat_id (int) –

exception telegram.error.Conflict (msg)

Bases: telegram.error.TelegramError

Raised when a long poll or webhook conflicts with another one.

Parameters msg (str) – The message from telegrams server.

exception telegram.error.InvalidToken

Bases: telegram.error.TelegramError

exception telegram.error.NetworkError (message)

Bases: telegram.error.TelegramError

exception telegram.error.RetryAfter (retry_after)

Bases: telegram.error.TelegramError

Parameters retry_after (int) –

exception telegram.error.TelegramError (message)

Bases: Exception
exception telegram.error.TimedOut
    Bases: telegram.error.NetworkError

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

1.15 telegram.File

class telegram.File(file_id, bot=None, file_size=None, file_path=None, **kwargs)
    Bases: telegram.base.TelegramObject

This object represents a file ready to be downloaded. The file can be downloaded with download. It is
guaranteed that the link will be valid for at least 1 hour. When the link expires, a new one can be requested
by calling getFile.

Note: Maximum file size to download is 20 MB

file_id
    str – Unique identifier for this file.

file_size
    str – Optional. File size.

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

Parameters

  • file_id (str) – Unique identifier for this file.
  • file_size (int, optional) – Optional. File size, if known.
  • file_path (str, optional) – File path. Use download to get the file.
  • bot (telegram.Bot, optional) – Bot to use with shortcut method.
  • **kwargs (dict) – Arbitrary keyword arguments.

Note: If you obtain an instance of this class from telegram.PassportFile.get_file, then it will
automatically be decrypted as it downloads when you call download().

download(custom_path=None, out=None, timeout=None)
    Download this file. By default, the file is saved in the current working directory with its original
filename as reported by Telegram. If a custom_path is supplied, it will be saved to that path
instead. If out is defined, the file contents will be saved to that object using the out.write method.

Note: custom_path and out are mutually exclusive.

Parameters

  • custom_path (str, optional) – Custom path.
  • out (io.BufferedWriter, optional) – A file-like object. Must be opened for
writing in binary mode, if applicable.
  • timeout (int | float, optional) – If this value is specified, use it as the read
timeout from the server (instead of the one specified during creation of the connection pool).
Returns The same object as `out` if specified. Otherwise, returns the filename downloaded to.

Return type `str | io.BufferedWriter`

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

download_as_bytearray(buf=None)

Download this file and return it as a bytearray.

Parameters `buf` (bytearray, optional) – Extend the given bytearray with the downloaded data.

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

Return type `bytearray`

1.16 `telegram.ForceReply`

class `telegram.ForceReply` (`force_reply=True, selective=False, **kwargs`)

Bases: `telegram.replymarkup.ReplyMarkup`

Upon receiving a message with this object, Telegram clients will display a reply interface to the user (act as if the user has selected the bot’s message and tapped ‘Reply’). This can be extremely useful if you want to create user-friendly step-by-step interfaces without having to sacrifice privacy mode.

force_reply

True – Shows reply interface to the user.

selective

bool – Optional. Force reply from specific users only.

Parameters

- `selective` (bool, optional) – Use this parameter if you want to force reply from specific users only. Targets:
  1. users that are @mentioned in the text of the Message object
  2. if the bot’s message is a reply (has reply_to_message_id), sender of the original message.

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

1.17 `telegram.InlineKeyboardButton`

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

Bases: `telegram.base.TelegramObject`

This object represents one button of an inline keyboard.

Note: You must use exactly one of the optional fields. Mind that `callback_game` is not working as expected. Putting a game short name in it might, but is not guaranteed to work.

- `text` (str) – Label text on the button.
url
str – Optional. HTTP url to be opened when button is pressed.

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

switch_inline_query
str – Optional. Will prompt the user to select one of their chats, open that chat and insert the bot’s username and the specified inline query in the input field.

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

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

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

Parameters

- **text**(str) – Label text on the button.
- **url**(str) – HTTP url to be opened when button is pressed.
- **callback_data**(str, optional) – Data to be sent in a callback query to the bot when button is pressed, 1-64 bytes.
- **switch_inline_query**(str, optional) – If set, pressing the button will prompt the user to select one of their chats, open that chat and insert the bot’s username and the specified inline query in the input field. Can be empty, in which case just the bot’s username will be inserted. This offers an easy way for users to start using your bot in inline mode when they are currently in a private chat with it. Especially useful when combined with switch_pm* actions - in this case the user will be automatically returned to the chat they switched from, skipping the chat selection screen.
- **switch_inline_query_current_chat**(str, optional) – If set, pressing the button will insert the bot’s username and the specified inline query in the current chat’s input field. Can be empty, in which case only the bot’s username will be inserted. This offers a quick way for the user to open your bot in inline mode in the same chat - good for selecting something from multiple options.
- **callback_game**(telegram.CallbackGame, optional) – Description of the game that will be launched when the user presses the button. This type of button must always be the first button in the first row.
- **pay**(bool, optional) – Specify True, to send a Pay button. This type of button must always be the first button in the first row.
- **kwargs**(dict) – Arbitrary keyword arguments.

1.18 telegram.InlineKeyboardMarkup

class telegram.InlineKeyboardMarkup(inline_keyboard, **kwargs)
Bases: telegram.replymarkup.ReplyMarkup

This object represents an inline keyboard that appears right next to the message it belongs to.

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

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

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

classmethod `from_button` (`button`, `**kwargs`)
Shortcut for:

```python
InlineKeyboardMarkup([[button]], **kwargs)
```

Return an InlineKeyboardMarkup from a single InlineKeyboardButton

Parameters

• `button` (telegram.InlineKeyboardButton) – The button to use in the markup

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

classmethod `from_column` (`button_column`, `**kwargs`)
Shortcut for:

```python
InlineKeyboardMarkup([[button] for button in button_column], **kwargs)
```

Return an InlineKeyboardMarkup from a single column of InlineKeyboardButtons

Parameters

• `button_column` (List[telegram.InlineKeyboardButton]) – The button to use in the markup

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

classmethod `from_row` (`button_row`, `**kwargs`)
Shortcut for:

```python
InlineKeyboardMarkup([button_row], **kwargs)
```

Return an InlineKeyboardMarkup from a single row of InlineKeyboardButtons

Parameters

• `button_row` (List[telegram.InlineKeyboardButton]) – The button to use in the markup

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

1.19 telegram.InputFile

class `telegram.InputFile` (`obj`, `filename=None`, `attach=None`)
Bases: object

This object represents a Telegram InputFile.

`input_file_content`

`bytes` – The binary content of the file to send.

`filename`

`str` – Optional, Filename for the file to be sent.

`attach`

`str` – Optional, attach id for sending multiple files.

Parameters
• obj (File handler) – An open file descriptor.
• filename (str, optional) – Filename for this InputFile.
• attach (bool, optional) – Whether this should be send as one file or is part of a collection of files.

Raises TelegramError

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

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

Returns The str mime-type of an image.

Return type str

1.20 telegram.InputMedia

class telegram.InputMedia
    Bases: telegram.base.TelegramObject

Base class for Telegram InputMedia Objects.


1.21 telegram.InputMediaAnimation

class telegram.InputMediaAnimation
    Bases: telegram.files.inputmedia.InputMedia

Represents an animation file (GIF or H.264/MPEG-4 AVC video without sound) to be sent.

    type str – animation.

    media str – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing telegram.Animation object to send.

    thumb filelike object – Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

    caption str – Optional. Caption of the animation to be sent, 0-1024 characters.

    parse_mode str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

    width int – Optional. Animation width.
height
int – Optional. Animation height.

duration
int – Optional. Animation duration.

Parameters

- **media** (str) – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.Animation` object to send.

- **thumb** (file-like object, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

- **caption** (str, optional) – Caption of the animation to be sent, 0-1024 characters.

- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

- **width** (int, optional) – Animation width.

- **height** (int, optional) – Animation height.

- **duration** (int, optional) – Animation duration.

---

**Note:** When using a `telegram.Animation` for the `media` attribute. It will take the width, height and duration from that video, unless otherwise specified with the optional arguments.

### 1.22 telegram.InputMediaAudio

class `telegram.InputMediaAudio`(media, thumb=None, caption=None, parse_mode=None, duration=None, performer=None, title=None)

**Bases:** `telegram.files.inputmedia.InputMedia`

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

**type**
str – audio.

**media**
str – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.Audio` object to send.

**caption**
str – Optional. Caption of the audio to be sent, 0-1024 characters.

**parse_mode**
str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

**duration**
int – Duration of the audio in seconds.

**performer**
str – Optional. Performer of the audio as defined by sender or by audio tags.

**title**
str – Optional. Title of the audio as defined by sender or by audio tags.
thumb

filelike object – Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not passed as a string or file_id.

Parameters

- **media** (str) – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing telegram.Document object to send.
- **caption** (str, optional) – Caption of the audio to be sent, 0-1024 characters.
- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
- **duration** (int) – Duration of the audio in seconds as defined by sender.
- **performer** (str, optional) – Performer of the audio as defined by sender or by audio tags.
- **title** (str, optional) – Title of the audio as defined by sender or by audio tags.
- **thumb** (filelike object, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not passed as a string or file_id.

Note: When using a telegram.Audio for the media attribute. It will take the duration, performer and title from that video, unless otherwise specified with the optional arguments.

1.23 telegram.InputMediaDocument

class telegram.InputMediaDocument (media, thumb=None, caption=None, parse_mode=None)

Bases: telegram.files.inputmedia.InputMedia

Represents a general file to be sent.

type

str – document.

media

str – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing telegram.Document object to send.

caption

str – Optional. Caption of the document to be sent, 0-1024 characters.

parse_mode

str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

thumb

filelike object – Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not passed as a string or file_id.

Parameters
• **media** (*str*) – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.Document` object to send.

• **caption** (*str, optional*) – Caption of the document to be sent, 0-1024 characters.

• **parse_mode** (*str, optional*) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

• **thumb** (*filelike object, optional*) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

1.24 `telegram.InputMediaPhoto`

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

    Bases: `telegram.files.inputmedia.InputMedia`

    Represents a photo to be sent.

    **type**
    
    *str* – photo.

    **media**
    
    *str* – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.PhotoSize` object to send.

    **caption**
    
    *str* – Optional. Caption of the photo to be sent, 0-1024 characters.

    **parse_mode**
    
    *str* – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

1.25 `telegram.InputMediaVideo`

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

    Bases: `telegram.files.inputmedia.InputMedia`

    Represents a video to be sent.

    **type**
    
    *str* – video.
media
str – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.Video` object to send.

caption
str – Optional. Caption of the video to be sent, 0-1024 characters.

parse_mode
str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

width
int – Optional. Video width.

height
int – Optional. Video height.

duration
int – Optional. Video duration.

supports_streaming
bool – Optional. Pass True, if the uploaded video is suitable for streaming.

thumb
filelike object – Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

Parameters

- **media** (str) – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.Video` object to send.

- **caption** (str, optional) – Caption of the video to be sent, 0-1024 characters.

- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

- **width** (int, optional) – Video width.

- **height** (int, optional) – Video height.

- **duration** (int, optional) – Video duration.

- **supports_streaming** (bool, optional) – Pass True, if the uploaded video is suitable for streaming.

- **thumb** (filelike object, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

Note: When using a `telegram.Video` for the **media** attribute. It will take the width, height and duration from that video, unless otherwise specified with the optional arguments.
1.26 `telegram.KeyboardButton`

```python
class telegram.KeyboardButton(text, request_contact=None, request_location=None, **kwargs)
```

**Bases:** `telegram.base.TelegramObject`

This object represents one button of the reply keyboard. For simple text buttons String can be used instead of this object to specify text of the button.

**Note:** Optional fields are mutually exclusive.

- `text` : `str` – Text of the button.
- `request_contact` : `bool` – Optional. If the user’s phone number will be sent.
- `request_location` : `bool` – Optional. If the user’s current location will be sent.

**Parameters**

- `text` *(str)* – Text of the button. If none of the optional fields are used, it will be sent to the bot as a message when the button is pressed.
- `request_contact` *(bool, optional)* – If True, the user’s phone number will be sent as a contact when the button is pressed. Available in private chats only.
- `request_location` *(bool, optional)* – If True, the user’s current location will be sent when the button is pressed. Available in private chats only.

**Note:** `request_contact` and `request_location` options will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

1.27 `telegram.Location`

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

**Bases:** `telegram.base.TelegramObject`

This object represents a point on the map.

- `longitude` : `float` – Longitude as defined by sender.
- `latitude` : `float` – Latitude as defined by sender.

**Parameters**

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

class telegram.Message(message_id, from_user, date, chat, forward_from=None, forward_from_chat=None, forward_from_message_id=None, forward_date=None, reply_to_message=None, edit_date=None, text=None, entities=None, caption_entities=None, audio=None, document=None, game=None, photo=None, sticker=None, video=None, voice=None, video_note=None, new_chat_members=None, caption=None, location=None, venue=None, left_chat_member=None, new_chat_title=None, new_chat_photo=None, delete_chat_photo=False, group_chat_created=False, supergroup_chat_created=False, channel_chat_created=False, migrate_to_chat_id=None, migrate_from_chat_id=None, pinned_message=None, invoice=None, successful_payment=None, forward_signature=None, author_signature=None, media_group_id=None, connected_website=None, animation=None, passport_data=None, bot=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents a message.

Note:

- In Python from is a reserved word, use from_user instead.

message_id

int – Unique message identifier inside this chat.

from_user


date

datetime.datetime – Date the message was sent.

chat

telegram.Chat – Conversation the message belongs to.

forward_from

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

forward_from_chat

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

forward_from_message_id

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

forward_date

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

reply_to_message

telegram.Message – Optional. The original message.

edit_date

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

media_group_id

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

text

str – Optional. The actual UTF-8 text of the message.
entities
   List[telegram.MessageEntity] – Optional. Special entities like usernames, URLs, bot commands, etc. that appear in the text. See Message.parse_entity and parse_entities methods for how to use properly.

caption_entities
   List[telegram.MessageEntity] – Optional. Special entities like usernames, URLs, bot commands, etc. that appear in the caption. See Message.parse_caption_entity and parse_caption_entities methods for how to use properly.

audio
document

animation
telegram.Animation – For backward compatibility, when this field is set, the document field will also be set.
game
telegram.Game – Optional. Information about the game.

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

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

voice

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

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

caption
   str – Optional. Caption for the document, photo or video, 0-1024 characters.

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

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

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

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

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

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

delete_chat_photo
   bool – Optional. The chat photo was deleted.
group_chat_created
  bool – Optional. The group has been created.

supergroup_chat_created
  bool – Optional. The supergroup has been created.

channel_chat_created
  bool – Optional. The channel has been created.

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

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

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

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

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

connected_website
  str – Optional. The domain name of the website on which the user has logged in.

forward_signature
  str – Optional. Signature of the post author for messages forwarded from channels.

author_signature
  str – Optional. Signature of the post author for messages in channels.

passport_data
  telegram.PassportData – Optional. Telegram Passport data

bot

Parameters

- **message_id** (int) – Unique message identifier inside this chat.
- **from_user** (telegram.User, optional) – Sender, can be empty for messages sent to channels.
- **date** (datetime.datetime) – Date the message was sent in Unix time. Converted to datetime.datetime.
- **chat** (telegram.Chat) – Conversation the message belongs to.
- **forward_from** (telegram.User, optional) – For forwarded messages, sender of the original message.
- **forward_from_chat** (telegram.Chat, optional) – For messages forwarded from a channel, information about the original channel.
- **forward_from_message_id**(int, optional) – For forwarded channel posts, identifier of the original message in the channel.
- **forward_date** (datetime.datetime, optional) – For forwarded messages, date the original message was sent in Unix time. Converted to datetime.datetime.
- **reply_to_message** (telegram.Message, optional) – For replies, the original message. Note that the Message object in this field will not contain further reply_to_message fields even if it itself is a reply.
- **edit_date** (datetime.datetime, optional) – Date the message was last edited in Unix time. Converted to datetime.datetime.
media_group_id (str, optional) – The unique identifier of a media message group this message belongs to.

text (str, optional) – For text messages, the actual UTF-8 text of the message, 0-4096 characters. Also found as telegram.constants.MAX_MESSAGE_LENGTH.

etories (List[telegram.MessageEntity], optional) – For text messages, special entities like usernames, URLs, bot commands, etc. that appear in the text. See attr:parse_entity and attr:parse_entities methods for how to use properly.

caption_entities (List[telegram.MessageEntity]) – Optional. For Messages with a Caption. Special entities like usernames, URLs, bot commands, etc. that appear in the caption. See Message.parse_caption_entity and parse_caption_entities methods for how to use properly.

audio (telegram.Audio, optional) – Message is an audio file, information about the file.


animation (telegram.Animation, optional) – Message is an animation, information about the animation. For backward compatibility, when this field is set, the document field will also be set.

game (telegram.Game, optional) – Message is a game, information about the game.

photo (List[telegram.PhotoSize], optional) – Message is a photo, available sizes of the photo.

sticker (telegram.Sticker, optional) – Message is a sticker, information about the sticker.

video (telegram.Video, optional) – Message is a video, information about the video.

voice (telegram.Voice, optional) – Message is a voice message, information about the file.

video_note (telegram.VideoNote, optional) – Message is a video note, information about the video message.

new_chat_members (List[telegram.User], optional) – New members that were added to the group or supergroup and information about them (the bot itself may be one of these members).

caption (str, optional) – Caption for the document, photo or video, 0-1024 characters.

contact (telegram.Contact, optional) – Message is a shared contact, information about the contact.

location (telegram.Location, optional) – Message is a shared location, information about the location.

venue (telegram.Venue, optional) – Message is a venue, information about the venue.

left_chat_member (telegram.User, optional) – A member was removed from the group, information about them (this member may be the bot itself).

new_chat_title (str, optional) – A chat title was changed to this value.

new_chat_photo (List[telegram.PhotoSize], optional) – A chat photo was change to this value.

delete_chat_photo (bool, optional) – Service message: The chat photo was deleted.
• **group_chat_created** (bool, optional) – Service message: The group has been created.

• **supergroup_chat_created** (bool, optional) – Service message: The supergroup has been created. This field can’t be received in a message coming through updates, because bot can’t be a member of a supergroup when it is created. It can only be found in *reply_to_message* if someone replies to a very first message in a directly created supergroup.

• **channel_chat_created** (bool, optional) – Service message: The channel has been created. This field can’t be received in a message coming through updates, because bot can’t be a member of a channel when it is created. It can only be found in *reply_to_message* if someone replies to a very first message in a channel.

• **migrate_to_chat_id** (int, optional) – The group has been migrated to a supergroup with the specified identifier. This number may be greater than 32 bits and some programming languages may have difficulty/silent defects in interpreting it. But it is smaller than 52 bits, so a signed 64 bit integer or double-precision float type are safe for storing this identifier.

• **migrate_from_chat_id** (int, optional) – The supergroup has been migrated from a group with the specified identifier. This number may be greater than 32 bits and some programming languages may have difficulty/silent defects in interpreting it. But it is smaller than 52 bits, so a signed 64 bit integer or double-precision float type are safe for storing this identifier.

• **pinned_message** (telegram.message, optional) – Specified message was pinned. Note that the Message object in this field will not contain further *reply_to_message* fields even if it is itself a reply.

• **invoice** (telegram.Invoice, optional) – Message is an invoice for a payment, information about the invoice.

• **successful_payment** (telegram.SuccessfulPayment, optional) – Message is a service message about a successful payment, information about the payment.

• **connected_website** (str, optional) – The domain name of the website on which the user has logged in.

• **forward_signature** (str, optional) – Signature of the post author for messages forwarded from channels.

• **author_signature** (str, optional) – Signature of the post author for messages in channels.

• **passport_data** (telegram.PassportData, optional) – Telegram Passport data

### caption_html

Creates an HTML-formatted string from the markup entities found in the message’s caption.

Use this if you want to retrieve the message caption with the caption entities formatted as HTML in the same way the original message was formatted.

**Returns**  
Message caption with captionentities formatted as HTML.

**Return type**  
str

### caption_html_urlencoded

Creates an HTML-formatted string from the markup entities found in the message’s caption.

Use this if you want to retrieve the message caption with the caption entities formatted as HTML. This also formats *telegram.MessageEntity.URL* as a hyperlink.

**Returns**  
Message caption with caption entities formatted as HTML.

**Return type**  
str
caption_markdown
Creates an Markdown-formatted string from the markup entities found in the message’s caption.
Use this if you want to retrieve the message caption with the caption entities formatted as Markdown
in the same way the original message was formatted.

Returns Message caption with caption entities formatted as Markdown.
Return type str

caption_markdown_url
caption_markdown_url
Creates an Markdown-formatted string from the markup entities found in the message’s caption.
Use this if you want to retrieve the message caption with the caption entities formatted as Markdown.
This also formats `telegram.MessageEntity.URL` as a hyperlink.

Returns Message caption with caption entities formatted as Markdown.
Return type str

chat_id

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

```python
bot.delete_message(chat_id=message.chat_id,
                   message_id=message.message_id,
                   *args,
                   **kwargs)
```

Returns On success, True is returned.
Return type bool

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

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

Note: You can only edit messages that the bot sent itself, therefore this method can only be used on
the return value of the `bot.send_*` family of methods.

Returns On success, instance representing the edited message.
Return type `telegram.Message`

edit_media(media, *args, **kwargs)
Shortcut for:

```python
bot.edit_message_media(chat_id=message.chat_id,
                       message_id=message.message_id,
                       *args,
                       **kwargs)
```

Note: You can only edit messages that the bot sent itself, therefore this method can only be used on
the return value of the `bot.send_*` family of methods.
Returns On success, instance representing the edited message.

Return type telegram.Message

edit_reply_markup(*args, **kwargs)

Shortcut for:

```python
bot.edit_message_reply_markup(chat_id=message.chat_id,
message_id=message.message_id,
*args,
**kwargs)
```

Note: You can only edit messages that the bot sent itself, therefore this method can only be used on the return value of the `bot.send_*` family of methods.

Returns On success, instance representing the edited message.

Return type telegram.Message

edit_text(*args, **kwargs)

Shortcut for:

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

Note: You can only edit messages that the bot sent itself, therefore this method can only be used on the return value of the `bot.send_*` family of methods.

Returns On success, instance representing the edited message.

Return type telegram.Message

effective_attachment

- telegram.Audio or telegram.Contact or telegram.Document or telegram.Animation or telegram.Game or telegram.Invoice or telegram.Location or List[telegram.PhotoSize] or telegram.Sticker or telegram.SuccessfulPayment or telegram.Venue or telegram.Video or telegram.VideoNote or telegram.Voice: The attachment that this message was sent with. May be None if no attachment was sent.

forward(chat_id, disable_notification=False)

Shortcut for:

```python
bot.forward_message(chat_id=chat_id,
from_chat_id=update.message.chat_id,
disable_notification=disable_notification,
message_id=update.message.message_id)
```

Returns On success, instance representing the message forwarded.

Return type telegram.Message

link

- str – Convenience property. If the chat of the message is a supergroup or a channel and has a `Chat.username`, returns a t.me link of the message.
parse_caption_entities(types=None)

Returns a dict that maps telegram.MessageEntity to str. It contains entities from this message’s caption filtered by their telegram.MessageEntity.type attribute as the key, and the text that each entity belongs to as the value of the dict.

**Note:** This method should always be used instead of the caption_entities attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See parse_entity for more info.

**Parameters**
- **types** (List[str], optional) – List of telegram.MessageEntity types as strings. If the type attribute of an entity is contained in this list, it will be returned. Defaults to a list of all types. All types can be found as constants in telegram.MessageEntity.

**Returns**
A dictionary of entities mapped to the text that belongs to them, calculated based on UTF-16 codepoints.

**Return type**
Dict[telegram.MessageEntity, str]

parse_caption_entity(entity)

Returns the text from a given telegram.MessageEntity.

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

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

**Returns**
The text of the given entity

**Return type**
str

parse_entities(types=None)

Returns a dict that maps telegram.MessageEntity to str. It contains entities from this message filtered by their telegram.MessageEntity.type attribute as the key, and the text that each entity belongs to as the value of the dict.

**Note:** This method should always be used instead of the entities attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See parse_entity for more info.

**Parameters**
- **types** (List[str], optional) – List of telegram.MessageEntity types as strings. If the type attribute of an entity is contained in this list, it will be returned. Defaults to a list of all types. All types can be found as constants in telegram.MessageEntity.

**Returns**
A dictionary of entities mapped to the text that belongs to them, calculated based on UTF-16 codepoints.

**Return type**
Dict[telegram.MessageEntity, str]
parse_entity(entity)
Returns the text from a given telegram.MessageEntity.

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

Parameters

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

Returns The text of the given entity
Return type str

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

```
bot.send_animation(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments

quote (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.
Return type telegram.Message

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

```
bot.send_audio(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments

quote (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.
Return type telegram.Message

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

```
bot.send_contact(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments

quote (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.
Return type telegram.Message

reply_document(*args, **kwargs)
Shortcut for:
bot.send_document(update.message.chat_id, *args, **kwargs)

**Keyword Arguments**

- **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

**Returns**

On success, instance representing the message posted.

**Return type** `telegram.Message`

### reply_html(*args, **kwargs)

Shortcut for:

```python
bot.send_message(update.message.chat_id, parse_mode=ParseMode.HTML, *args, **kwargs)
```

Sends a message with HTML formatting.

**Keyword Arguments**

- **quote** (bool, optional) – If set to True, the message is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

### reply_location(*args, **kwargs)

Shortcut for:

```python
bot.send_location(update.message.chat_id, *args, **kwargs)
```

**Keyword Arguments**

- **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

**Returns**

On success, instance representing the message posted.

**Return type** `telegram.Message`

### reply_markdown(*args, **kwargs)

Shortcut for:

```python
bot.send_message(update.message.chat_id, parse_mode=ParseMode.MARKDOWN, *args, **kwargs)
```

Sends a message with markdown formatting.

**Keyword Arguments**

- **quote** (bool, optional) – If set to True, the message is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

### reply_media_group(*args, **kwargs)

Shortcut for:

```python
bot.reply_media_group(update.message.chat_id, *args, **kwargs)
```

**Keyword Arguments**

- **quote** (bool, optional) – If set to True, the media group is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

**Returns**

An array of the sent Messages.

**Return type** `List[telegram.Message]`

**Raises** `telegram.TelegramError`
**reply_photo** (*args, **kwargs*)

Shortcut for:

```python
bot.send_photo(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments

- **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

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

Shortcut for:

```python
bot.send_sticker(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments

- **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

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

Shortcut for:

```python
bot.send_message(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments

- **quote** (bool, optional) – If set to True, the message is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

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

Shortcut for:

```python
bot.send_venue(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments

- **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

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

Shortcut for:

```python
bot.send_video(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments

- **quote** (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`
reply_video_note(*args, **kwargs)

Shortcut for:

```
bot.send_video_note(update.message.chat_id, *args, **kwargs)
```

**Keyword Arguments**
- quote (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

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

**Return type** telegram.Message

reply_voice(*args, **kwargs)

Shortcut for:

```
bot.send_voice(update.message.chat_id, *args, **kwargs)
```

**Keyword Arguments**
- quote (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

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

**Return type** telegram.Message

text_html

Creates an HTML-formatted string from the markup entities found in the message.

Use this if you want to retrieve the message text with the entities formatted as HTML in the same way the original message was formatted.

**Returns** Message text with entities formatted as HTML.

**Return type** str

text_html_urled

Creates an HTML-formatted string from the markup entities found in the message.

Use this if you want to retrieve the message text with the entities formatted as HTML. This also formats telegram.MessageEntity.URL as a hyperlink.

**Returns** Message text with entities formatted as HTML.

**Return type** str

text_markdown

Creates an Markdown-formatted string from the markup entities found in the message.

Use this if you want to retrieve the message text with the entities formatted as Markdown in the same way the original message was formatted.

**Returns** Message text with entities formatted as Markdown.

**Return type** str

text_markdown_urled

Creates an Markdown-formatted string from the markup entities found in the message.

Use this if you want to retrieve the message text with the entities formatted as Markdown. This also formats telegram.MessageEntity.URL as a hyperlink.

**Returns** Message text with entities formatted as Markdown.

**Return type** str
1.29 telegram.MessageEntity

class telegram.MessageEntity (type, offset, length, url=None, user=None, **kwargs)
Bases: telegram.base.TelegramObject

This object represents one special entity in a text message. For example, hashtags, usernames, URLs, etc.

type
str – Type of the entity.

offset
int – Offset in UTF-16 code units to the start of the entity.

length
int – Length of the entity in UTF-16 code units.

url
str – Optional. Url that will be opened after user taps on the text.

user
telegram.User – Optional. The mentioned user.

Parameters

• type (str) – Type of the entity. Can be mention (@username), hashtag, bot_command, url, email, bold (bold text), italic (italic text), code (monowidth string), pre (monowidth block), text_link (for clickable text URLs), text_mention (for users without usernames).

• offset (int) – Offset in UTF-16 code units to the start of the entity.

• length (int) – Length of the entity in UTF-16 code units.

• url (str, optional) – For “text_link” only, url that will be opened after user taps on the text.

• user (telegram.User, optional) – For “text_mention” only, the mentioned user.

ALL_TYPES = ['mention', 'hashtag', 'cashtag', 'phone_number', 'bot_command', 'url',
List[str] – List of all the types.

BOLD = 'bold'
str – ‘bold’

BOT_COMMAND = 'bot_command'
str – ‘bot_command’

CASHTAG = 'cashtag'
str – ‘cashtag’

CODE = 'code'
str – ‘code’

EMAIL = 'email'
str – ‘email’

HASHTAG = 'hashtag'
str – ‘hashtag’

ITALIC = 'italic'
str – ‘italic’

MENTION = 'mention'
str – ‘mention’

PHONE_NUMBER = 'phone_number'
str – ‘phone_number’
PRE = 'pre'
    str - 'pre'

TEXT_LINK = 'text_link'
    str - 'text_link'

TEXT_MENTION = 'text_mention'
    str - 'text_mention'

URL = 'url'
    str - 'url'

1.30 telegram.ParseMode

class telegram.ParseMode
    Bases: object

    This object represents a Telegram Message Parse Modes.

    HTML = 'HTML'
        str - 'HTML'

    MARKDOWN = 'Markdown'
        str - 'Markdown'

1.31 telegram.PhotoSize

class telegram.PhotoSize
    Bases: telegram.base.TelegramObject

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

    file_id
        str - Unique identifier for this file.

    width
        int - Photo width.

    height
        int - Photo height.

    file_size
        int - Optional. File size.

    bot
        telegram.Bot - Optional. The Bot to use for instance methods.

    Parameters

        • file_id (str) – Unique identifier for this file.
        • width (int) – Photo width.
        • height (int) – Photo height.
        • file_size (int, optional) – File size.
        • bot (telegram.Bot, optional) – The Bot to use for instance methods.
        • **kwargs (dict) – Arbitrary keyword arguments.

    get_file (timeout=None, **kwargs)
        Convenience wrapper over telegram.Bot.get_file
Parameters

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

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

Returns *telegram.File*

Raises *telegram.TelegramError*

### 1.32 telegram.ReplyKeyboardRemove

**class** `telegram.ReplyKeyboardRemove` *(selective=False, **kwargs)*

**Bases:** `telegram.replymarkup.ReplyMarkup`

Upon receiving a message with this object, Telegram clients will remove the current custom keyboard and display the default letter-keyboard. By default, custom keyboards are displayed until a new keyboard is sent by a bot. An exception is made for one-time keyboards that are hidden immediately after the user presses a button (see `telegram.ReplyKeyboardMarkup`).

- **remove_keyboard** *(bool)* – Requests clients to remove the custom keyboard.

- **selective** *(bool)* – Optional. Use this parameter if you want to remove the keyboard for specific users only.

**Example**

A user votes in a poll, bot returns confirmation message in reply to the vote and removes the keyboard for that user, while still showing the keyboard with poll options to users who haven’t voted yet.

**Parameters**

- **selective** *(bool, optional)* – Use this parameter if you want to remove the keyboard for specific users only. Targets:

  1. users that are @mentioned in the text of the Message object
  2. if the bot’s message is a reply (has reply_to_message_id), sender of the original message.

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

### 1.33 telegram.ReplyKeyboardMarkup

**class** `telegram.ReplyKeyboardMarkup` *(keyboard, resize_keyboard=False, one_time_keyboard=False, selective=False, **kwargs)*

**Bases:** `telegram.replymarkup.ReplyMarkup`

This object represents a custom keyboard with reply options.


- **resize_keyboard** *(bool)* – Optional. Requests clients to resize the keyboard.

- **one_time_keyboard** *(bool)* – Optional. Requests clients to hide the keyboard as soon as it’s been used.
selective
    bool – Optional. Show the keyboard to specific users only.

Example

A user requests to change the bot’s language, bot replies to the request with a keyboard to select the new language. Other users in the group don’t see the keyboard.

Parameters

- **keyboard** (List[List[telegram.KeyboardButton]]) – Array of button rows, each represented by an Array of telegram.KeyboardButton objects.
- **resize_keyboard** (bool, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to false, in which case the custom keyboard is always of the same height as the app’s standard keyboard. Defaults to False.
- **one_time_keyboard** (bool, optional) – Requests clients to hide the keyboard as soon as it’s been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to False.
- **selective** (bool, optional) – Use this parameter if you want to show the keyboard to specific users only. Targets:
  1. users that are @mentioned in the text of the Message object
  2. if the bot’s message is a reply (has reply_to_message_id), sender of the original message.

  Defaults to False.

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

classmethod from_button

    button, resize_keyboard=False, one_time_keyboard=False, selective=False, **kwargs)

Shortcut for:

    ReplyKeyboardMarkup([[button]], **kwargs)

Return an ReplyKeyboardMarkup from a single KeyboardButton

Parameters

- **button** (telegram.KeyboardButton|str) – The button to use in the markup
- **resize_keyboard** (bool, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to false, in which case the custom keyboard is always of the same height as the app’s standard keyboard. Defaults to False.
- **one_time_keyboard** (bool, optional) – Requests clients to hide the keyboard as soon as it’s been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to False.
- **selective** (bool, optional) – Use this parameter if you want to show the keyboard to specific users only. Targets:
  1. users that are @mentioned in the text of the Message object
  2. if the bot’s message is a reply (has reply_to_message_id), sender of the original message.

  Defaults to False.
··· kwargs (dict) – Arbitrary keyword arguments.

classmethod from_column (button_column, resize_keyboard=False, one_time_keyboard=False, selective=False, **kwargs)

Shortcut for:

ReplyKeyboardMarkup([[button] for button in button_column], **kwargs)

Return an ReplyKeyboardMarkup from a single column of KeyboardButtons

Parameters

• button_column (List[telegram.KeyboardButton | str]) – The button to use in the markup

• resize_keyboard (bool, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to false, in which case the custom keyboard is always of the same height as the app’s standard keyboard. Defaults to False

• one_time_keyboard (bool, optional) – Requests clients to hide the keyboard as soon as it’s been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to False.

• selective (bool, optional) – Use this parameter if you want to show the keyboard to specific users only. Targets:
  1. users that are @mentioned in the text of the Message object
  2. if the bot’s message is a reply (has reply_to_message_id), sender of the original message.

Defaults to False.

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

classmethod from_row (button_row, resize_keyboard=False, one_time_keyboard=False, selective=False, **kwargs)

Shortcut for:

ReplyKeyboardMarkup([button_row], **kwargs)

Return an ReplyKeyboardMarkup from a single row of KeyboardButtons

Parameters

• button_row (List[telegram.KeyboardButton | str]) – The button to use in the markup

• resize_keyboard (bool, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to false, in which case the custom keyboard is always of the same height as the app’s standard keyboard. Defaults to False

• one_time_keyboard (bool, optional) – Requests clients to hide the keyboard as soon as it’s been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to False.

• selective (bool, optional) – Use this parameter if you want to show the keyboard to specific users only. Targets:
  1. users that are @mentioned in the text of the Message object
  2. if the bot’s message is a reply (has reply_to_message_id), sender of the original message.

Defaults to False.
**kwargs (dict) – Arbitrary keyword arguments.

### 1.34 `telegram.ReplyMarkup`

```python
class telegram.ReplyMarkup
    Bases: telegram.base.TelegramObject

    Base class for Telegram ReplyMarkup Objects.
```

### 1.35 `telegram.TelegramObject`

```python
class telegram.TelegramObject
    Bases: object

    Base class for most telegram objects.
    
    to_json()
        Returns str
```

### 1.36 `telegram.Update`

```python
class telegram.Update(update_id=None, message=None, edited_message=None, channel_post=None, edited_channel_post=None, inline_query=None, chosen_inline_result=None, callback_query=None, shipping_query=None, pre_checkout_query=None, **kwargs)
    Bases: telegram.base.TelegramObject

    This object represents an incoming update.

    **Note:** At most one of the optional parameters can be present in any given update.
```

- `update_id` – int – The update’s unique identifier.
- `chosen_inline_result` – `telegram.ChosenInlineResult` – Optional. The result of an inline query that was chosen by a user.
shipping_query

pre_checkout_query

Parameters

- **update_id** (int) – The update’s unique identifier. Update identifiers start from a certain positive number and increase sequentially. This ID becomes especially handy if you’re using Webhooks, since it allows you to ignore repeated updates or to restore the correct update sequence, should they get out of order.

- **message** (telegram.Message, optional) – New incoming message of any kind - text, photo, sticker, etc.

- **edited_message** (telegram.Message, optional) – New version of a message that is known to the bot and was edited.

- **channel_post** (telegram.Message, optional) – New incoming channel post of any kind - text, photo, sticker, etc.

- **edited_channel_post** (telegram.Message, optional) – New version of a channel post that is known to the bot and was edited.

- **inline_query** (telegram.InlineQuery, optional) – New incoming inline query.

- **chosen_inline_result** (telegram.ChosenInlineResult, optional) – The result of an inline query that was chosen by a user and sent to their chat partner.

- **callback_query** (telegram.CallbackQuery, optional) – New incoming callback query.

- **shipping_query** (telegram.ShippingQuery, optional) – New incoming shipping query. Only for invoices with flexible price.

- **pre_checkout_query** (telegram.PreCheckoutQuery, optional) – New incoming pre-checkout query. Contains full information about checkout

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

### classmethod de_json(data, bot)

effective_chat
telegram.Chat – The chat that this update was sent in, no matter what kind of update this is. Will be None for inline_query, chosen_inline_result, callback_query from inline messages, shipping_query and pre_checkout_query.

effective_message
telegram.Message – The message included in this update, no matter what kind of update this is. Will be None for inline_query, chosen_inline_result, callback_query from inline messages, shipping_query and pre_checkout_query.

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

### 1.37 telegram.User

class telegram.User(id, first_name, is_bot, last_name=None, username=None, language_code=None, bot=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents a Telegram user or bot.
id
    int – Unique identifier for this user or bot.

is_bot
    bool – True, if this user is a bot

first_name
    str – User’s or bot’s first name.

last_name
    str – Optional. User’s or bot’s last name.

username
    str – Optional. User’s or bot’s username.

language_code
    str – Optional. IETF language tag of the user’s language.

bot

Parameters

• id (int) – Unique identifier for this user or bot.
• is_bot (bool) – True, if this user is a bot
• first_name (str) – User’s or bot’s first name.
• last_name (str, optional) – User’s or bot’s last name.
• username (str, optional) – User’s or bot’s username.
• language_code (str, optional) – IETF language tag of the user’s language.
• bot (telegram.Bot, optional) – The Bot to use for instance methods.

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

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

get_profile_photos (*args, **kwargs)
Shortcut for:
    bot.get_user_profile_photos(update.message.from_user.id, *args, **kwargs)

link
    str – Convenience property. If username is available, returns a t.me link of the user.

mention_html (name=None)
Parameters
    name (str) – The name used as a link for the user. Defaults to full_name.
Returns
    The inline mention for the user as HTML.
Return type
    str

mention_markdown (name=None)
Parameters
    name (str) – The name used as a link for the user. Defaults to full_name.
Returns
    The inline mention for the user as markdown.
Return type
    str

name
    str – Convenience property. If available, returns the user’s username prefixed with “@”. If username is not available, returns full_name.
**send_animation**(*args, **kwargs*)

Shortcut for:

```python
bot.send_animation(User.id, *args, **kwargs)
```

Where User is the current instance.

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

**Return type** `telegram.Message`

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

Shortcut for:

```python
bot.send_audio(User.id, *args, **kwargs)
```

Where User is the current instance.

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

**Return type** `telegram.Message`

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

Shortcut for:

```python
bot.send_document(User.id, *args, **kwargs)
```

Where User is the current instance.

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

**Return type** `telegram.Message`

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

Shortcut for:

```python
bot.send_message(User.id, *args, **kwargs)
```

Where User is the current instance.

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

**Return type** `telegram.Message`

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

Shortcut for:

```python
bot.send_photo(User.id, *args, **kwargs)
```

Where User is the current instance.

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

**Return type** `telegram.Message`

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

Shortcut for:

```python
bot.send_sticker(User.id, *args, **kwargs)
```

Where User is the current instance.

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

**Return type** `telegram.Message`

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

Shortcut for:
bot.send_video(User.id, *args, **kwargs)

Where User is the current instance.

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

**Return type**  telegram.Message

send_video_note(*args, **kwargs)

Shortcut for:

bot.send_video_note(User.id, *args, **kwargs)

Where User is the current instance.

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

**Return type**  telegram.Message

send_voice(*args, **kwargs)

Shortcut for:

bot.send_voice(User.id, *args, **kwargs)

Where User is the current instance.

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

**Return type**  telegram.Message

**1.38 telegram.UserProfilePhotos**

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

**Bases:** telegram.base.TelegramObject

This object represents a user’s profile pictures.

**total_count**

int – Total number of profile pictures.

**photos**


**Parameters**

- **total_count** (int) – Total number of profile pictures the target user has.
- **photos** (List[List[telegram.PhotoSize]]) – Requested profile pictures (in up to 4 sizes each).

**1.39 telegram.Venue**

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

**Bases:** telegram.base.TelegramObject

This object represents a venue.

**location**

telegram.Location – Venue location.

**title**

str – Name of the venue.
address
str – Address of the venue.

foursquare_id
str – Optional. Foursquare identifier of the venue.

foursquare_type
str – Optional. Foursquare type of the venue. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”.)

Parameters
• location(telegram.Location) – Venue location.
• title(str) – Name of the venue.
• address(str) – Address of the venue.
• foursquare_id(str, optional) – Foursquare identifier of the venue.
• foursquare_type(str, optional) – Foursquare type of the venue. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”).
• **kwargs(dict) – Arbitrary keyword arguments.

1.40 telegram.Video

class telegram.Video(file_id, width, height, duration, thumb=None, mime_type=None, file_size=None, bot=None, **kwargs)
Bases: telegram.base.TelegramObject

This object represents a video file.

file_id
str – Unique identifier for this file.

width
int – Video width as defined by sender.

height
int – Video height as defined by sender.

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

thumb

mime_type
str – Optional. Mime type of a file as defined by sender.

file_size
int – Optional. File size.

bot

Parameters
• file_id(str) – Unique identifier for this file.
• width(int) – Video width as defined by sender.
• height(int) – Video height as defined by sender.
• duration(int) – Duration of the video in seconds as defined by sender.
• thumb(telegram.PhotoSize, optional) – Video thumbnail.
• `mime_type` (str, optional) – Mime type of a file as defined by sender.
• `file_size` (int, optional) – File size.
• `bot` (telegram.Bot, optional) – The Bot to use for instance methods.
• `**kwargs` (dict) – Arbitrary keyword arguments.

`get_file` (timeout=None, **kwargs)
Convenience wrapper over `telegram.Bot.get_file`

Parameters

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

Returns `telegram.File`

Raises `telegram.TelegramError`

1.41 `telegram.VideoNote`

class `telegram.VideoNote` (file_id, length, duration, thumb=None, file_size=None, bot=None, **kwargs)
Bases: `telegram.base.TelegramObject`

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

`file_id`  
str – Unique identifier for this file.

`length`  
int – Video width and height as defined by sender.

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

`thumb`  

`file_size`  
int – Optional. File size.

`bot`  

Parameters

• `file_id` (str) – Unique identifier for this file.
• `length` (int) – Video width and height as defined by sender.
• `duration` (int) – Duration of the video in seconds as defined by sender.
• `thumb` (telegram.PhotoSize, optional) – Video thumbnail.
• `file_size` (int, optional) – File size.
• `bot` (telegram.Bot, optional) – The Bot to use for instance methods.
• `**kwargs` (dict) – Arbitrary keyword arguments.

`get_file` (timeout=None, **kwargs)
Convenience wrapper over `telegram.Bot.get_file`
Parameters

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

Returns telegram.File

Raises telegram.TelegramError

1.42 telegram.Voice

class telegram.Voice(file_id, duration, mime_type=None, file_size=None, bot=None, **kwargs)
Bases: telegram.base.TelegramObject

This object represents a voice note.

**file_id**
str – Unique identifier for this file.

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

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

**file_size**
int – Optional. File size.

**bot**

Parameters

- **file_id**(str) – Unique identifier for this file.
- **duration**(int, optional) – Duration of the audio in seconds as defined by sender.
- **mime_type**(str, optional) – MIME type of the file as defined by sender.
- **file_size**(int, optional) – File size.
- **bot**(telegram.Bot, optional) – The Bot to use for instance methods.
- **kwargs**(dict) – Arbitrary keyword arguments.

get_file(timeout=None, **kwargs)
Convenience wrapper over telegram.Bot.get_file

Parameters

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

Returns telegram.File

Raises telegram.TelegramError
1.43 Telegram.WebhookInfo

```python
class telegram.WebhookInfo(url, has_custom_certificate, pending_update_count, last_error_date=None, last_error_message=None, max_connections=None, allowed_updates=None, **kwargs)

Bases: telegram.base.TelegramObject
```

This object represents a Telegram WebhookInfo.

Contains information about the current status of a webhook.

- **url** (str) – Webhook URL.
- **has_custom_certificate** (bool) – If a custom certificate was provided for webhook.
- **pending_update_count** (int) – Number of updates awaiting delivery.
- **last_error_date** (int, optional) – Unix time for the most recent error that happened.
- **last_error_message** (str, optional) – Error message in human-readable format.
- **max_connections** (int, optional) – Maximum allowed number of simultaneous HTTPS connections.
- **allowed_updates** (List[str], optional) – A list of update types the bot is subscribed to.

**Parameters**

- **url** (str) – Webhook URL, may be empty if webhook is not set up.
- **has_custom_certificate** (bool) – True, if a custom certificate was provided for webhook certificate checks.
- **pending_update_count** (int) – Number of updates awaiting delivery.
- **last_error_date** (int, optional) – Unix time for the most recent error that happened when trying to deliver an update via webhook.
- **last_error_message** (str, optional) – Error message in human-readable format for the most recent error that happened when trying to deliver an update via webhook.
- **max_connections** (int, optional) – Maximum allowed number of simultaneous HTTPS connections to the webhook for update delivery.
- **allowed_updates** (List[str], optional) – A list of update types the bot is subscribed to. Defaults to all update types.

1.44 Stickers

1.44.1 Telegram.Sticker

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

Bases: telegram.base.TelegramObject
```

This object represents a sticker.

- **file_id** (str) – Unique identifier for this file.
width
    int – Sticker width.

height
    int – Sticker height.

thumb
    telegram.PhotoSize – Optional. Sticker thumbnail in the .webp or .jpg format.

emoji
    str – Optional. Emoji associated with the sticker.

set_name
    str – Optional. Name of the sticker set to which the sticker belongs.

mask_position
    telegram.MaskPosition – Optional. For mask stickers, the position where the mask should be placed.

file_size
    int – Optional. File size.

bot

Parameters

- file_id (str) – Unique identifier for this file.
- width (int) – Sticker width.
- height (int) – Sticker height.
- thumb (telegram.PhotoSize, optional) – Sticker thumbnail in the .webp or .jpg format.
- emoji (str, optional) – Emoji associated with the sticker
- set_name (str, optional) – Name of the sticker set to which the sticker belongs.
- mask_position (telegram.MaskPosition, optional) – For mask stickers, the position where the mask should be placed.
- file_size (int, optional) – File size.
- (obj (**kwargs) – dict): Arbitrary keyword arguments.
- bot (telegram.Bot, optional) – The Bot to use for instance methods.

get_file (timeout=None, **kwargs)
    Convenience wrapper over telegram.Bot.get_file

Parameters

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

Returns telegram.File

Raises telegram.TelegramError

1.44.2 telegram.StickerSet

class telegram.StickerSet (name, title, contains_masks, stickers, bot=None, **kwargs)
    Bases: telegram.base.TelegramObject
This object represents a sticker set.

```python
definition:
    name (str) – Sticker set name.
    title (str) – Sticker set title.
    contains_masks (bool) – True, if the sticker set contains masks.
    stickers (List[telegram.Sticker]) – List of all set stickers.
```

### Parameters

- **name (str)** – Sticker set name.
- **title (str)** – Sticker set title.
- **contains_masks (bool)** – True, if the sticker set contains masks.
- **stickers (List[telegram.Sticker])** – List of all set stickers.

#### 1.44.3 telegram.MaskPosition

```python
class definition:
    mask_position (point, x_shift, y_shift, scale, **kwargs)
    Bases: telegram.base.TelegramObject
```

This object describes the position on faces where a mask should be placed by default.

- **point (str)** – The part of the face relative to which the mask should be placed.
- **x_shift (float)** – Shift by X-axis measured in widths of the mask scaled to the face size, from left to right.
- **y_shift (float)** – Shift by Y-axis measured in heights of the mask scaled to the face size, from top to bottom.
- **scale (float)** – Mask scaling coefficient. For example, 2.0 means double size.

#### Notes

- **type** should be one of the following: *forehead, eyes, mouth or chin*. You can use the class constants for those.

#### Parameters

- **point (str)** – The part of the face relative to which the mask should be placed.
- **x_shift (float)** – Shift by X-axis measured in widths of the mask scaled to the face size, from left to right. For example, choosing -1.0 will place mask just to the left of the default mask position.
- **y_shift (float)** – Shift by Y-axis measured in heights of the mask scaled to the face size, from top to bottom. For example, 1.0 will place the mask just below the default mask position.
- **scale (float)** – Mask scaling coefficient. For example, 2.0 means double size.

```python
CHIN = 'chin'
definition:
    str = 'chin'
```

EYES = 'eyes'
    str = 'eyes'

FOREHEAD = 'forehead'
    str = 'forehead'

MOUTH = 'mouth'
    str = 'mouth'

### 1.45 Inline Mode

#### 1.45.1 telegram.InlineQuery

class telegram.InlineQuery(id, from_user, query, offset, location=None, bot=None, **kwargs)

    Bases: telegram.base.TelegramObject

This object represents an incoming inline query. When the user sends an empty query, your bot could return some default or trending results.

**Note:**

- In Python `from` is a reserved word, use `from_user` instead.

**Parameters**

- `id` (str) – Unique identifier for this query.
- `from_user` (telegram.User) – Sender.
- `location` (telegram.Location, optional) – Optional. Sender location, only for bots that request user location.
- `query` (str) – Text of the query (up to 512 characters).
- `offset` (str) – Offset of the results to be returned, can be controlled by the bot.

**Parameters**

- `id` (str) – Unique identifier for this query.
- `from_user` (telegram.User) – Sender.
- `location` (telegram.Location, optional) – Sender location, only for bots that request user location.
- `query` (str) – Text of the query (up to 512 characters).
- `offset` (str) – Offset of the results to be returned, can be controlled by the bot.
- `bot` (telegram.Bot, optional) – The Bot to use for instance methods.
- `**kwargs` (dict) – Arbitrary keyword arguments.

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

Shortcut for:

```python
bot.answer_inline_query(update.inline_query.id, *args, **kwargs)
```

**Parameters**
• **results** (List[telegram.InlineQueryResult]) – A list of results for the inline query.

• **cache_time** (int, optional) – The maximum amount of time in seconds that the result of the inline query may be cached on the server. Defaults to 300.

• **is_personal** (bool, optional) – Pass True, if results may be cached on the server side only for the user that sent the query. By default, results may be returned to any user who sends the same query.

• **next_offset** (str, optional) – Pass the offset that a client should send in the next query with the same text to receive more results. Pass an empty string if there are no more results or if you don’t support pagination. Offset length can’t exceed 64 bytes.

• **switch_pm_text** (str, optional) – If passed, clients will display a button with specified text that switches the user to a private chat with the bot and sends the bot a start message with the parameter switch_pm_parameter.

• **switch_pm_parameter** (str, optional) – Deep-linking parameter for the /start message sent to the bot when user presses the switch button. 1-64 characters, only A-Z, a-z, 0-9, _ and - are allowed.

### 1.45.2 telegram.InlineQueryResult

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

Bases: telegram.base.TelegramObject

Baseclass for the InlineQueryResult* classes.

- **type** (`str`) – Type of the result.
- **id** (`str`) – Unique identifier for this result, 1-64 Bytes.

**Parameters**

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

### 1.45.3 telegram.InlineQueryResultArticle

class telegram.InlineQueryResultArticle(id, title, input_message_content, reply_markup=None, url=None, hide_url=None, description=None, thumb_url=None, thumb_width=None, thumb_height=None, **kwargs)

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

This object represents a Telegram InlineQueryResultArticle.

- **type** (`str`) – ‘article’.
- **id** (`str`) – Unique identifier for this result, 1-64 Bytes.
- **title** (`str`) – Title of the result.
- **input_message_content** (`telegram.InputMessageContent`) – Content of the message to be sent.
reply_markup
telegram.ReplyMarkup – Optional. Inline keyboard attached to the message.

url
str – Optional. URL of the result.

hide_url
bool – Optional. Pass True, if you don’t want the URL to be shown in the message.

description
str – Optional. Short description of the result.

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

thumb_width
int – Optional. Thumbnail width.

thumb_height
int – Optional. Thumbnail height.

Parameters

• id (str) – Unique identifier for this result, 1-64 Bytes.
• title (str) – Title of the result.
• input_message_content (telegram.InputMessageContent) – Content of the message to be sent.
• reply_markup (telegram.ReplyMarkup, optional) – Inline keyboard attached to the message
• url (str, optional) – URL of the result.
• hide_url (bool, optional) – Pass True, if you don’t want the URL to be shown in the message.
• description (str, optional) – Short description of the result.
• thumb_url (str, optional) – Url of the thumbnail for the result.
• thumb_width (int, optional) – Thumbnail width.
• thumb_height (int, optional) – Thumbnail height.
• **kwargs (dict) – Arbitrary keyword arguments.

1.45.4 telegram.InlineQueryResultAudio
class telegram.InlineQueryResultAudio
(id, audio_url, title, performer=None, audio_duration=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)
Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to an mp3 audio file. By default, this audio file will be sent by the user. Alternatively, you can use input_message_content to send a message with the specified content instead of the audio.

type
str – ‘audio’.

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

audio_url
str – A valid URL for the audio file.
title
str – Title.

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

audio_duration
str – Optional. Performer.

caption
str – Optional. Audio duration in seconds.

parse_mode
str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

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

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

Parameters

• id (str) – Unique identifier for this result, 1-64 bytes.
• audio_url (str) – A valid URL for the audio file.
• title (str) – Title.
• performer (str, optional) – Caption, 0-200 characters.
• audio_duration (str, optional) – Performer.
• caption (str, optional) – Audio duration in seconds.
• parse_mode (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
• reply_markup (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.
• input_message_content (telegram.InputMessageContent, optional) – Content of the message to be sent instead of the audio.
• **kwargs (dict) – Arbitrary keyword arguments.

1.45.5 telegram.InlineQueryResultCachedAudio

class telegram.InlineQueryResultCachedAudio(id, audio_file_id, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to an mp3 audio file stored on the Telegram servers. By default, this audio file will be sent by the user. Alternatively, you can use input_message_content to send a message with the specified content instead of the audio.

type
str – ‘audio’.

id
str – Unique identifier for this result, 1-64 bytes.
audio_file_id
str – A valid file identifier for the audio file.

caption
str – Optional. Caption, 0-1024 characters

parse_mode
str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

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

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

Parameters

* id (str) – Unique identifier for this result, 1-64 bytes.
* audio_file_id (str) – A valid file identifier for the audio file.
* caption (str, optional) – Caption, 0-1024 characters
* parse_mode (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
* reply_markup (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.
* input_message_content (telegram.InputMessageContent, optional) – Content of the message to be sent instead of the audio.
* **kwargs (dict) – Arbitrary keyword arguments.

1.45.6 telegram.InlineQueryResultCachedDocument
class telegram.InlineQueryResultCachedDocument (id, title, document_file_id, description=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to a file stored on the Telegram servers. By default, this file will be sent by the user with an optional caption. Alternatively, you can use input_message_content to send a message with the specified content instead of the file.

type
str – ‘document’.

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

title
str – Title for the result.

document_file_id
str – A valid file identifier for the file.

description
str – Optional. Short description of the result.
caption
str – Optional. Caption, 0-1024 characters

parse_mode
str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

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

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

Parameters
• id(str) – Unique identifier for this result, 1-64 bytes.
• title(str) – Title for the result.
• document_file_id(str) – A valid file identifier for the file.
• description(str, optional) – Short description of the result.
• caption(str, optional) – Caption, 0-1024 characters
• parse_mode(str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
• reply_markup(telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.
• input_message_content(telegram.InputMessageContent, optional) – Content of the message to be sent instead of the file.
• **kwargs(dict) – Arbitrary keyword arguments.

1.45.7 telegram.InlineQueryResultCachedGif
class telegram.InlineQueryResultCachedGif(id, gif_file_id, title=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to an animated GIF file stored on the Telegram servers. By default, this animated GIF file will be sent by the user with an optional caption. Alternatively, you can use input_message_content to send a message with specified content instead of the animation.

type
str – ‘gif’.

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

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

title
str – Optional. Title for the result.

caption
str – Optional. Caption, 0-1024 characters
parse_mode
str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

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

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

Parameters
- id (str) – Unique identifier for this result, 1-64 bytes.
- gif_file_id (str) – A valid file identifier for the GIF file.
- title (str, optional) – Title for the result.
- caption (str, optional) – Caption, 0-1024 characters
- parse_mode (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
- reply_markup (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.
- input_message_content (telegram.InputMessageContent, optional) – Content of the message to be sent instead of the gif.
- **kwargs (dict) – Arbitrary keyword arguments.

1.45.8 telegram.InlineQueryResultCachedMpeg4Gif

class telegram.InlineQueryResultCachedMpeg4Gif (id, mpeg4_file_id, title=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to a video animation (H.264/MPEG-4 AVC video without sound) stored on the Telegram servers. By default, this animated MPEG-4 file will be sent by the user with an optional caption. Alternatively, you can use input_message_content to send a message with the specified content instead of the animation.

type
str – 'mpeg4_gif'.

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

mpeg4_file_id
str – A valid file identifier for the MP4 file.

title
str – Optional. Title for the result.

caption
str – Optional. Caption, 0-1024 characters

parse_mode
str – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
reply_markup

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

input_message_content

telegram.InputMessageContent – Optional. Content of the message to be sent instead of the MPEG-4 file.

Parameters

• id (str) – Unique identifier for this result, 1-64 bytes.
• mpeg4_file_id (str) – A valid file identifier for the MP4 file.
• title (str, optional) – Title for the result.
• caption (str, optional) – Caption, 0-1024 characters
• parse_mode (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
• reply_markup (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.
• input_message_content (telegram.InputMessageContent, optional) – Content of the message to be sent instead of the MPEG-4 file.
• **kwargs (dict) – Arbitrary keyword arguments.

1.45.9 telegram.InlineQueryResultCachedPhoto

class telegram.InlineQueryResultCachedPhoto (id, photo_file_id, title=None, description=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to a photo stored on the Telegram servers. By default, this photo will be sent by the user with an optional caption. Alternatively, you can use input_message_content to send a message with the specified content instead of the photo.

    type
    str – ‘photo’.

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

photo_file_id
    str – A valid file identifier of the photo.

title
    str – Optional. Title for the result.

description
    str – Optional. Short description of the result.

caption
    str – Optional. Caption, 0-1024 characters

parse_mode
    str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
**reply_markup**
*telegram.InlineKeyboardMarkup* – Optional. Inline keyboard attached to the message.

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

**Parameters**

- **id** *(str)* – Unique identifier for this result, 1-64 bytes.
- **photo_file_id** *(str)* – A valid file identifier of the photo.
- **title** *(str, optional)* – Title for the result.
- **description** *(str, optional)* – Short description of the result.
- **caption** *(str, optional)* – Caption, 0-1024 characters
- **parse_mode** *(str, optional)* – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.
- **reply_markup** *(telegram.InlineKeyboardMarkup, optional)* – Inline keyboard attached to the message.
- **input_message_content** *(telegram.InputMessageContent, optional)* – Content of the message to be sent instead of the photo.
- ****kwargs** *(dict)* – Arbitrary keyword arguments.

### 1.45.10 telegram.InlineQueryResultCachedSticker

**class telegram.InlineQueryResultCachedSticker**

```python
id, sticker_file_id, reply_markup=None, input_message_content=None, **kwargs)
```

**Bases:** :class:`telegram.inline.inlinequeryresult.InlineQueryResult`

Represents a link to a sticker stored on the Telegram servers. By default, this sticker will be sent by the user. Alternatively, you can use *input_message_content* to send a message with the specified content instead of the sticker.

**type**
*str* – 'sticker'.

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

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

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

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

**Parameters**

- **id** *(str)* –
- **sticker_file_id** *(str)* –
- **reply_markup** *(telegram.InlineKeyboardMarkup, optional)* – Inline keyboard attached to the message.
* `input_message_content` *(telegram.InputMessageContent, optional)* – Content of the message to be sent instead of the sticker.

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

### 1.45.11 telegram.InlineQueryResultCachedVideo

#### class telegram.InlineQueryResultCachedVideo *(id, video_file_id, title, description=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)*

Bases: `telegram.inline.inlinequeryresult.InlineQueryResult`

Represents a link to a video file stored on the Telegram servers. By default, this video file will be sent by the user with an optional caption. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the video.

- **`type`** *(str)* – ‘video’.

- **`id`** *(str)* – Unique identifier for this result, 1-64 bytes.

- **`video_file_id`** *(str)* – A valid file identifier for the video file.

- **`title`** *(str)* – Title for the result.

- **`description`** *(str, optional)* – Short description of the result.

- **`caption`** *(str, optional)* – Caption, 0-1024 characters.

- **`parse_mode`** *(str, optional)* – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

- **`reply_markup`** *(telegram.InlineKeyboardMarkup)* – Optional. Inline keyboard attached to the message.

- **`input_message_content`** *(telegram.InputMessageContent)* – Optional. Content of the message to be sent instead of the video.

#### Parameters

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

- **video_file_id**(str) – A valid file identifier for the video file.

- **title**(str) – Title for the result.

- **description**(str, optional) – Short description of the result.

- **caption**(str, optional) – Caption, 0-1024 characters.

- **parse_mode**(str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

- **reply_markup**(telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.
• **input_message_content** (**telegram.InputMessageContent**, optional) – Content of the message to be sent instead of the video.
• **kwargs** (**dict**) – Arbitrary keyword arguments.

### 1.45.12 telegram.InlineQueryResultCachedVoice

**class** `telegram.InlineQueryResultCachedVoice(id, voice_file_id, title, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)`

**Bases:** `telegram.inline.inlinequeryresult.InlineQueryResult`

Represents a link to a voice message stored on the Telegram servers. By default, this voice message will be sent by the user. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the voice message.

- **type**
  ```
  str – ‘voice’.
  ```

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

- **voice_file_id**
  ```
  str – A valid file identifier for the voice message.
  ```

- **title**
  ```
  str – Voice message title.
  ```

- **caption**
  ```
  str – Optional. Caption, 0-1024 characters.
  ```

- **parse_mode**
  ```
  str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
  ```

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

- **input_message_content**
  ```
  `telegram.InputMessageContent` – Optional. Content of the message to be sent instead of the voice.
  ```

**Parameters**

- **id** (**str**) – Unique identifier for this result, 1-64 bytes.
- **voice_file_id** (**str**) – A valid file identifier for the voice message.
- **title** (**str**) – Voice message title.
- **caption** (**str**, optional) – Caption, 0-1024 characters.
- **parse_mode** (**str**, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (**`telegram.InlineKeyboardMarkup`**, optional) – Inline keyboard attached to the message.
- **input_message_content** (**`telegram.InputMessageContent`**, optional) – Content of the message to be sent instead of the voice.
- **kwargs** (**dict**) – Arbitrary keyword arguments.
1.45.13  telegram.InlineQueryResultContact

class telegram.InlineQueryResultContact(id,  phone_number,  first_name,  
last_name=None,  reply_markup=None,  
input_message_content=None,  thumb_url=None,  thumb_width=None,  
thumb_height=None,  vcard=None,  **kwargs)

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a contact with a phone number. By default, this contact will be sent by the user. Alternatively, 
you can use input_message_content to send a message with the specified content instead of the 
contact.

type
  str – ‘contact’.

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

phone_number
  str – Contact’s phone number.

first_name
  str – Contact’s first name.

last_name
  str – Optional. Contact’s last name.

vcard
  str – Optional. Additional data about the contact in the form of a vCard, 0-2048 bytes.

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

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

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

thumb_width
  int – Optional. Thumbnail width.

thumb_height
  int – Optional. Thumbnail height.

Parameters

• id (str) – Unique identifier for this result, 1-64 bytes.
• phone_number (str) – Contact’s phone number.
• first_name (str) – Contact’s first name.
• last_name (str, optional) – Contact’s last name.
• vcard (str, optional) – Additional data about the contact in the form of a vCard, 
0-2048 bytes.
• reply_markup (telegram.InlineKeyboardMarkup, optional) – Inline key-
board attached to the message.
• input_message_content (telegram.InputMessageContent, optional) – 
Content of the message to be sent instead of the contact.
• thumb_url (str, optional) – Url of the thumbnail for the result.
• thumb_width (int, optional) – Thumbnail width.
Python Telegram Bot Documentation, Release 12.0.0b1

• **thumb_height** (int, optional) – Thumbnai height.
• **kwargs** (dict) – Arbitrary keyword arguments.

### 1.45.14 **telegram.InlineQueryResultDocument**

class **telegram.InlineQueryResultDocument**

```python
(id, document_url, title, mime_type,
caption=None, description=None,
reply_markup=None, input_message_content=None,
thumb_url=None, thumb_width=None,
thumb_height=None, parse_mode=None,**kwargs)
```

Bases: **telegram.inline.inlinequeryresult.InlineQueryResult**

Represents a link to a file. By default, this file will be sent by the user with an optional caption. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the file. Currently, only .PDF and .ZIP files can be sent using this method.

**type**

str – ‘document’.

**id**

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

**title**

str – Title for the result.

**caption**

str – Optional. Caption, 0-1024 characters

**parse_mode**

str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in **telegram.ParseMode** for the available modes.

**document_url**

str – A valid URL for the file.

**mime_type**

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

**description**

str – Optional. Short description of the result.

**reply_markup**

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

**input_message_content**

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

**thumb_url**

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

**thumb_width**

int – Optional. Thumbnail width.

**thumb_height**

int – Optional. Thumbnail height.

**Parameters**

• **id** (str) – Unique identifier for this result, 1-64 bytes.
• **title** (str) – Title for the result.
• `caption (str, optional)` – Caption, 0-1024 characters

• `parse_mode (str, optional)` – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

• `document_url (str)` – A valid URL for the file.

• `mime_type (str)` – Mime type of the content of the file, either “application/pdf” or “application/zip”.

• `description (str, optional)` – Short description of the result.

• `reply_markup (telegram.InlineKeyboardMarkup)` – Optional. Inline keyboard attached to the message.

• `input_message_content (telegram.InputMessageContent)` – Optional. Content of the message to be sent instead of the file.

• `thumb_url (str, optional)` – URL of the thumbnail (jpeg only) for the file.

• `thumb_width (int, optional)` – Thumbnail width.

• `thumb_height (int, optional)` – Thumbnail height.

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

### 1.45.15 `telegram.InlineQueryResultGame`

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

Bases: `telegram.inline.inlinequeryresult.InlineQueryResult`

Represents a Game.

**type**  
```
str – 'game'.
```

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

**game_short_name**  
```
str – Short name of the game.
```

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

**Parameters**

• `id (str)` – Unique identifier for this result, 1-64 bytes.

• `game_short_name (str)` – Short name of the game.

• `reply_markup (telegram.InlineKeyboardMarkup, optional)` – Inline keyboard attached to the message.

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

### 1.45.16 `telegram.InlineQueryResultGif`

class `telegram.InlineQueryResultGif`(
    id,  
    gif_url,  
    thumb_url,  
    gif_width=None,  
    gif_height=None,  
    title=None,  
    caption=None,  
    reply_markup=None,  
    input_message_content=None,  
    gif_duration=None,  
    parse_mode=None,  
    **kwargs
)

Bases: `telegram.inline.inlinequeryresult.InlineQueryResult`

Represents a GIF.
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 \texttt{input_message_content} to send a message with the specified content instead of the animation.

\begin{itemize}
  \item \textbf{type}
    \begin{itemize}
    \item \texttt{str} – ‘gif’.
    \end{itemize}
  \item \textbf{id}
    \begin{itemize}
    \item \texttt{str} – Unique identifier for this result, 1-64 bytes.
    \end{itemize}
  \item \textbf{gif_url}
    \begin{itemize}
    \item \texttt{str} – A valid URL for the GIF file. File size must not exceed 1MB.
    \end{itemize}
  \item \textbf{gif_width}
    \begin{itemize}
    \item \texttt{int} – Optional. Width of the GIF.
    \end{itemize}
  \item \textbf{gif_height}
    \begin{itemize}
    \item \texttt{int} – Optional. Height of the GIF.
    \end{itemize}
  \item \textbf{gif_duration}
    \begin{itemize}
    \item \texttt{int} – Optional. Duration of the GIF.
    \end{itemize}
  \item \textbf{thumb_url}
    \begin{itemize}
    \item \texttt{str} – URL of the static thumbnail for the result (jpeg or gif).
    \end{itemize}
  \item \textbf{title}
    \begin{itemize}
    \item \texttt{str} – Optional. Title for the result.
    \end{itemize}
  \item \textbf{caption}
    \begin{itemize}
    \item \texttt{str} – Optional. Caption, 0-1024 characters
    \end{itemize}
  \item \textbf{parse_mode}
    \begin{itemize}
    \item \texttt{str} – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in \texttt{telegram.ParseMode} for the available modes.
    \end{itemize}
  \item \textbf{reply_markup}
    \begin{itemize}
    \item \texttt{telegram.InlineKeyboardMarkup} – Optional. Inline keyboard attached to the message.
    \end{itemize}
  \item \textbf{input_message_content}
    \begin{itemize}
    \item \texttt{telegram.InputMessageContent} – Optional. Content of the message to be sent instead of the gif.
    \end{itemize}
\end{itemize}

\textbf{Parameters}

- \textbf{id} (\texttt{str}) – Unique identifier for this result, 1-64 bytes.
- \textbf{gif_url} (\texttt{str}) – A valid URL for the GIF file. File size must not exceed 1MB.
- \textbf{gif_width} (\texttt{int}, optional) – Width of the GIF.
- \textbf{gif_height} (\texttt{int}, optional) – Height of the GIF.
- \textbf{gif_duration} (\texttt{int}, optional) – Duration of the GIF.
- \textbf{thumb_url} (\texttt{str}) – URL of the static thumbnail for the result (jpeg or gif).
- \textbf{title} (\texttt{str}, optional) – Title for the result.
- \textbf{caption} (\texttt{str}, optional) – Caption, 0-1024 characters
- \textbf{parse_mode} (\texttt{str}, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in \texttt{telegram.ParseMode} for the available modes.
- \textbf{reply_markup} (\texttt{telegram.InlineKeyboardMarkup}, optional) – Inline keyboard attached to the message.
- \textbf{input_message_content} (\texttt{telegram.InputMessageContent}, optional) – Content of the message to be sent instead of the gif.
• **kwargs (dict) – Arbitrary keyword arguments.

### 1.45.17 telegram.InlineQueryResultLocation

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

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a location on a map. By default, the location will be sent by the user. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the location.

- **type**
  - str – ‘location’.

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

- **latitude**
  - float – Location latitude in degrees.

- **longitude**
  - float – Location longitude in degrees.

- **title**
  - str – Location title.

- **live_period**
  - int – Optional. Period in seconds for which the location can be updated, should be between 60 and 86400.

- **reply_markup**
  - telegram.InlineKeyboardMarkup – Optional. Inline keyboard attached to the message.

- **input_message_content**
  - telegram.InputMessageContent – Optional. Content of the message to be sent instead of the location.

- **thumb_url**
  - str – Optional. Url of the thumbnail for the result.

- **thumb_width**
  - int – Optional. Thumbnail width.

- **thumb_height**
  - int – Optional. Thumbnail height.

Parameters

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

- **latitude** (float) – Location latitude in degrees.

- **longitude** (float) – Location longitude in degrees.

- **title** (str) – Location title.

- **live_period** (int, optional) – Period in seconds for which the location can be updated, should be between 60 and 86400.

- **reply_markup** (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.

- **input_message_content** (telegram.InputMessageContent, optional) – Content of the message to be sent instead of the location.
• **thumb_url** (str, optional) – Url of the thumbnail for the result.
  • **thumb_width** (int, optional) – Thumbnail width.
  • **thumb_height** (int, optional) – Thumbnail height.
  • **kwargs** (dict) – Arbitrary keyword arguments.

### 1.45.18 `telegram.InlineQueryResultMpeg4Gif`

```python
class telegram.InlineQueryResultMpeg4Gif(id, mpeg4_url, thumb_url, mpeg4_width=None, mpeg4_height=None, title=None, caption=None, reply_markup=None, input_message_content=None, mpeg4_duration=None, parse_mode=None, **kwargs)
```

Bases: `telegram.inline.inlinequeryresult.InlineQueryResult`

Represents a link to a video animation (H.264/MPEG-4 AVC video without sound). By default, this animated MPEG-4 file will be sent by the user with optional caption. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the animation.

**type**

str – ‘mpeg4_gif’.

**id**

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

**mpeg4_url**

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

**mpeg4_width**

int – Optional. Video width.

**mpeg4_height**

int – Optional. Video height.

**mpeg4_duration**

int – Optional. Video duration.

**thumb_url**

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

**title**

str – Optional. Title for the result.

**caption**

str – Optional. Caption, 0-1024 characters

**parse_mode**

str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

**reply_markup**

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

**input_message_content**

`telegram.InputMessageContent` – Optional. Content of the message to be sent instead of the MPEG-4 file.

**Parameters**

• **id** (str) – Unique identifier for this result, 1-64 bytes.
• **mpeg4_url** *(str)* – A valid URL for the MP4 file. File size must not exceed 1MB.
• **mpeg4_width** *(int, optional)* – Video width.
• **mpeg4_height** *(int, optional)* – Video height.
• **mpeg4_duration** *(int, optional)* – Video duration.
• **thumb_url** *(str)* – URL of the static thumbnail (jpeg or gif) for the result.
• **title** *(str, optional)* – Title for the result.
• **caption** *(str, optional)* – Caption, 0-1024 characters
• **parse_mode** *(str, optional)* – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.
• **reply_markup** *(telegram.InlineKeyboardMarkup, optional)* – Inline keyboard attached to the message.
• **input_message_content** *(telegram.InputMessageContent, optional)* – Content of the message to be sent instead of the MPEG-4 file.
• ****kwargs** *(dict)* – Arbitrary keyword arguments.

### 1.45.19 telegram.InlineQueryResultPhoto

class telegram.InlineQueryResultPhoto *(id, photo_url, thumb_url, photo_width=None, photo_height=None, title=None, description=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)*

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to a photo. By default, this photo will be sent by the user with optional caption. Alternatively, you can use *input_message_content* to send a message with the specified content instead of the photo.

**type**
str – ‘photo’.

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

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

**thumb_url**
str – URL of the thumbnail for the photo.

**photo_width**
int – Optional. Width of the photo.

**photo_height**
int – Optional. Height of the photo.

**title**
str – Optional. Title for the result.

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

**caption**
str – Optional. Caption, 0-1024 characters
parse_mode
str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

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

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

Parameters

• id (str) – Unique identifier for this result, 1-64 bytes.
• photo_url (str) – A valid URL of the photo. Photo must be in jpeg format. Photo size must not exceed 5MB.
• thumb_url (str) – URL of the thumbnail for the photo.
• photo_width (int, optional) – Width of the photo.
• photo_height (int, optional) – Height of the photo.
• title (str, optional) – Title for the result.
• description (str, optional) – Short description of the result.
• caption (str, optional) – Caption, 0-1024 characters
• parse_mode (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
• reply_markup (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.
• input_message_content (telegram.InputMessageContent, optional) – Content of the message to be sent instead of the photo.
• **kwargs (dict) – Arbitrary keyword arguments.

1.45.20 telegram.InlineQueryResultVenue

class telegram.InlineQueryResultVenue (id, latitude, longitude, title,
address=None, foursquare_id=None, foursquare_type=None,
reply_markup=None, input_message_content=None, thumb_url=None,
thumb_width=None, thumb_height=None, **kwargs)
Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a venue. By default, the venue will be sent by the user. Alternatively, you can use input_message_content to send a message with the specified content instead of the venue.

type
str – ‘venue’.

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

latitude
float – Latitude of the venue location in degrees.

longitude
float – Longitude of the venue location in degrees.
title
str – Title of the venue.

address
str – Address of the venue.

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

foursquare_type
str – Optional. Foursquare type of the venue, if known. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”.)

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

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

thumb_url
str – Optional. Url of the thumbnail for the result.

thumb_width
int – Optional. Thumbnail width.

thumb_height
int – Optional. Thumbnail height.

Parameters

• id (str) – Unique identifier for this result, 1-64 Bytes.
• latitude (float) – Latitude of the venue location in degrees.
• longitude (float) – Longitude of the venue location in degrees.
• title (str) – Title of the venue.
• address (str) – Address of the venue.
• foursquare_id (str, optional) – Foursquare identifier of the venue if known.
• foursquare_type (str, optional) – Foursquare type of the venue, if known. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”.)
• reply_markup (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.
• input_message_content (telegram.InputMessageContent, optional) – Content of the message to be sent instead of the location.
• thumb_url (str, optional) – Url of the thumbnail for the result.
• thumb_width (int, optional) – Thumbnail width.
• thumb_height (int, optional) – Thumbnail height.
• **kwargs (dict) – Arbitrary keyword arguments.
1.45.21  `telegram.InlineQueryResultVideo`

class `telegram.InlineQueryResultVideo(id, video_url, mime_type, thumb_url, title, caption=None, video_width=None, video_height=None, video_duration=None, description=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)`

Bases: `telegram.inline.inlinequeryresult.InlineQueryResult`

Represents a link to a page containing an embedded video player or a video file. By default, this video file will be sent by the user with an optional caption. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the video.

**type**

str – ‘video’.

**id**

str – Unique identifier for this result, 1-64 bytes.

**video_url**

str – A valid URL for the embedded video player or video file.

**mime_type**

str – Mime type of the content of video url, “text/html” or “video/mp4”.

**thumb_url**

str – URL of the thumbnail (jpeg only) for the video.

**title**

str – Title for the result.

**caption**

str – Optional. Caption, 0-1024 characters

**parse_mode**

str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

**video_width**

int – Optional. Video width.

**video_height**

int – Optional. Video height.

**video_duration**

int – Optional. Video duration in seconds.

**description**

str – Optional. Short description of the result.

**reply_markup**

`telegram.InlineKeyboardMarkup` – Optional. Inline keyboard attached to the message.

**input_message_content**

`telegram.InputMessageContent` – Optional. Content of the message to be sent instead of the video.

**Parameters**

- **id**(str) – Unique identifier for this result, 1-64 bytes.
- **video_url**(str) – A valid URL for the embedded video player or video file.
- **mime_type**(str) – Mime type of the content of video url, “text/html” or “video/mp4”.

---

1.45.  Inline Mode
• `thumb_url (str)` – URL of the thumbnail (jpeg only) for the video.
• `title (str)` – Title for the result.
• `caption (str, optional)` – Caption, 0-1024 characters.
• `parse_mode (str, optional)` – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
• `video_width (int, optional)` – Video width.
• `video_height (int, optional)` – Video height.
• `video_duration (int, optional)` – Video duration in seconds.
• `description (str, optional)` – Short description of the result.
• `reply_markup (telegram.InlineKeyboardMarkup, optional)` – Inline keyboard attached to the message.
• `input_message_content (telegram.InputMessageContent, optional)` – Content of the message to be sent instead of the video.
• **kwargs (dict) – Arbitrary keyword arguments.

1.45.22 `telegram.InlineQueryResultVoice`

class `telegram.InlineQueryResultVoice`(
    `id`, `voice_url`, `title`, `voice_duration=None`, `caption=None`, `reply_markup=None`, `input_message_content=None`, `parse_mode=None`, **kwargs)

Bases: `telegram.inline.inlinequeryresult.InlineQueryResult`

Represents a link to a voice recording in an .ogg container encoded with OPUS. By default, this voice recording will be sent by the user. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the voice message.

**type**
    str – 'voice'.

**id**
    str – Unique identifier for this result, 1-64 bytes.

**voice_url**
    str – A valid URL for the voice recording.

**title**
    str – Voice message title.

**caption**
    str – Optional. Caption, 0-1024 characters.

**parse_mode**
    str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

**voice_duration**
    int – Optional. Recording duration in seconds.

**reply_markup**
    `telegram.InlineKeyboardMarkup` – Optional. Inline keyboard attached to the message.

**input_message_content**
    `telegram.InputMessageContent` – Optional. Content of the message to be sent instead of the voice.
Parameters

- **id** *(str)* – Unique identifier for this result, 1-64 bytes.
- **voice_url** *(str)* – A valid URL for the voice recording.
- **title** *(str)* – Voice message title.
- **caption** *(str, optional)* – Caption, 0-1024 characters.
- **parse_mode** *(str, optional)* – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **voice_duration** *(int, optional)* – Recording duration in seconds.
- **reply_markup** *(`telegram.InlineKeyboardMarkup`, optional)* – Inline keyboard attached to the message.
- **input_message_content** *(`telegram.InputMessageContent`, optional)* – Content of the message to be sent instead of the voice.
- ****kwargs** *(dict)* – Arbitrary keyword arguments.

### 1.45.23 `telegram.InputMessageContent`

class `telegram.InputMessageContent`

Bases: `telegram.base.TelegramObject`

Base class for Telegram InputMessageContent Objects.


### 1.45.24 `telegram.InputTextMessageContent`

class `telegram.InputTextMessageContent` *(message_text, parse_mode=None, disable_web_page_preview=None, **kwargs)*

Bases: `telegram.inline.inputmessagecontent.InputMessageContent`

Represents the content of a text message to be sent as the result of an inline query.

**message_text**

`str` – Text of the message to be sent, 1-4096 characters.

**parse_mode**

`str` – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot’s message.

**disable_web_page_preview**

`bool` – Optional. Disables link previews for links in the sent message.

Parameters

- **message_text** *(str)* – Text of the message to be sent, 1-4096 characters. Also found as `telegram.constants.MAX_MESSAGE_LENGTH`.
- **parse_mode** *(str, optional)* – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot’s message.
- **disable_web_page_preview** *(bool, optional)* – Disables link previews for links in the sent message.
- ****kwargs** *(dict)* – Arbitrary keyword arguments.
1.45.25  telegram.InputLocationMessageContent

class telegram.InputLocationMessageContent (latitude, longitude, live_period=None, **kwargs)
Bases: telegram.inline.inputmessagecontent.InputMessageContent

Represents the content of a location message to be sent as the result of an inline query.

latitude
  float – Latitude of the location in degrees.

longitude
  float – Longitude of the location in degrees.

Parameters
  • latitude (float) – Latitude of the location in degrees.
  • longitude (float) – Longitude of the location in degrees.
  • live_period (int, optional) – Period in seconds for which the location can be updated, should be between 60 and 86400.
  • **kwargs (dict) – Arbitrary keyword arguments.

1.45.26  telegram.InputVenueMessageContent

class telegram.InputVenueMessageContent (latitude, longitude, title, address, foursquare_id=None, foursquare_type=None, **kwargs)
Bases: telegram.inline.inputmessagecontent.InputMessageContent

Represents the content of a venue message to be sent as the result of an inline query.

latitude
  float – Latitude of the location in degrees.

longitude
  float – Longitude of the location in degrees.

title
  str – Name of the venue.

address
  str – Address of the venue.

foursquare_id
  str – Optional. Foursquare identifier of the venue, if known.

foursquare_type
  str – Optional. Foursquare type of the venue, if known. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”.)

Parameters
  • latitude (float) – Latitude of the location in degrees.
  • longitude (float) – Longitude of the location in degrees.
  • title (str) – Name of the venue.
  • address (str) – Address of the venue.
  • foursquare_id (str, optional) – Foursquare identifier of the venue, if known.
  • foursquare_type (str, optional) – Foursquare type of the venue, if known. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”.)
**kwargs (dict) – Arbitrary keyword arguments.

1.45.27 telegram.InputContactMessageContent

class telegram.InputContactMessageContent(phone_number, first_name, last_name=None, vcard=None, **kwargs)

Bases: telegram.inline.inputmessagecontent.InputMessageContent

 Represents the content of a contact message to be sent as the result of an inline query.

phone_number
str – Contact’s phone number.

first_name
str – Contact’s first name.

last_name
str – Optional. Contact’s last name.

vcard
str – Optional. Additional data about the contact in the form of a vCard, 0-2048 bytes.

Parameters

• phone_number (str) – Contact’s phone number.
• first_name (str) – Contact’s first name.
• last_name (str, optional) – Contact’s last name.
• vcard (str, optional) – Additional data about the contact in the form of a vCard, 0-2048 bytes.
• **kwargs (dict) – Arbitrary keyword arguments.

1.45.28 telegram.ChosenInlineResult

class telegram.ChosenInlineResult(result_id, from_user, query, location=None, inline_message_id=None, **kwargs)

Bases: telegram.base.TelegramObject

 Represents a result of an inline query that was chosen by the user and sent to their chat partner.

Note: In Python from is a reserved word, use from_user instead.

result_id
str – The unique identifier for the result that was chosen.

from_user
telegram.User – The user that chose the result.

location
telegram.Location – Optional. Sender location.

inline_message_id
str – Optional. Identifier of the sent inline message.

query
str – The query that was used to obtain the result.

Parameters

• result_id (str) – The unique identifier for the result that was chosen.
Python Telegram Bot Documentation, Release 12.0.0b1

• **from_user** (*telegram.User*) – The user that chose the result.

• **location** (*telegram.Location*, optional) – Sender location, only for bots that require user location.

• **inline_message_id** (*str*, optional) – Identifier of the sent inline message. Available only if there is an inline keyboard attached to the message. Will be also received in callback queries and can be used to edit the message.

• **query** (*str*) – The query that was used to obtain the result.

• **kwargs** (*dict*) – Arbitrary keyword arguments.

### 1.46 Payments

#### 1.46.1 telegram.LabeledPrice

class *telegram.LabeledPrice* (*label*, *amount*, **kwargs)

This object represents a portion of the price for goods or services.

- **label** (*str*) – Portion label.
- **amount** (*int*) – Price of the product in the smallest units of the currency.

Parameters

- **label** (*str*) – Portion label
- **amount** (*int*) – Price of the product in the smallest units of the currency (integer, not float/double). For example, for a price of US$ 1.45 pass amount = 145. See the exp parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).

- **kwargs** (*dict*) – Arbitrary keyword arguments.

#### 1.46.2 telegram.Invoice

class *telegram.Invoice* (*title*, *description*, *start_parameter*, *currency*, *total_amount*, **kwargs)

This object contains basic information about an invoice.

- **title** (*str*) – Product name.
- **description** (*str*) – Product description.
- **start_parameter** (*str*) – Unique bot deep-linking parameter.
- **currency** (*str*) – Three-letter ISO 4217 currency code.
- **total_amount** (*int*) – Total price in the smallest units of the currency.

Parameters

- **title** (*str*) – Product name.
• description (str) – Product description.
• start_parameter (str) – Unique bot deep-linking parameter that can be used to generate this invoice.
• currency (str) – Three-letter ISO 4217 currency code.
• total_amount (int) – Total price in the smallest units of the currency (integer, not float/double). For example, for a price of US$ 1.45 pass amount = 145.
• **kwargs (dict) – Arbitrary keyword arguments.

1.46.3 telegram.ShippingAddress

class telegram.ShippingAddress(country_code, state, city, street_line1, street_line2, post_code, **kwargs)
Bases: telegram.base.TelegramObject

This object represents a Telegram ShippingAddress.

country_code
str – ISO 3166-1 alpha-2 country code.

state
str – State, if applicable.

city
str – City.

street_line1
str – First line for the address.

street_line2
str – Second line for the address.

post_code
str – Address post code.

Parameters

• country_code (str) – ISO 3166-1 alpha-2 country code.
• state (str) – State, if applicable.
• city (str) – City.
• street_line1 (str) – First line for the address.
• street_line2 (str) – Second line for the address.
• post_code (str) – Address post code.
• **kwargs (dict) – Arbitrary keyword arguments.

1.46.4 telegram.OrderInfo

class telegram.OrderInfo(name=None, phone_number=None, email=None, shipping_address=None, **kwargs)
Bases: telegram.base.TelegramObject

This object represents information about an order.

name
str – Optional. User name.

phone_number
str – Optional. User’s phone number.
email
  str – Optional. User email.

shipping_address
  telegram.ShippingAddress – Optional. User shipping address.

Parameters
  • name (str, optional) – User name.
  • phone_number (str, optional) – User’s phone number.
  • email (str, optional) – User email.
  • shipping_address (telegram.ShippingAddress, optional) – User ship-
    ping address.
  • **kwargs (dict) – Arbitrary keyword arguments.

1.46.5 telegram.ShippingOption

class telegram.ShippingOption (id, title, prices, **kwargs)
Bases: telegram.base.TelegramObject
This object represents one shipping option.

  id
    str – Shipping option identifier.

  title
    str – Option title.

  prices

Parameters
  • id (str) – Shipping option identifier.
  • title (str) – Option title.
  • prices (List[telegram.LabeledPrice]) – List of price portions.
  • **kwargs (dict) – Arbitrary keyword arguments.

1.46.6 telegram.SuccessfulPayment

class telegram.SuccessfulPayment (currency, total_amount, invoice_payload, tel-
gram_payment_charge_id, provider_payment_charge_id, shipping_option_id=None, order_info=None, **kwargs)
Bases: telegram.base.TelegramObject
This object contains basic information about a successful payment.

  currency
    str – Three-letter ISO 4217 currency code.

  total_amount
    int – Total price in the smallest units of the currency.

  invoice_payload
    str – Bot specified invoice payload.

  shipping_option_id
    str – Optional. Identifier of the shipping option chosen by the user.
**order_info**

*telegram.OrderInfo* – Optional. Order info provided by the user.

**telegram_payment_charge_id**

*str* – Telegram payment identifier.

**provider_payment_charge_id**

*str* – Provider payment identifier.

### Parameters

- **currency** *(str)* – Three-letter ISO 4217 currency code.
- **total_amount** *(int)* – Total price in the smallest units of the currency (integer, not float/double). For example, for a price of US$ 1.45 pass amount = 145. See the exp parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- **invoice_payload** *(str)* – Bot specified invoice payload.
- **shipping_option_id** *(str, optional)* – Identifier of the shipping option chosen by the user.
- **order_info** *(telegram.OrderInfo, optional)* – Order info provided by the user.
- **telegram_payment_charge_id** *(str)* – Telegram payment identifier.
- **provider_payment_charge_id** *(str)* – Provider payment identifier.
- ****kwargs** *(dict)* – Arbitrary keyword arguments.

#### 1.46.7 *telegram.ShippingQuery*

**class** *telegram.ShippingQuery*(id, from_user, invoice_payload, shipping_address, bot=None, **kwargs)

*Based on:* *telegram.base.TelegramObject*

This object contains information about an incoming shipping query.

---

**Note:**

- In Python *from* is a reserved word, use *from_user* instead.

**id**

*str* – Unique query identifier.

**from_user**

*telegram.User* – User who sent the query.

**invoice_payload**

*str* – Bot specified invoice payload.

**shipping_address**

*telegram.ShippingAddress* – User specified shipping address.

**bot**

*telegram.Bot* – Optional. The Bot to use for instance methods.

### Parameters

- **id** *(str)* – Unique query identifier.
- **from_user** *(telegram.User)* – User who sent the query.
- **invoice_payload** *(str)* – Bot specified invoice payload.
• **shipping_address** ([`telegram.ShippingAddress`](#)) – User specified shipping address.

• **bot** ([`telegram.Bot`](#), optional) – The Bot to use for instance methods.

• **kwargs** (dict) – Arbitrary keyword arguments.

```python
answer(*args, **kwargs)
```

Shortcut for:

```python
bot.answer_shipping_query(update.shipping_query.id, *args, **kwargs)
```

**Parameters**

• **ok** (bool) – Specify True if delivery to the specified address is possible and False if there are any problems (for example, if delivery to the specified address is not possible).

• **shipping_options** (List[`telegram.ShippingOption`], optional) – Required if ok is True. A JSON-serialized array of available shipping options.

• **error_message** (str, optional) – Required if ok is False. Error message in human readable form that explains why it is impossible to complete the order (e.g. ‘Sorry, delivery to your desired address is unavailable’). Telegram will display this message to the user.

1.46.8 **telegram.PreCheckoutQuery**

```python
class telegram.PreCheckoutQuery(id, from_user, currency, total_amount, invoice_payload, shipping_option_id=None, order_info=None, bot=None, **kwargs)
```

**Bases:** `telegram.base.TelegramObject`

This object contains information about an incoming pre-checkout query.

**Note:**

• In Python `from` is a reserved word, use `from_user` instead.

**id**

str – Unique query identifier.

**from_user**

`telegram.User` – User who sent the query.

**currency**

str – Three-letter ISO 4217 currency code.

**total_amount**

int – Total price in the smallest units of the currency.

**invoice_payload**

str – Bot specified invoice payload.

**shipping_option_id**

str – Optional. Identifier of the shipping option chosen by the user.

**order_info**

`telegram.OrderInfo` – Optional. Order info provided by the user.

**bot**

Parameters

- **id** *(str)* – Unique query identifier.
- **from_user** *(telegram.User)* – User who sent the query.
- **currency** *(str)* – Three-letter ISO 4217 currency code
- **total_amount** *(int)* – Total price in the smallest units of the currency (integer, not float/double). For example, for a price of US$ 1.45 pass amount = 145. See the exp parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- **invoice_payload** *(str)* – Bot specified invoice payload.
- **shipping_option_id** *(str, optional)* – Identifier of the shipping option chosen by the user.
- **order_info** *(telegram.OrderInfo, optional)* – Order info provided by the user.
- **bot** *(telegram.Bot, optional)* – The Bot to use for instance methods.
- ****kwargs** *(dict)* – Arbitrary keyword arguments.

```python
answer(*args, **kwargs)
```

Shortcut for:

```python
bot.answer_pre_checkout_query(update.pre_checkout_query.id, *args, **kwargs)
```

Parameters

- **ok** *(bool)* – Specify True if everything is alright (goods are available, etc.) and the bot is ready to proceed with the order. Use False if there are any problems.
- **error_message** *(str, optional)* – Required if ok is False. Error message in human readable form that explains the reason for failure to proceed with the checkout (e.g. “Sorry, somebody just bought the last of our amazing black T-shirts while you were busy filling out your payment details. Please choose a different color or garment!”). Telegram will display this message to the user.
- ****kwargs** *(dict)* – Arbitrary keyword arguments.

1.47 Games

1.47.1 telegram.Game

```python
class telegram.Game(title, description, photo, text=None, text_entities=None, animation=None, **kwargs)
```

Bases: telegram.base.TelegramObject

This object represents a game. Use BotFather to create and edit games, their short names will act as unique identifiers.

- **title** *(str)* – Title of the game.
- **description** *(str)* – Description of the game.
- **photo** *(List[telegram.PhotoSize]*) – Photo that will be displayed in the game message in chats.
text
str – Optional. Brief description of the game or high scores included in the game message. Can be automatically edited to include current high scores for the game when the bot calls set_game_score, or manually edited using edit_message_text.

text_entities
List[telegram.MessageEntity] – Optional. Special entities that appear in text, such as usernames, URLs, bot commands, etc.

animation
telegram.Animation – Optional. Animation that will be displayed in the game message in chats. Upload via BotFather.

Parameters

- title (str) – Title of the game.
- description (str) – Description of the game.
- photo (List[telegram.PhotoSize]) – Photo that will be displayed in the game message in chats.
- text (str, optional) – Brief description of the game or high scores included in the game message. Can be automatically edited to include current high scores for the game when the bot calls set_game_score, or manually edited using edit_message_text. 0-4096 characters. Also found as telegram.constants.MAX_MESSAGE_LENGTH.
- text_entities (List[telegram.MessageEntity], optional) – Special entities that appear in text, such as usernames, URLs, bot commands, etc.
- animation (telegram.Animation, optional) – Animation that will be displayed in the game message in chats. Upload via BotFather.

parse_text_entities (types=None)
Returns a dict that maps telegram.MessageEntity to str. It contains entities from this message filtered by their type attribute as the key, and the text that each entity belongs to as the value of the dict.

Note: This method should always be used instead of the text_entities attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See parse_text_entity for more info.

Parameters types (List[str], optional) – List of MessageEntity types as strings. If the type attribute of an entity is contained in this list, it will be returned. Defaults to telegram.MessageEntity.ALL_TYPES.

Returns A dictionary of entities mapped to the text that belongs to them, calculated based on UTF-16 codepoints.

Return type Dict[telegram.MessageEntity, str]

parse_text_entity (entity)
Returns the text from a given telegram.MessageEntity.

Note: This method is present because Telegram calculates the offset and length in UTF-16 codepoint pairs, which some versions of Python don’t handle automatically. (That is, you can’t just slice Message.text with the offset and length.)

Parameters entity (telegram.MessageEntity) – The entity to extract the text from. It must be an entity that belongs to this message.
Returns  The text of the given entity.
Return type  str

1.47.2  telegram.Callbackgame

class  telegram.CallbackGame
    Bases:  telegram.base.TelegramObject
    A placeholder, currently holds no information. Use BotFather to set up your game.

1.47.3  telegram.GameHighScore

class  telegram.GameHighScore(position, user, score)
    Bases:  telegram.base.TelegramObject
    This object represents one row of the high scores table for a game.
    position
        int – Position in high score table for the game.
    user
    score
        int – Score.

Parameters

- position (int) – Position in high score table for the game.
- user (telegram.User) – User.
- score (int) – Score.

1.48  Passport

1.48.1  telegram.PassportElementError

class  telegram.PassportElementError(source, type, message, **kwargs)
    Bases:  telegram.base.TelegramObject
    Baseclass for the PassportElementError* classes.
    source
        str – Error source.
    type
        str – The section of the user’s Telegram Passport which has the error.
    message
        str – Error message

Parameters

- source (str) – Error source.
- type (str) – The section of the user’s Telegram Passport which has the error.
- **kwargs (dict) – Arbitrary keyword arguments.
1.48.2 `telegram.PassportElementErrorFile`

```python
class telegram.PassportElementErrorFile(type, file_hash, message, **kwargs):
    Bases: telegram.passport.passportelementerrors.PassportElementError
    Represents an issue with a document scan. The error is considered resolved when the file with the document
    scan changes.

    type
        str – The section of the user’s Telegram Passport which has the issue, one of “utility_bill”,
             “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.

    file_hash
        str – Base64-encoded file hash.

    message
        str – Error message.
```

Parameters

- `type (str)` – The section of the user’s Telegram Passport which has the issue, one of “utility_bill”,
  “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.
- `file_hash (str)` – Base64-encoded file hash.
- `message (str)` – Error message.
- `**kwargs (dict)` – Arbitrary keyword arguments.

1.48.3 `telegram.PassportElementErrorReverseSide`

```python
class telegram.PassportElementErrorReverseSide(type, file_hash, message, **kwargs):
    Bases: telegram.passport.passportelementerrors.PassportElementError
    Represents an issue with the front side of a document. The error is considered resolved when the file with
    the reverse side of the document changes.

    type
        str – The section of the user’s Telegram Passport which has the issue, one of “passport”,
             “driver_license”, “identity_card”, “internal_passport”.

    file_hash
        str – Base64-encoded hash of the file with the reverse side of the document.

    message
        str – Error message.
```

Parameters

- `type (str)` – The section of the user’s Telegram Passport which has the issue, one of “passport”,
  “driver_license”, “identity_card”.
- `file_hash (str)` – Base64-encoded hash of the file with the reverse side of the document.
- `message (str)` – Error message.
- `**kwargs (dict)` – Arbitrary keyword arguments.

1.48.4 `telegram.PassportElementErrorFrontSide`

```python
class telegram.PassportElementErrorFrontSide(type, file_hash, message, **kwargs):
    Bases: telegram.passport.passportelementerrors.PassportElementError
```

170 Chapter 1. telegram package
Represents an issue with the front side of a document. The error is considered resolved when the file with the front side of the document changes.

```python
type
def type(
    str – The section of the user’s Telegram Passport which has the issue, one of “passport”, “driver_license”, “identity_card”, “internal_passport”.

file_hash
def file_hash(
    str – Base64-encoded hash of the file with the front side of the document.

message
def message(
    str – Error message.

Parameters

- **type**(str) – The section of the user’s Telegram Passport which has the issue, one of “passport”, “driver_license”, “identity_card”, “internal_passport”.
- **file_hash**(str) – Base64-encoded hash of the file with the front side of the document.
- **message**(str) – Error message.
- ****kwags(dict) – Arbitrary keyword arguments.

1.48.5 telegram.PassportElementErrorFiles

class
def telegram.PassportElementErrorFiles(
    type, file_hashes, message, **kwargs)
    Bases: telegram.passport.passportelementerrors.PassportElementError

Represents an issue with a list of scans. The error is considered resolved when the file with the document scan changes.

```python
type
def type(
    str – The section of the user’s Telegram Passport which has the issue, one of “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.

file_hash
def file_hash(
    str – Base64-encoded file hash.

message
def message(
    str – Error message.

Parameters

- **type**(str) – The section of the user’s Telegram Passport which has the issue, one of “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.
- **file_hashes**(List[str]) – List of base64-encoded file hashes.
- **message**(str) – Error message.
- ****kwags(dict) – Arbitrary keyword arguments.

1.48.6 telegram.PassportElementErrorDataField

class
def telegram.PassportElementErrorDataField(
    type, field_name, data_hash, message, **kwargs)
    Bases: telegram.passport.passportelementerrors.PassportElementError

Represents an issue in one of the data fields that was provided by the user. The error is considered resolved when the field’s value changes.

```
**Parameters**

- **`type` (str)** – The section of the user’s Telegram Passport which has the error, one of “personal_details”, “passport”, “driver_license”, “identity_card”, “internal_passport”, “address”.
- **`field_name` (str)** – Name of the data field which has the error.
- **`data_hash` (str)** – Base64-encoded data hash.
- **`message` (str)** – Error message.

1.48.7 **telegram.Credentials**

```python
class telegram.Credentials(secure_data, nonce, bot=None, **kwargs)
Bases: telegram.base.TelegramObject
secure_data  
telegram.SecureData – Credentials for encrypted data
nonce  
str – Bot-specified nonce
```

1.48.8 **telegram.DataCredentials**

```python
class telegram.DataCredentials(data_hash, secret, **kwargs)
Bases: telegram.passport.credentials._CredentialsBase
These credentials can be used to decrypt encrypted data from the data field in EncryptedPassportData.
hash  
str – Checksum of encrypted data
secret  
str – Secret of encrypted data
```
1.48.9 `telegram.SecureData`

```python
class telegram.SecureData (personal_details=None, passport=None, internal_passport=None, 
    driver_license=None, identity_card=None, address=None, utility_bill=None, bank_statement=None, rental_agreement=None, 
    passport_registration=None, temporary_registration=None, bot=None, **kwargs)
```

Bases: `telegram.base.TelegramObject`

This object represents the credentials that were used to decrypt the encrypted data. All fields are optional and depend on fields that were requested.

- **personal_details**

- **passport**

- **internal_passport**
  - `telegram.SecureValue`, optional – Credentials for encrypted internal passport.

- **driver_license**
  - `telegram.SecureValue`, optional – Credentials for encrypted driver license.

- **identity_card**
  - `telegram.SecureValue`, optional – Credentials for encrypted ID card

- **address**
  - `telegram.SecureValue`, optional – Credentials for encrypted residential address.

- **utility_bill**

- **bank_statement**

- **rental_agreement**
  - `telegram.SecureValue`, optional – Credentials for encrypted rental agreement.

- **passport_registration**
  - `telegram.SecureValue`, optional – Credentials for encrypted registration from internal passport.

- **temporary_registration**
  - `telegram.SecureValue`, optional – Credentials for encrypted temporary registration.

1.48.10 `telegram.FileCredentials`

```python
class telegram.FileCredentials (file_hash, secret, **kwargs)
```

Bases: `telegram.passport.credentials._CredentialsBase`

These credentials can be used to decrypt encrypted files from the front_side, reverse_side, selfie and files fields in EncryptedPassportData.

**Parameters**

- **file_hash** (`str`) – Checksum of encrypted file
- **secret** (`str`) – Secret of encrypted file

**hash**

- `str` – Checksum of encrypted file

**secret**

- `str` – Secret of encrypted file
1.48.11 telegram.IdDocumentData

class telegram.IdDocumentData(document_no, expiry_date, bot=None, **kwargs)
    Bases: telegram.base.TelegramObject
    This object represents the data of an identity document.
    
    document_no
    str – Document number.
    
    expiry_date
    str – Optional. Date of expiry, in DD.MM.YYYY format.

1.48.12 telegram.PersonalDetails

class telegram.PersonalDetails(first_name, last_name, birth_date, gender, country_code, residence_country_code, first_name_native=None, last_name_native=None, middle_name=None, middle_name_native=None, bot=None, **kwargs)
    Bases: telegram.base.TelegramObject
    This object represents personal details.
    
    first_name
    str – First Name.
    
    middle_name
    str – Optional. First Name.
    
    last_name
    str – Last Name.
    
    birth_date
    str – Date of birth in DD.MM.YYYY format.
    
    gender
    str – Gender, male or female.
    
    country_code
    str – Citizenship (ISO 3166-1 alpha-2 country code).
    
    residence_country_code
    str – Country of residence (ISO 3166-1 alpha-2 country code).
    
    first_name
    str – First Name in the language of the user’s country of residence.
    
    middle_name
    str – Optional. Middle Name in the language of the user’s country of residence.
    
    last_name
    str – Last Name in the language of the user’s country of residence.

1.48.13 telegram.ResidentialAddress

class telegram.ResidentialAddress(street_line1, street_line2, city, state, country_code, post_code, bot=None, **kwargs)
    Bases: telegram.base.TelegramObject
    This object represents a residential address.
    
    street_line1
    str – First line for the address.
street_line2
   str – Optional. Second line for the address.

city
   str – City.

state
   str – Optional. State.

country_code
   str – ISO 3166-1 alpha-2 country code.

post_code
   str – Address post code.

1.48.14 telegram.PassportData

class telegram.PassportData(data, credentials, bot=None, **kwargs)
Bases: telegram.base.TelegramObject

Contains information about Telegram Passport data shared with the bot by the user.

data
   List[telegram.EncryptedPassportElement] – Array with encrypted information about
documents and other Telegram Passport elements that was shared with the bot.

credentials
   telegram.EncryptedCredentials – Encrypted credentials.

bot

Parameters

• data (List[telegram.EncryptedPassportElement]) – Array with encrypted
  information about documents and other Telegram Passport elements that was shared
  with the bot.

• credentials (str) – Encrypted credentials.

• bot (telegram.Bot, optional) – The Bot to use for instance methods.

• **kwargs (dict) – Arbitrary keyword arguments.

Note: To be able to decrypt this object, you must pass your private_key to either telegram.Updater
or telegram.Bot. Decrypted data is then found in decrypted_data and the payload can be found

decrypted_credentials
   telegram.Credentials –

   Lazily decrypt and return credentials that were used to decrypt the data. This object also contains
   the user specified payload as decrypted_data.payload.

Raises telegram.TelegramDecryptionError – Decryption failed. Usually due to
   bad private/public key but can also suggest malformed/tampered data.

decrypted_data
   List[telegram.EncryptedPassportElement] –

   Lazily decrypt and return information about documents and other Telegram Passport elements
   which were shared with the bot.
**Raises** `telegram.TelegramDecryptionError` – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

### 1.48.15 `telegram.PassportFile`

#### class `telegram.PassportFile`

```python
class telegram.PassportFile(file_id, file_date, file_size=None, bot=None, credentials=None, **kwargs)
```

Bases: `telegram.base.TelegramObject`

This object represents a file uploaded to Telegram Passport. Currently all Telegram Passport files are in JPEG format when decrypted and don’t exceed 10MB.

- **file_id** `str` – Unique identifier for this file.
- **file_size** `int` – File size.
- **file_date** `int` – Unix time when the file was uploaded.
- **bot** `telegram.Bot` – Optional. The Bot to use for instance methods.

**Parameters**

- `file_id` (str) – Unique identifier for this file.
- `file_size` (int) – File size.
- `file_date` (int) – Unix time when the file was uploaded.
- `bot` (`telegram.Bot`, optional) – The Bot to use for instance methods.
- `**kwargs` (dict) – Arbitrary keyword arguments.

#### `get_file` *(timeout=None, **kwargs)*


**Parameters**

- `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- `**kwargs` (dict) – Arbitrary keyword arguments.

**Returns** `telegram.File`

**Raises** `telegram.TelegramError`

### 1.48.16 `telegram.EncryptedPassportElement`

#### class `telegram.EncryptedPassportElement`

```python
class telegram.EncryptedPassportElement(type, data=None, phone_number=None, email=None, files=None, front_side=None, reverse_side=None, selfie=None, translation=None, hash=None, bot=None, credentials=None, **kwargs)
```

Bases: `telegram.base.TelegramObject`

Contains information about documents or other Telegram Passport elements shared with the bot by the user. The data has been automatically decrypted by python-telegram-bot.
type


data

telegram.PersonalDetails or telegram.IdDocument or telegram.ResidentialAddress or str – Optional. Decrypted or encrypted data, available for “personal_details”, “passport”, “driver_license”, “identity_card”, “identity_passport” and “address” types.

phone_number

str – Optional. User’s verified phone number, available only for “phone_number” type.

e-mail

str – Optional. User’s verified email address, available only for “email” type.

files


front_side

telegram.PassportFile – Optional. Encrypted/decrypted file with the front side of the document, provided by the user. Available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

reverse_side

telegram.PassportFile – Optional. Encrypted/decrypted file with the reverse side of the document, provided by the user. Available for “driver_license” and “identity_card”.

selfie

telegram.PassportFile – Optional. Encrypted/decrypted file with the selfie of the user holding a document, provided by the user; available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

translation


hash

str – Base64-encoded element hash for using in telegram.PassportElementErrorUnspecified.

bot


Parameters


• data (telegram.PersonalDetails or telegram.IdDocument or telegram.ResidentialAddress or str, optional) – Decrypted or encrypted data, available for “personal_details”, “passport”, “driver_license”, “identity_card”, “identity_passport” and “address” types.

• phone_number (str, optional) – User’s verified phone number, available only for “phone_number” type.

• email (str, optional) – User’s verified email address, available only for “email” type.
• **files** (List[telegram.PassportFile], optional) – Array of encrypted/decrypted files with documents provided by the user, available for “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.

• **front_side** (telegram.PassportFile, optional) – Encrypted/decrypted file with the front side of the document, provided by the user. Available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

• **reverse_side** (telegram.PassportFile, optional) – Encrypted/decrypted file with the reverse side of the document, provided by the user. Available for “driver_license” and “identity_card”.

• **selfie** (telegram.PassportFile, optional) – Encrypted/decrypted file with the selfie of the user holding a document, provided by the user; available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

• **translation** (List[telegram.PassportFile], optional) – Array of encrypted/decrypted files with translated versions of documents provided by the user. Available if requested for “passport”, “driver_license”, “identity_card”, “internal_passport”, “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.

• **hash** (str) – Base64-encoded element hash for using in telegram.PassportElementErrorUnspecified.

• **bot** (telegram.Bot, optional) – The Bot to use for instance methods.

• ****kwargs (dict) – Arbitrary keyword arguments.

---

**Note:** This object is decrypted only when originating from telegram.PassportData.decrypted_data.

### 1.48.17 telegram.EncryptedCredentials

class telegram.EncryptedCredentials(data, hash, secret, bot=None, **kwargs)

Bases: telegram.base.TelegramObject

Contains data required for decrypting and authenticating EncryptedPassportElement. See the Telegram Passport Documentation for a complete description of the data decryption and authentication processes.

data

telegram.Credentials or str – Decrypted data with unique user’s nonce, data hashes and secrets used for EncryptedPassportElement decryption and authentication or base64 encrypted data.

hash

str – Base64-encoded data hash for data authentication.

secret

str – Decrypted or encrypted secret used for decryption.

Parameters

• **data** (telegram.Credentials or str) – Decrypted data with unique user’s nonce, data hashes and secrets used for EncryptedPassportElement decryption and authentication or base64 encrypted data.

• **hash** (str) – Base64-encoded data hash for data authentication.

• **secret** (str) – Decrypted or encrypted secret used for decryption.

• ****kwargs (dict) – Arbitrary keyword arguments.
decrypted_data

telegram.Credentials – Lazily decrypt and return credentials data. This object also contains the user specified nonce as decrypted_data.nonce.

Raises telegram.TelegramDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

decrypted_secret

str – Lazily decrypt and return secret.

Raises telegram.TelegramDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

1.49 Module contents

A library that provides a Python interface to the Telegram Bot API

class telegram.Audio(file_id, duration, performer=None, title=None, mime_type=None,
    file_size=None, thumb=None, bot=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents an audio file to be treated as music by the Telegram clients.

file_id

str – Unique identifier for this file.

duration

int – Duration of the audio in seconds.

performer

str – Optional. Performer of the audio as defined by sender or by audio tags.

title

str – Optional. Title of the audio as defined by sender or by audio tags.

mime_type

str – Optional. MIME type of the file as defined by sender.

file_size

int – Optional. File size.

thumb

telegram.PhotoSize – Optional. Thumbnail of the album cover to which the music file belongs.

bot


Parameters

• file_id(str) – Unique identifier for this file.

• duration(int) – Duration of the audio in seconds as defined by sender.

• performer(str, optional) – Performer of the audio as defined by sender or by audio tags.

• title(str, optional) – Title of the audio as defined by sender or by audio tags.

• mime_type(str, optional) – MIME type of the file as defined by sender.
```
• **file_size** (int, optional) – File size.
• **thumb** (*telegram.PhotoSize*, optional) – Thumbnail of the album cover to which
  the music file belongs.
• **bot** (*telegram.Bot*, optional) – The Bot to use for instance methods.
• ****kwargs** (dict) – Arbitrary keyword arguments.

classmethod **de_json**(data, **bot**)

**get_file**(timeout=None, ****kwargs**)

Convenience wrapper over *telegram.Bot.get_file*

Parameters

• **timeout** (int | float, optional) – If this value is specified, use it as the read
  timeout from the server (instead of the one specified during creation of the connection
  pool).

• ****kwargs** (dict) – Arbitrary keyword arguments.

Returns **telegram.File**

 Raises **telegram.TelegramError**

class **telegram.Bot**(token, base_url=None, base_file_url=None, request=None, pri-
   vate_key=None, private_key_password=None)

Bases: *telegram.base.TelegramObject*

This object represents a Telegram Bot.

Parameters

• **token** (str) – Bot's unique authentication.

• **base_url** (str, optional) – Telegram Bot API service URL.

• **base_file_url** (str, optional) – Telegram Bot API file URL.

• **request** (*telegram.utils.request.Request*, optional) – Pre initialized
  *telegram.utils.request.Request*.

• **private_key** (bytes, optional) – Private key for decryption of telegram passport
  data.

• **private_key_password** (bytes, optional) – Password for above private key.

**addStickerToSet**(user_id, name, png_sticker, emojis, mask_position=None, timeout=20,
  ****kwargs**)

Alias for **add_sticker_to_set**

**add_sticker_to_set**(user_id, name, png_sticker, emojis, mask_position=None, timeout=20,
  ****kwargs**)

Use this method to add a new sticker to a set created by the bot.

**Note:** The *png_sticker* argument can be either a file_id, an URL or a file from disk
open(filename, 'rb')
the Telegram servers, pass an HTTP URL as a String for Telegram to get a file from
the Internet, or upload a new one using multipart/form-data.

- **emojis** *(str)* – One or more emoji corresponding to the sticker.
- **mask_position** *(telegram.MaskPosition, optional)* – Position where the
  mask should be placed on faces.
- **timeout** *(int | float, optional)* – If this value is specified, use it as the read
  timeout from the server (instead of the one specified during creation of the connection
  pool).

- ****kwargs**(dict)* – Arbitrary keyword arguments.

**Returns** On success, True is returned.

**Return type** bool

**Raises** telegram.TelegramError

**answerCallbackQuery** *(callback_query_id=None, text=None, show_alert=False, url=None,
  cache_time=None, timeout=None, **kwargs)*

Alias for **answer_callback_query**

**answerInlineQuery** *(inline_query_id, results, cache_time=300, is_personal=None,
  next_offset=None, switch_pm_text=None, switch_pm_parameter=None,
  timeout=None, **kwargs)*

Alias for **answer_inline_query**

**answerPreCheckoutQuery** *(pre_checkout_query_id, ok, error_message=None,
  timeout=None, **kwargs)*

Alias for **answer_pre_checkout_query**

**answerShippingQuery** *(shipping_query_id, ok, shipping_options=None, error_message=None,
  timeout=None, **kwargs)*

Alias for **answer_shipping_query**

**answer_callback_query** *(callback_query_id=None, text=None, show_alert=False, url=None,
  cache_time=None, timeout=None, **kwargs)*

Use this method to send answers to callback queries sent from inline keyboards. The answer
will be displayed to the user as a notification at the top of the chat screen or as an alert. Alternatively,
the user can be redirected to the specified Game URL. For this option to work, you must first cre-
ate a game for your bot via BotFather and accept the terms. Otherwise, you may use links like
t.me/your_bot?start=XXXX that open your bot with a parameter.

**Parameters**

- **callback_query_id** *(str)* – Unique identifier for the query to be answered.
- **text** *(str, optional)* – Text of the notification. If not specified, nothing will be
  shown to the user, 0-200 characters.
- **show_alert** *(bool, optional)* – If true, an alert will be shown by the client instead
  of a notification at the top of the chat screen. Defaults to false.
- **url** *(str, optional)* – URL that will be opened by the user’s client. If you have
  created a Game and accepted the conditions via @Botfather, specify the URL that
  opens your game - note that this will only work if the query comes from a callback
  game button. Otherwise, you may use links like t.me/your_bot?start=XXXX that open
  your bot with a parameter.
- **cache_time** *(int, optional)* – The maximum amount of time in seconds that the
  result of the callback query may be cached client-side. Defaults to 0.
- **timeout** *(int | float, optional)* – If this value is specified, use it as the read
  timeout from the server (instead of the one specified during creation of the connection
  pool).
- ****kwargs**(dict)* – Arbitrary keyword arguments.
Returns bool On success, True is returned.

Raises telegram.TelegramError

answer_inline_query(inline_query_id, results, cache_time=300, is_personal=None, next_offset=None, switch_pm_text=None, switch_pm_parameter=None, timeout=None, **kwargs)

Use this method to send answers to an inline query. No more than 50 results per query are allowed.

Parameters

- inline_query_id (str) – Unique identifier for the answered query.
- results (List[telegram.InlineQueryResult]) – A list of results for the inline query.
- cache_time (int, optional) – The maximum amount of time in seconds that the result of the inline query may be cached on the server. Defaults to 300.
- is_personal (bool, optional) – Pass True, if results may be cached on the server side only for the user that sent the query. By default, results may be returned to any user who sends the same query.
- next_offset (str, optional) – Pass the offset that a client should send in the next query with the same text to receive more results. Pass an empty string if there are no more results or if you don’t support pagination. Offset length can’t exceed 64 bytes.
- switch_pm_text (str, optional) – If passed, clients will display a button with specified text that switches the user to a private chat with the bot and sends the bot a start message with the parameter switch_pm_parameter.
- switch_pm_parameter (str, optional) – Deep-linking parameter for the /start message sent to the bot when user presses the switch button. 1-64 characters, only A-Z, a-z, 0-9, _ and - are allowed.
- timeout (int|float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **kwargs (dict) – Arbitrary keyword arguments.

Example

An inline bot that sends YouTube videos can ask the user to connect the bot to their YouTube account to adapt search results accordingly. To do this, it displays a ‘Connect your YouTube account’ button above the results, or even before showing any. The user presses the button, switches to a private chat with the bot and, in doing so, passes a start parameter that instructs the bot to return an oauth link. Once done, the bot can offer a switch_inline button so that the user can easily return to the chat where they wanted to use the bot’s inline capabilities.

Returns bool On success, True is returned.

Raises telegram.TelegramError

answer_pre_checkout_query(pre_checkout_query_id, ok, error_message=None, timeout=None, **kwargs)

Once the user has confirmed their payment and shipping details, the Bot API sends the final confirmation in the form of an Update with the field pre_checkout_query. Use this method to respond to such pre-checkout queries.

Note: The Bot API must receive an answer within 10 seconds after the pre-checkout query was sent.

Parameters
• `pre_checkout_query_id` *(str)* – Unique identifier for the query to be answered.

• `ok` *(bool)* – Specify True if everything is alright (goods are available, etc.) and the bot is ready to proceed with the order. Use False if there are any problems.

• `error_message` *(str, optional)* – Required if ok is False. Error message in human readable form that explains the reason for failure to proceed with the checkout (e.g. “Sorry, somebody just bought the last of our amazing black T-shirts while you were busy filling out your payment details. Please choose a different color or garment!”). Telegram will display this message to the user.

• `timeout` *(int | float, optional)* – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **`**kwargs**` *(dict)* – Arbitrary keyword arguments.

**Returns** On success, True is returned.

**Return type** bool

**Raises** `telegram.TelegramError`

```python
answer_shipping_query(shipping_query_id, ok, shipping_options=None, error_message=None, timeout=None, **kwargs)
```

If you sent an invoice requesting a shipping address and the parameter is_flexible was specified, the Bot API will send an Update with a shipping_query field to the bot. Use this method to reply to shipping queries.

**Parameters**

• `shipping_query_id` *(str)* – Unique identifier for the query to be answered.

• `ok` *(bool)* – Specify True if delivery to the specified address is possible and False if there are any problems (for example, if delivery to the specified address is not possible).

• `shipping_options` *(List[telegram.ShippingOption])* – Required if ok is True. A JSON-serialized array of available shipping options.

• `error_message` *(str, optional)* – Required if ok is False. Error message in human readable form that explains why it is impossible to complete the order (e.g. “Sorry, delivery to your desired address is unavailable”). Telegram will display this message to the user.

• `timeout` *(int | float, optional)* – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **`**kwargs**` *(dict)* – Arbitrary keyword arguments.

**Returns** bool; On success, True is returned.

**Raises** `telegram.TelegramError`

```python
createNewStickerSet(user_id, name, title, png_sticker, emojis, contains_masks=None, mask_position=None, timeout=20, **kwargs)
```

Alias for `create_new_sticker_set`

**create_new_sticker_set**(user_id, name, title, png_sticker, emojis, contains_masks=None, mask_position=None, timeout=20, **kwargs)

Use this method to create new sticker set owned by a user.

The bot will be able to edit the created sticker set.
Note: The `png_sticker` argument can be either a file_id, an URL or a file from disk.

```python
text(open(filename, 'rb'))
```

**Parameters**

- `user_id` *(int)* – User identifier of created sticker set owner.
- `name` *(str)* – Short name of sticker set, to be used in t.me/addstickers/ URLs (e.g., animals). Can contain only English letters, digits and underscores. Must begin with a letter, can’t contain consecutive underscores and must end in “_<bot_username>”.
  `<bot_username>` is case insensitive. 1-64 characters.
- `title` *(str)* – Sticker set title, 1-64 characters.
- `png_sticker` *(str | filelike object)* – Png image with the sticker, must be up to 512 kilobytes in size, dimensions must not exceed 512px, and either width or height must be exactly 512px. Pass a file_id as a String to send a file that already exists on the Telegram servers, pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data.
- `emojis` *(str)* – One or more emoji corresponding to the sticker.
- `contains_masks` *(bool, optional)* – Pass True, if a set of mask stickers should be created.
- `mask_position` *(telegram.MaskPosition, optional)* – Position where the mask should be placed on faces.
- `timeout` *(int | float, optional)* – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **kwargs** *(dict)* – Arbitrary keyword arguments.

**Returns**

On success, True is returned.

**Return type** `bool`

**Raises** `telegram.TelegramError`

```python
deflectChatPhoto (chat_id, timeout=None, **kwargs)
```

Alias for `delete_chat_photo`

```python
deflectChatStickerSet (chat_id, timeout=None, **kwargs)
```

Alias for `delete_chat_sticker_set`

```python
deflectMessage (chat_id, message_id, timeout=None, **kwargs)
```

Alias for `delete_message`

```python
deflectStickerFromSet (sticker, timeout=None, **kwargs)
```

Alias for `delete_sticker_from_set`

```python
deflectWebhook (timeout=None, **kwargs)
```

Alias for `delete_webhook`

```python
deflect_chat_photo (chat_id, timeout=None, **kwargs)
```

Use this method to delete a chat photo. Photos can’t be changed for private chats. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

**Parameters**

- `chat_id` *(int | str)* – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- `timeout` *(int | float, optional)* – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
**kwargs (dict) – Arbitrary keyword arguments

Note: In regular groups (non-supergroups), this method will only work if the ‘All Members Are Admins’ setting is off in the target group.

Returns Returns True on success.

Return type bool

Raises telegram.TelegramError

delete_chat_sticker_set (chat_id, timeout=None, **kwargs)

Use this method to delete a group sticker set from a supergroup. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights. Use the field `telegram.Chat.can_set_sticker_set` optionally returned in `get_chat` requests to check if the bot can use this method.

Parameters

- chat_id (int | str) – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername).
- timeout (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **kwargs (dict) – Arbitrary keyword arguments.

Returns True on success.

Return type bool

delete_message (chat_id, message_id, timeout=None, **kwargs)

Use this method to delete a message. A message can only be deleted if it was sent less than 48 hours ago. Any such recently sent outgoing message may be deleted. Additionally, if the bot is an administrator in a group chat, it can delete any message. If the bot is an administrator in a supergroup, it can delete messages from any other user and service messages about people joining or leaving the group (other types of service messages may only be removed by the group creator). In channels, bots can only remove their own messages.

Parameters

- chat_id (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- message_id (int) – Identifier of the message to delete.
- timeout (int | float, optional) – If this value is specified, use it as
- read timeout (the) – from the server (instead of the one specified during creation of the connection pool).
- **kwargs (dict) – Arbitrary keyword arguments.

Returns On success, True is returned.

Return type bool

Raises telegram.TelegramError

delete_sticker_from_set (sticker, timeout=None, **kwargs)

Use this method to delete a sticker from a set created by the bot.

Parameters

- sticker (str) – File identifier of the sticker.
**timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

**kwargs (dict) – Arbitrary keyword arguments.

**Returns** On success, True is returned.

**Return type** bool

**Raises** telegram.TelegramError

delete_webhook (timeout=None, **kwargs)

Use this method to remove webhook integration if you decide to switch back to getUpdates. Requires no parameters.

**Parameters**

**timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

**kwargs (dict) – Arbitrary keyword arguments.

**Returns** bool On success, True is returned.

** Raises ** telegram.TelegramError

editMessageCaption (chat_id=None, message_id=None, inline_message_id=None, caption=None, reply_markup=None, timeout=None, parse_mode=None, **kwargs)

Alias for edit_message_caption

editMessageLiveLocation (chat_id=None, message_id=None, inline_message_id=None, latitude=None, longitude=None, location=None, reply_markup=None, timeout=None, **kwargs)

Alias for edit_message_live_location

editMessageMedia (chat_id=None, message_id=None, inline_message_id=None, media=None, reply_markup=None, timeout=None, **kwargs)

Alias for edit_message_media

editMessageReplyMarkup (chat_id=None, message_id=None, inline_message_id=None, reply_markup=None, timeout=None, **kwargs)

Alias for edit_message_reply_markup

editMessageText (text, chat_id=None, message_id=None, inline_message_id=None, parse_mode=None, disable_web_page_preview=None, reply_markup=None, timeout=None, **kwargs)

Alias for edit_message_text

edit_message_caption (chat_id=None, message_id=None, inline_message_id=None, caption=None, reply_markup=None, timeout=None, parse_mode=None, **kwargs)

Use this method to edit captions of messages sent by the bot or via the bot (for inline bots).

**Parameters**

**chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

**message_id** (int, optional) – Required if inline_message_id is not specified. Identifier of the sent message.

**inline_message_id** (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.

**caption** (str, optional) – New caption of the message.
• **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

• **reply_markup** (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

• **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• ****kwargs (dict) – Arbitrary keyword arguments.

Returns
On success, if edited message is sent by the bot, the edited Message is returned, otherwise True is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

edit_message_live_location

Use this method to edit live location messages sent by the bot or via the bot (for inline bots). A location can be edited until its live_period expires or editing is explicitly disabled by a call to `stop_message_live_location`.

**Parameters**

• **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• **message_id** (int, optional) – Required if inline_message_id is not specified. Identifier of the sent message.

• **inline_message_id** (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.

• **latitude** (float, optional) – Latitude of location.

• **longitude** (float, optional) – Longitude of location.

• **location** (telegram.Location, optional) – The location to send.

• **reply_markup** (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

• **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

Returns
On success the edited message.

Return type `telegram.Message`

edit_message_media

Use this method to edit audio, document, photo, or video messages. If a message is a part of a message album, then it can be edited only to a photo or a video. Otherwise, message type can be changed arbitrarily. When inline message is edited, new file can’t be uploaded. Use previously uploaded file
via its file_id or specify a URL. On success, if the edited message was sent by the bot, the edited
Message is returned, otherwise True is returned.

Parameters

- **chat_id** (int | str, optional) – Unique identifier for the target chat or username of
  the target channel (in the format @channelusername).
- **message_id** (int, optional) – Required if inline_message_id is not specified. Identifier
  of the sent message.
- **inline_message_id** (str, optional) – Required if chat_id and message_id are not specified.
  Identifier of the inline message.
- **media** (telegram.InputMedia) – An object for a new media content of the
  message.
- **reply_markup** (telegram.ReplyMarkup, optional) – Additional interface options. A
  JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove
  reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – If this value is specified, use it as the read
  timeout from the server (instead of the one specified during creation of the connection
  pool).
- ****kwars (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 (**kwargs)

Use this method to edit only the reply markup of messages sent by the bot or via the bot (for inline bots).

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target
  channel (in the format @channelusername).
- **message_id** (int, optional) – Required if inline_message_id is not specified. Identifier
  of the sent message.
- **inline_message_id** (str, optional) – Required if chat_id and message_id are not specified.
  Identifier of the inline message.
- **reply_markup** (telegram.ReplyMarkup, optional) – Additional interface options. A
  JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove
  reply keyboard or to force a reply from the user.
- **timeout** (int | float, optional) – If this value is specified, use it as the read
  timeout from the server (instead of the one specified during creation of the connection
  pool).
- ****kwars (dict) – Arbitrary keyword arguments.

Returns

On success, if edited message is sent by the bot, the editedMessage is returned, otherwise True is returned.

Return type: telegram.Message

Raises: telegram.TelegramError

edit_message_text (**kwargs)

Use this method to edit text and game messages sent by the bot or via the bot (for inline bots).

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target
  channel (in the format @channelusername).
- `message_id` (int, optional) – Required if inline_message_id is not specified. Identifier of the sent message.
- `inline_message_id` (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.
- `text` (str) – New text of the message.
- `parse_mode` (str) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot’s message. See the constants in `telegram.ParseMode` for the available modes.
- `disable_web_page_preview` (bool, optional) – Disables link previews for links in this message.
- `reply_markup` (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **kwargs (dict) – Arbitrary keyword arguments.

Returns
On success, if edited message is sent by the bot, the edited Message is returned, otherwise True is returned.

Return type `telegram.Message`

Raises `telegram.TelegramError`

`exportChatInviteLink` (chat_id, timeout=None, **kwargs)
Alias for `export_chat_invite_link`

`export_chat_invite_link` (chat_id, timeout=None, **kwargs)
Use this method to export an invite link to a supergroup or a channel. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Parameters
- `chat_id` (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **kwargs (dict) – Arbitrary keyword arguments

Returns
Exported invite link on success.

Return type `str`

Raises `telegram.TelegramError`

`first_name` str – Bot’s first name.

`forwardMessage` (chat_id, from_chat_id, message_id, disable_notification=False, timeout=None, **kwargs)
Alias for `forward_message`

`forward_message` (chat_id, from_chat_id, message_id, disable_notification=False, timeout=None, **kwargs)
Use this method to forward messages of any kind.

Parameters
• **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• **from_chat_id** (int | str) – Unique identifier for the chat where the original message was sent (or channel username in the format @channelusername).

• **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.

• **message_id** (int) – Message identifier in the chat specified in from_chat_id.

• **timeout** (int | float, optional) – If this value is specified, use it as read timeout (the) – from the server (instead of the one specified during creation of the connection pool).

• **kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type **telegram.Message**

Raises **telegram.TelegramError**

getChat (chat_id, timeout=None, **kwargs)

Alias for **get_chat**

getChatAdministrators (chat_id, timeout=None, **kwargs)

Alias for **get_chat_administrators**

getChatMember (chat_id, user_id, timeout=None, **kwargs)

Alias for **get_chat_member**

getChatMembersCount (chat_id, timeout=None, **kwargs)

Alias for **get_chat_members_count**

getFile (file_id, timeout=None, **kwargs)

Alias for **get_file**

getGameHighScores (user_id, chat_id=None, message_id=None, inline_message_id=None, timeout=None, **kwargs)

Alias for **get_game_high_scores**

getMe (timeout=None, **kwargs)

Alias for **get_me**

getStickerSet (name, timeout=None, **kwargs)

Alias for **get_sticker_set**

getUpdates (offset=None, limit=100, timeout=0, read_latency=2.0, allowed_updates=None, **kwargs)

Alias for **get_updates**

getUserProfilePhotos (user_id, offset=None, limit=100, timeout=None, **kwargs)

Alias for **get_user_profile_photos**

getWebhookInfo (timeout=None, **kwargs)

Alias for **get_webhook_info**

get_chat (chat_id, timeout=None, **kwargs)

Use this method to get up to date information about the chat (current name of the user for one-on-one conversations, current username of a user, group or channel, etc.).

Parameters

• **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
• **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **kwargs** (dict) – Arbitrary keyword arguments.

Returns `telegram.Chat`

Raises `telegram.TelegramError`

**get_chat_administrators** *(chat_id, timeout=None, **kwargs)*

Use this method to get a list of administrators in a chat. On success, returns an Array of ChatMember objects that contains information about all chat administrators except other bots. If the chat is a group or a supergroup and no administrators were appointed, only the creator will be returned.

Parameters

• **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **kwargs** (dict) – Arbitrary keyword arguments.

Returns `List[telegram.ChatMember]`

Raises `telegram.TelegramError`

**get_chat_member** *(chat_id, user_id, timeout=None, **kwargs)*

Use this method to get information about a member of a chat.

Parameters

• **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• **user_id** (int) – Unique identifier of the target user.

• **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **kwargs** (dict) – Arbitrary keyword arguments.

Returns `telegram.ChatMember`

Raises `telegram.TelegramError`

**get_chat_members_count** *(chat_id, timeout=None, **kwargs)*

Use this method to get the number of members in a chat

Parameters

• **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **kwargs** (dict) – Arbitrary keyword arguments.

Returns `Number of members in the chat`

Return type `int`

Raises `telegram.TelegramError`
**get_file** *(file_id, timeout=None, **kwargs)*

Use this method to get basic info about a file and prepare it for downloading. For the moment, bots can download files of up to 20MB in size. The file can then be downloaded with `telegram.File.download`. It is guaranteed that the link will be valid for at least 1 hour. When the link expires, a new one can be requested by calling `get_file` again.

**Parameters**

- `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- `**kwargs` (dict) – Arbitrary keyword arguments.

**Returns** `telegram.File`

**Raises** `telegram.TelegramError`

**get_game_high_scores** *(user_id, chat_id=None, message_id=None, inline_message_id=None, timeout=None, **kwargs)*

Use this method to get data for high score tables. Will return the score of the specified user and several of his neighbors in a game.

**Parameters**

- `user_id` (int) – User identifier.
- `chat_id` (int | str, optional) – Required if `inline_message_id` is not specified. Unique identifier for the target chat.
- `message_id` (int, optional) – Required if `inline_message_id` is not specified. Identifier of the sent message.
- `inline_message_id` (str, optional) – Required if `chat_id` and `message_id` are not specified. Identifier of the inline message.
- `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- `**kwargs` (dict) – Arbitrary keyword arguments.

**Returns** List[`telegram.GameHighScore`]

**Raises** `telegram.TelegramError`

**get_me** *(timeout=None, **kwargs)*

A simple method for testing your bot’s auth token. Requires no parameters.

**Parameters** `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

**Returns** A `telegram.User` instance representing that bot if the credentials are valid, None otherwise.

**Return type** `telegram.User`

**Raises** `telegram.TelegramError`

**get_sticker_set** *(name, timeout=None, **kwargs)*

Use this method to get a sticker set.

**Parameters**
• **name** *(str)* – Short name of the sticker set that is used in t.me/addstickers/ URLs (e.g., animals)

• **timeout** *(int | float, optional)* – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **kwargs** *(dict)* – Arbitrary keyword arguments.

Returns : `telegram.StickerSet`

Raises : `telegram.TelegramError`

`get_updates` *(offset=None, limit=100, timeout=0, read_latency=2.0, allowed_updates=None, **kwargs)*

Use this method to receive incoming updates using long polling.

Parameters

• **offset** *(int, optional)* – Identifier of the first update to be returned. Must be greater by one than the highest among the identifiers of previously received updates. By default, updates starting with the earliest unconfirmed update are returned. An update is considered confirmed as soon as getUpdates is called with an offset higher than its update_id. The negative offset can be specified to retrieve updates starting from -offset update from the end of the updates queue. All previous updates will forgotten.

• **limit** *(int, optional)* – Limits the number of updates to be retrieved. Values between 1-100 are accepted. Defaults to 100.

• **timeout** *(int, optional)* – Timeout in seconds for long polling. Defaults to 0, i.e. usual short polling. Should be positive, short polling should be used for testing purposes only.

• **allowed_updates** *(List[str], optional)* – List the types of updates you want your bot to receive. For example, specify ["message", "edited_channel_post", "callback_query"] to only receive updates of these types. See `telegram.Update` for a complete list of available update types. Specify an empty list to receive all updates regardless of type (default). If not specified, the previous setting will be used. Please note that this parameter doesn’t affect updates created before the call to the get_updates, so unwanted updates may be received for a short period of time.

• **kwargs** *(dict)* – Arbitrary keyword arguments.

Notes

1. This method will not work if an outgoing webhook is set up.
2. In order to avoid getting duplicate updates, recalculate offset after each server response.
3. To take full advantage of this library take a look at `telegram.ext.Updater`

Returns : `List[telegram.Update]`

Raises : `telegram.TelegramError`

`get_user_profile_photos` *(user_id, offset=None, limit=100, timeout=None, **kwargs)*

Use this method to get a list of profile pictures for a user.

Parameters

• **user_id** *(int)* – Unique identifier of the target user.

• **offset** *(int, optional)* – Sequential number of the first photo to be returned. By default, all photos are returned.

• **limit** *(int, optional)* – Limits the number of photos to be retrieved. Values between 1-100 are accepted. Defaults to 100.
• **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **kwargs** (dict) – Arbitrary keyword arguments.

Returns `telegram.UserProfilePhotos`

Raises `telegram.TelegramError`

**get_webhook_info** (timeout=None, **kwargs)

Use this method to get current webhook status. Requires no parameters.

If the bot is using getUpdates, will return an object with the url field empty.

Parameters

• **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **kwargs** (dict) – Arbitrary keyword arguments.

Returns `telegram.WebhookInfo`

id

int – Unique identifier for this bot.

**kickChatMember** (chat_id, user_id, timeout=None, until_date=None, **kwargs)

Alias for `kick_chat_member`

**kick_chat_member** (chat_id, user_id, timeout=None, until_date=None, **kwargs)

Use this method to kick a user from a group or a supergroup. In the case of supergroups, the user will not be able to return to the group on their own using invite links, etc., unless unbanned first. The bot must be an administrator in the group for this to work.

Parameters

• **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• **user_id** (int) – Unique identifier of the target user.

• **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **until_date** (int | datetime.datetime, optional) – Date when the user will be unbanned, unix time. If user is banned for more than 366 days or less than 30 seconds from the current time they are considered to be banned forever.

• **kwargs** (dict) – Arbitrary keyword arguments.

Note: In regular groups (non-supergroups), this method will only work if the ‘All Members Are Admins’ setting is off in the target group. Otherwise members may only be removed by the group’s creator or by the member that added them.

Returns bool On success, True is returned.

Raises `telegram.TelegramError`

**last_name**

str – Optional. Bot’s last name.

**leaveChat** (chat_id, timeout=None, **kwargs)

Alias for `leave_chat`
**leave_chat** *(chat_id, timeout=None, **kwargs)*

Use this method for your bot to leave a group, supergroup or channel.

**Parameters**

- **chat_id** *(int | str)* – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **timeout** *(int | float, optional)* – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs**(dict) – Arbitrary keyword arguments.

**Returns** *bool* On success, True is returned.

**Raises** *telegram.TelegramError*

**name**

*str* – Bot’s @username.

**pinChatMessage** *(chat_id, message_id, disable_notification=None, timeout=None, **kwargs)*

Alias for **pin_chat_message**

**pin_chat_message** *(chat_id, message_id, disable_notification=None, timeout=None, **kwargs)*

Use this method to pin a message in a supergroup. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

**Parameters**

- **chat_id** *(int | str)* – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **message_id** *(int)* – Identifier of a message to pin.
- **disable_notification** *(bool, optional)* – Pass True, if it is not necessary to send a notification to all group members about the new pinned message.
- **timeout** *(int | float, optional)* – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs**(dict) – Arbitrary keyword arguments

**Returns**  

Returns True on success.

**Return type** *bool*

**Raises** *telegram.TelegramError*

**promoteChatMember** *(chat_id, user_id, can_change_info=None, can_post_messages=None, can_edit_messages=None, can_delete_messages=None, can_invite_users=None, can_restrict_members=None, can_pin_messages=None, can_promote_members=None, timeout=None, **kwargs)*

Alias for **promote_chat_member**

**promote_chat_member** *(chat_id, user_id, can_change_info=None, can_post_messages=None, can_edit_messages=None, can_delete_messages=None, can_invite_users=None, can_restrict_members=None, can_pin_messages=None, can_promote_members=None, timeout=None, **kwargs)*

Use this method to promote or demote a user in a supergroup or a channel. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights. Pass False for all boolean parameters to demote a user.

**Parameters**
• chat_id (int | str) – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername).

• user_id (int) – Unique identifier of the target user.

• can_change_info (bool, optional) – Pass True, if the administrator can change chat title, photo and other settings.

• can_post_messages (bool, optional) – Pass True, if the administrator can create channel posts, channels only.

• can_edit_messages (bool, optional) – Pass True, if the administrator can edit messages of other users, channels only.

• can_delete_messages (bool, optional) – Pass True, if the administrator can delete messages of other users.

• can_invite_users (bool, optional) – Pass True, if the administrator can invite new users to the chat.

• can_restrict_members (bool, optional) – Pass True, if the administrator can restrict, ban or unban chat members.

• can_pin_messages (bool, optional) – Pass True, if the administrator can pin messages, supergroups only.

• can_promote_members (bool, optional) – Pass True, if the administrator can add new administrators with a subset of his own privileges or demote administrators that he has promoted, directly or indirectly (promoted by administrators that were appointed by him).

• timeout (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **kwargs (dict) – Arbitrary keyword arguments

Returns Returns True on success.

Return type bool

Raises telegram.TelegramError

request

restrictChatMember (chat_id, user_id, until_date=None, can_send_messages=None, can_send_media_messages=None, can_send_other_messages=None, can_add_web_page_previews=None, timeout=None, **kwargs)

Alias for restrict_chat_member

restrict_chat_member (chat_id, user_id, until_date=None, can_send_messages=None, can_send_media_messages=None, can_send_other_messages=None, can_add_web_page_previews=None, timeout=None, **kwargs)

Use this method to restrict a user in a supergroup. The bot must be an administrator in the supergroup for this to work and must have the appropriate admin rights. Pass True for all boolean parameters to lift restrictions from a user.

Parameters

• chat_id (int | str) – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername).

• user_id (int) – Unique identifier of the target user.

• until_date (int | datetime.datetime, optional) – Date when restrictions will be lifted for the user, unix time. If user is restricted for more than 366 days or less than 30 seconds from the current time, they are considered to be restricted forever.
• `can_send_messages` (bool, optional) – Pass True, if the user can send text messages, contacts, locations and venues.

• `can_send_media_messages` (bool, optional) – Pass True, if the user can send audios, documents, photos, videos, video notes and voice notes, implies `can_send_messages`.

• `can_send_other_messages` (bool, optional) – Pass True, if the user can send animations, games, stickers and use inline bots, implies `can_send_media_messages`.

• `can_add_web_page_previews` (bool, optional) – Pass True, if the user may add web page previews to their messages, implies `can_send_media_messages`.

• `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• `**kwargs` (dict) – Arbitrary keyword arguments

**Returns** Returns True on success.

**Return type** bool

**Raises** `telegram.TelegramError`

`sendAnimation` (chat_id, animation, duration=None, width=None, height=None, thumb=None, caption=None, parse_mode=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, **kwargs)

Alias for `send_animation`

`sendAudio` (chat_id, audio, duration=None, performer=None, title=None, caption=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, thumb=None, **kwargs)

Alias for `send_audio`

`sendChatAction` (chat_id, action, timeout=None, **kwargs)

Alias for `send_chat_action`

`sendContact` (chat_id, phone_number=None, first_name=None, last_name=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=None, contact=None, vcard=None, **kwargs)

Alias for `send_contact`

`sendDocument` (chat_id, document, filename=None, caption=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, thumb=None, **kwargs)

Alias for `send_document`

`sendGame` (chat_id, game_short_name, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=None, **kwargs)

Alias for `send_game`

`sendInvoice` (chat_id, title, description, payload, provider_token, start_parameter, currency, prices, photo_url=None, photo_size=None, photo_width=None, photo_height=None, need_name=None, need_phone_number=None, need_email=None, need_shipping_address=None, is_flexible=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, provider_data=None, send_phone_number_to_provider=None, send_email_to_provider=None, timeout=None, **kwargs)

Alias for `send_invoice`

`sendLocation` (chat_id, latitude=None, longitude=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=None, location=None, live_period=None, **kwargs)

Alias for `send_location`
sendMediaGroup(chat_id, media, disable_notification=None, reply_to_message_id=None, timeout=20, **kwargs)

Alias for send_media_group

sendMessage(chat_id, text, parse_mode=None, disable_web_page_preview=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=None, **kwargs)

Alias for sendMessage

sendPhoto(chat_id, photo, caption=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, **kwargs)

Alias for send_photo

sendSticker(chat_id, sticker, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, **kwargs)

Alias for send_sticker

sendVenue(chat_id, latitude=None, longitude=None, title=None, address=None, foursquare_id=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=None, venue=None, foursquare_type=None, **kwargs)

Alias for send_venue

sendVideo(chat_id, video, duration=None, caption=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, width=None, height=None, parse_mode=None, supports_streaming=None, thumb=None, **kwargs)

Alias for send_video

sendVideoNote(chat_id, video_note, duration=None, length=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, thumb=None, **kwargs)

Alias for send_video_note

sendVoice(chat_id, voice, duration=None, caption=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, **kwargs)

Alias for send_voice

send_animation(chat_id, animation, duration=None, width=None, height=None, thumb=None, caption=None, parse_mode=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, **kwargs)

Use this method to send animation files (GIF or H.264/MPEG-4 AVC video without sound).

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **animation** (str | filelike object | telegram.Animation) – Animation to send. Pass a file_id as String to send an animation that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an animation from the Internet, or upload a new animation using multipart/form-data. Lastly you can pass an existing telegram.Animation object to send.
- **duration** (int, optional) – Duration of sent animation in seconds.
- **width** (int, optional) – Animation width.
- **height** (int, optional) – Animation height.
- **thumb** (filelike object, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.
- **caption** (str, optional) – Animation caption (may also be used when resending animations by file_id), 0-1024 characters.
**parse_mode** *(str, optional)* – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.

**disable_notification** *(bool, optional)* – Sends the message silently. Users will receive a notification with no sound.

**reply_to_message_id** *(int, optional)* – If the message is a reply, ID of the original message.

**reply_markup** *(telegram.ReplyMarkup, optional)* – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

**timeout** *(int|float, optional)* – Send file timeout (default: 20 seconds).

****kwargs** *(dict)* – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type *telegram.Message*

Raises *telegram.TelegramError*

---

**send_audio** *(chat_id, audio, duration=None, performer=None, title=None, caption=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, thumb=None, **kwargs)*

Use this method to send audio files, if you want Telegram clients to display them in the music player. Your audio must be in the .mp3 format. On success, the sent Message is returned. Bots can currently send audio files of up to 50 MB in size, this limit may be changed in the future.

For sending voice messages, use the sendVoice method instead.

---

**Note:** The audio argument can be either a file_id, an URL or a file from disk `open(filename, 'rb')`

---

**Parameters**

- **chat_id** *(int|str)* – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

- **audio** *(str|filelike object|telegram.Audio)* – Audio file to send. Pass a file_id as String to send an audio file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an audio file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing *telegram.Audio* object to send.

- **caption** *(str, optional)* – Audio caption, 0-1024 characters.

- **parse_mode** *(str, optional)* – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.

- **duration** *(int, optional)* – Duration of sent audio in seconds.

- **performer** *(str, optional)* – Performer.

- **title** *(str, optional)* – Track name.

- **disable_notification** *(bool, optional)* – Sends the message silently. Users will receive a notification with no sound.

- **reply_to_message_id** *(int, optional)* – If the message is a reply, ID of the original message.
• **reply_markup**(*telegram.ReplyMarkup*, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

• **thumb**(*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

• **timeout**(*int|float*, optional) – Send file timeout (default: 20 seconds).

• **kwargs**(*dict*) – Arbitrary keyword arguments.

**Returns** On success, the sent Message is returned.

**Return type** *telegram.Message*

**Raises** *telegram.TelegramError*

**send_chat_action**(*chat_id*, *action*, *timeout=None*, **kwargs)

Use this method when you need to tell the user that something is happening on the bot’s side. The status is set for 5 seconds or less (when a message arrives from your bot, Telegram clients clear its typing status).

**Parameters**

• **chat_id**(*int|str*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• **action**(*telegram.ChatAction|str*) – Type of action to broadcast. Choose one, depending on what the user is about to receive. For convenience look at the constants in *telegram.ChatAction*

• **timeout**(*int|float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **kwargs**(*dict*) – Arbitrary keyword arguments.

**Returns** *True* on success.

**Return type** *bool*

**Raises** *telegram.TelegramError*

**send_contact**(*chat_id*, *phone_number=None*, *first_name=None*, *last_name=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*, *timeout=None*, *contact=None*, *vcard=None*, **kwargs)

Use this method to send phone contacts.

**Parameters**

• **chat_id**(*int|str*) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• **phone_number**(*str*, optional) – Contact’s phone number.

• **first_name**(*str*, optional) – Contact’s first name.

• **last_name**(*str*, optional) – Contact’s last name.

• **vcard**(*str*, optional) – Additional data about the contact in the form of a vCard, 0-2048 bytes.

• **contact**(*telegram.Contact*, optional) – The contact to send.

**Note:** You can either supply *contact* or *phone_number* and *first_name* with optionally *last_name* and optionally *vcard.*
• **disable_notification** *(bool, optional)* – Sends the message silently. Users will receive a notification with no sound.

• **reply_to_message_id** *(int, optional)* – If the message is a reply, ID of the original message.

• **reply_markup** *(telegram.ReplyMarkup, optional)* – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

• **timeout** *(int | float, optional)* – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• ****kwargs**(dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type telegram.Message

Raises telegram.TelegramError

**send_document** *(chat_id, document, filename=None, caption=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, thumb=None, **kwargs)*

Use this method to send general files.

Note: The document argument can be either a file_id, an URL or a file from disk `open(filename, 'rb')`

Parameters

• **chat_id**(int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• **document**(str | filelike object | telegram.Document) – File to send. Pass a file_id as String to send a file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing telegram.Document object to send.

• **filename**(str, optional) – File name that shows in telegram message (it is useful when you send file generated by temp module, for example). Undocumented.

• **caption**(str, optional) – Document caption (may also be used when resend documents by file_id), 0-1024 characters.

• **parse_mode**(str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

• **disable_notification**(bool, optional) – Sends the message silently. Users will receive a notification with no sound.

• **reply_to_message_id**(int, optional) – If the message is a reply, ID of the original message.

• **reply_markup**(telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

• **thumb**(filelike object, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

• **timeout**(int | float, optional) – Send file timeout (default: 20 seconds).
**kwargs (dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type telegram.Message

Raises telegram.TelegramError

**send_game**(chat_id, game_short_name, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=None, **kwargs)

Use this method to send a game.

Parameters

• chat_id (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• game_short_name (str) – Short name of the game, serves as the unique identifier for the game. Set up your games via Botfather.

• disable_notification (bool, optional) – Sends the message silently. Users will receive a notification with no sound.

• reply_to_message_id (int, optional) – If the message is a reply, ID of the original message.

• reply_markup (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

• timeout (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **kwargs (dict) – Arbitrary keyword arguments.

**send_invoice**(chat_id, title, description, payload, provider_token, start_parameter, currency, prices, photo_url=None, photo_size=None, photo_width=None, photo_height=None, need_name=None, need_phone_number=None, need_email=None, need_shipping_address=None, is_flexible=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, provider_data=None, send_phone_number_to_provider=None, send_email_to_provider=None, timeout=None, **kwargs)

Use this method to send invoices.

Parameters

• chat_id (int | str) – Unique identifier for the target private chat.

• title (str) – Product name.

• description (str) – Product description.

• payload (str) – Bot-defined invoice payload, 1-128 bytes. This will not be displayed to the user, use for your internal processes.

• provider_token (str) – Payments provider token, obtained via Botfather.

• start_parameter (str) – Unique deep-linking parameter that can be used to generate this invoice when used as a start parameter.

• currency (str) – Three-letter ISO 4217 currency code.

• prices (List[telegram.LabeledPrice]) – Price breakdown, a list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.).
• `provider_data` *(str | object, optional)* – JSON-encoded data about the invoice, which will be shared with the payment provider. A detailed description of required fields should be provided by the payment provider. When an object is passed, it will be encoded as JSON.

• `photo_url` *(str, optional)* – URL of the product photo for the invoice. Can be a photo of the goods or a marketing image for a service. People like it better when they see what they are paying for.

• `photo_size` *(str, optional)* – Photo size.

• `photo_width` *(int, optional)* – Photo width.

• `photo_height` *(int, optional)* – Photo height.

• `need_name` *(bool, optional)* – Pass True, if you require the user’s full name to complete the order.

• `need_phone_number` *(bool, optional)* – Pass True, if you require the user’s phone number to complete the order.

• `need_email` *(bool, optional)* – Pass True, if you require the user’s email to complete the order.

• `need_shipping_address` *(bool, optional)* – Pass True, if you require the user’s shipping address to complete the order.

• `send_phone_number_to_provider` *(bool, optional)* – Pass True, if user’s phone number should be sent to provider.

• `send_email_to_provider` *(bool, optional)* – Pass True, if user’s email address should be sent to provider.

• `is_flexible` *(bool, optional)* – Pass True, if the final price depends on the shipping method.

• `disable_notification` *(bool, optional)* – Sends the message silently. Users will receive a notification with no sound.

• `reply_to_message_id` *(int, optional)* – If the message is a reply, ID of the original message.

• `reply_markup` *(telegram.ReplyMarkup, optional)* – Additional interface options. An inlinekeyboard. If empty, one ‘Pay total price’ button will be shown. If not empty, the first button must be a Pay button.

• `timeout` *(int | float, optional)* – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **`**kwargs**` *(dict)* – Arbitrary keyword arguments.

Returns
On success, the sent Message is returned.

Return type
`telegram.Message`

Raises
`telegram.TelegramError`

**send_location** *(chat_id, latitude=None, longitude=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=None, location=None, live_period=None, **kwargs)*

Use this method to send point on the map.

**Note:** You can either supply a latitude and longitude or a location.

Parameters
• **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• **latitude** (float, optional) – Latitude of location.

• **longitude** (float, optional) – Longitude of location.

• **location** (telegram.Location, optional) – The location to send.

• **live_period** (int, optional) – Period in seconds for which the location will be updated, should be between 60 and 86400.

• **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.

• **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.

• **reply_markup** (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

• **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type telegram.Message

Raises telegram.TelegramError

send_media_group (chat_id, media, disable_notification=None, reply_to_message_id=None, timeout=20, **kwargs)

Use this method to send a group of photos or videos as an album.

Parameters

• **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• **media** (List[telegram.InputMedia]) – An array describing photos and videos to be sent, must include 2–10 items.

• **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.

• **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.

• **timeout** (int | float, optional) – Send file timeout (default: 20 seconds).

• ****kwargs** (dict) – Arbitrary keyword arguments.

Returns An array of the sent Messages.

Return type List[telegram.Message]

Raises telegram.TelegramError

send_message (chat_id, text, parse_mode=None, disable_web_page_preview=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=None, **kwargs)

Use this method to send text messages.

Parameters

• **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
• **text** *(str)* – Text of the message to be sent. Max 4096 characters. Also found as `telegram.constants.MAX_MESSAGE_LENGTH`.

• **parse_mode** *(str)* – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot’s message. See the constants in `telegram.ParseMode` for the available modes.

• **disable_web_page_preview** *(bool, optional)* – Disables link previews for links in this message.

• **disable_notification** *(bool, optional)* – Sends the message silently. Users will receive a notification with no sound.

• **reply_to_message_id** *(int, optional)* – If the message is a reply, ID of the original message.

• **reply_markup** *(telegram.ReplyMarkup, optional)* – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

• **timeout** *(int | float, optional)* – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **kwargs** *(dict)* – Arbitrary keyword arguments.

**Returns** On success, the sent message is returned.

**Return type** `telegram.Message`

**Raises** `telegram.TelegramError`

**send_photo** *(chat_id, photo, caption=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, **kwargs)*

Use this method to send photos.

**Note:** The photo argument can be either a file_id, an URL or a file from disk `open(filename, 'rb')`

**Parameters**

• **chat_id** *(int | str)* – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• **photo** *(str | filelike object | telegram.PhotoSize)* – Photo to send. Pass a file_id as a String to send a photo that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a photo from the Internet, or upload a new photo using multipart/form-data. Lastly you can pass an existing `telegram.PhotoSize` object to send.

• **caption** *(str, optional)* – Photo caption (may also be used when resending photos by file_id), 0-1024 characters.

• **parse_mode** *(str, optional)* – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

• **disable_notification** *(bool, optional)* – Sends the message silently. Users will receive a notification with no sound.

• **reply_to_message_id** *(int, optional)* – If the message is a reply, ID of the original message.
send_sticker(chat_id, sticker, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, **kwargs)

Use this method to send .webp stickers.

Parameters

• chat_id(int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• sticker(str | filelike object telegram.Sticker) – Sticker to send. Pass a file_id as String to send a file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a .webp file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing telegram.Sticker object to send.

• disable_notification(bool, optional) – Sends the message silently. Users will receive a notification with no sound.

• reply_to_message_id(int, optional) – If the message is a reply, ID of the original message.

• reply_markup(telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

• timeout(int | float, optional) – Send file timeout (default: 20 seconds).

• **kwargs(dict) – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type telegram.Message

Raises telegram.TelegramError

send_venue(chat_id, latitude=None, longitude=None, title=None, address=None, foursquare_id=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=None, venue=None, foursquare_type=None, **kwargs)

Use this method to send information about a venue.

Note: you can either supply venue, or latitude, longitude, title and address and optionally foursquare_id and optionally foursquare_type.
• `chat_id` (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
• `latitude` (float, optional) – Latitude of venue.
• `longitude` (float, optional) – Longitude of venue.
• `title` (str, optional) – Name of the venue.
• `address` (str, optional) – Address of the venue.
• `foursquare_id` (str, optional) – Foursquare identifier of the venue.
• `foursquare_type` (str, optional) – Foursquare type of the venue, if known. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”.)
• `venue` (telegram.Venue, optional) – The venue to send.
• `disable_notification` (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
• `reply_to_message_id` (int, optional) – If the message is a reply, ID of the original message.
• `reply_markup` (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
• `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
• **kwargs (dict) – Arbitrary keyword arguments.

Returns
On success, the sent Message is returned.

Return type telegram.Message

Raises telegrams.TelegramError

send_video

Use this method to send video files, Telegram clients support mp4 videos (other formats may be sent as Document).

Parameters

• `chat_id` (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• `video` (str | filelike object | telegram.Vide0) – Video file to send. Pass a file_id as String to send an video file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an video file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing telegram.Vide0 object to send.

• `duration` (int, optional) – Duration of sent video in seconds.

• `width` (int, optional) – Video width.

• `height` (int, optional) – Video height.
• caption (str, optional) – Video caption (may also be used when resend videos by file_id), 0-1024 characters.

• parse_mode (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

• supports_streaming (bool, optional) – Pass True, if the uploaded video is suitable for streaming.

• disable_notification (bool, optional) – Sends the message silently. Users will receive a notification with no sound.

• reply_to_message_id (int, optional) – If the message is a reply, ID of the original message.

• reply_markup (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

• thumb (filelike object, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

• timeout (int|float, optional) – Send file timeout (default: 20 seconds).

• **kwargs (dict) – Arbitrary keyword arguments.

Returns
On success, the sent Message is returned.

Return type  telegram.Message

Raises  telegram.TelegramError

send_video_note (chat_id, video_note, duration=None, length=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, thumb=None, **kwargs)

Use this method to send video messages.

Note:  The video_note argument can be either a file_id or a file from disk open(filename, 'rb')

Parameters

• chat_id (int|str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• video_note (str | filelike object | telegram.VideoNote) – Video note to send. Pass a file_id as String to send a video note that exists on the Telegram servers (recommended) or upload a new video using multipart/form-data. Or you can pass an existing telegram.VideoNote object to send. Sending video notes by a URL is currently unsupported.

• duration (int, optional) – Duration of sent video in seconds.

• length (int, optional) – Video width and height

• disable_notification (bool, optional) – Sends the message silently. Users will receive a notification with no sound.

• reply_to_message_id (int, optional) – If the message is a reply, ID of the original message.

• reply_markup (telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
• **thumb** *(filelike object, optional)* – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not passed as a string or file_id.

• **timeout** *(int|float, optional)* – Send file timeout (default: 20 seconds).

• **kwargs** *(dict)* – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type *telegram.Message*

Raises *telegram.TelegramError*

### send_voice

```
@bot.send_voice(chat_id, voice, duration=None, caption=None, disable_notification=False, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, **kwargs)
```

Use this method to send audio files, if you want Telegram clients to display the file as a playable voice message. For this to work, your audio must be in an .ogg file encoded with OPUS (other formats may be sent as Audio or Document).

**Note:** The voice argument can be either a file_id, an URL or a file from disk `open(filename, 'rb')`

#### Parameters

• **chat_id** *(int|str)* – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• **voice** *(str|filelike object|telegram.Voice)* – Voice file to send. Pass a file_id as String to send an voice file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an voice file from the Internet, or upload a new one using multipart/form-data. Lastly you can pass an existing `telegram.Voice` object to send.

• **caption** *(str, optional)* – Voice message caption, 0-1024 characters.

• **parse_mode** *(str, optional)* – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

• **duration** *(int, optional)* – Duration of the voice message in seconds.

• **disable_notification** *(bool, optional)* – Sends the message silently. Users will receive a notification with no sound.

• **reply_to_message_id** *(int, optional)* – If the message is a reply, ID of the original message.

• **reply_markup** *(telegram.ReplyMarkup, optional)* – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

• **timeout** *(int|float, optional)* – Send file timeout (default: 20 seconds).

• **kwargs** *(dict)* – Arbitrary keyword arguments.

Returns On success, the sent Message is returned.

Return type *telegram.Message*

Raises *telegram.TelegramError*

### setChatDescription

```
@bot.set_chat_description(chat_id, description, timeout=None, **kwargs)
```

Alias for `set_chat_description`
**setChatPhoto** *(chat_id, photo, timeout=20, **kwargs)*

Alias for **set_chat_photo**

**setChatStickerSet** *(chat_id, sticker_set_name, timeout=None, **kwargs)*

Alias for **set_chat_sticker_set**

**setChatTitle** *(chat_id, title, timeout=None, **kwargs)*

Alias for **set_chat_title**

**setGameScore** *(user_id, score, chat_id=None, message_id=None, inline_message_id=None, force=None, disable_edit_message=None, timeout=None, **kwargs)*

Alias for **set_game_score**

**setPassportDataErrors** *(user_id, errors, timeout=None, **kwargs)*

Alias for **set_passport_data_errors**

**setStickerPositionInSet** *(sticker, position, timeout=None, **kwargs)*

Alias for **set_sticker_position_in_set**

**setWebhook** *(url=None, certificate=None, timeout=None, max_connections=40, allowed_updates=None, **kwargs)*

Alias for **set_webhook**

**set_chat_description** *(chat_id, description, timeout=None, **kwargs)*

Use this method to change the description of a supergroup or a channel. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

**Parameters**

- **chat_id** *(int | str)* – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **description** *(str)* – New chat description, 1-255 characters.
- **timeout** *(int | float, optional)* – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs**(dict) – Arbitrary keyword arguments

**Returns**

Returns True on success.

**Return type**

bool

**Raises**

telegram.TelegramError

**set_chat_photo** *(chat_id, photo, timeout=20, **kwargs)*

Use this method to set a new profile photo for the chat.

Photos can’t be changed for private chats. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

**Parameters**

- **chat_id** *(int | str)* – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **photo** *(filelike object)* – New chat photo.
- **timeout** *(int | float, optional)* – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs**(dict) – Arbitrary keyword arguments

**Note:** In regular groups (non-supergroups), this method will only work if the ‘All Members Are Admins’ setting is off in the target group.
Returns True on success.

Return type bool

Raises telegram.TelegramError

set_chat_sticker_set (chat_id, sticker_set_name, timeout=None, **kwargs)

Use this method to set a new group sticker set for a supergroup. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights. Use the field telegram.Chat.can_set_sticker_set optionally returned in get_chat requests to check if the bot can use this method.

Parameters

• chat_id (int | str) – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername).

• sticker_set_name (str) – Name of the sticker set to be set as the group sticker set.

• timeout (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **kwargs (dict) – Arbitrary keyword arguments.

Returns True on success.

Return type bool

set_chat_title (chat_id, title, timeout=None, **kwargs)

Use this method to change the title of a chat. Titles can’t be changed for private chats. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Parameters

• chat_id (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• title (str) – New chat title, 1-255 characters.

• timeout (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **kwargs (dict) – Arbitrary keyword arguments

Note: In regular groups (non-supergroups), this method will only work if the ‘All Members Are Admins’ setting is off in the target group.

Returns True on success.

Return type bool

Raises telegram.TelegramError

set_game_score (user_id, score, chat_id=None, message_id=None, inline_message_id=None, force=None, disable_edit_message=None, timeout=None, **kwargs)

Use this method to set the score of the specified user in a game. On success, if the message was sent by the bot, returns the edited Message, otherwise returns True. Returns an error, if the new score is not greater than the user’s current score in the chat and force is False.

Parameters

• user_id (int) – User identifier.

• score (int) – New score, must be non-negative.
• **force** (bool, optional) – Pass True, if the high score is allowed to decrease. This can be useful when fixing mistakes or banning cheaters

• **disable_edit_message** (bool, optional) – Pass True, if the game message should not be automatically edited to include the current scoreboard.

• **chat_id** (int|str, optional) – Required if inline_message_id is not specified. Unique identifier for the target chat.

• **message_id** (int, optional) – Required if inline_message_id is not specified. Identifier of the sent message.

• **inline_message_id** (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.

• **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• ****kwargs** (dict) – Arbitrary keyword arguments.

Returns The edited message, or if the message wasn’t sent by the bot, True.

Return type  *telegram.Message*

Raises

• telegram.TelegramError – If the new score is not greater than the user’s current score in the chat and force is False.

**set_passport_data_errors** (*user_id*, *errors*, *timeout=None*, **kwargs)

Informs a user that some of the Telegram Passport elements they provided contains errors. The user will not be able to re-submit their Passport to you until the errors are fixed (the contents of the field for which you returned the error must change). Returns True on success.

Use this if the data submitted by the user doesn’t satisfy the standards your service requires for any reason. For example, if a birthday date seems invalid, a submitted document is blurry, a scan shows evidence of tampering, etc. Supply some details in the error message to make sure the user knows how to correct the issues.

Parameters

• **user_id** (int) – User identifier

• **errors** (List[PassportElementError]) – A JSON-serialized array describing the errors.

• **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• ****kwargs** (dict) – Arbitrary keyword arguments.

Returns On success, True is returned.

Return type  *bool*

Raises  *telegram.TelegramError*

**set_sticker_position_in_set** (*sticker*, *position*, *timeout=None*, **kwargs)

Use this method to move a sticker in a set created by the bot to a specific position.

Parameters

• **sticker** (str) – File identifier of the sticker.

• **position** (int) – New sticker position in the set, zero-based.
• **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **kwargs** (dict) – Arbitrary keyword arguments.

**Returns** On success, True is returned.

**Return type** bool

**Raises** telegram.TelegramError

set_webhook

Use this method to specify a url and receive incoming updates via an outgoing webhook. Whenever there is an update for the bot, we will send an HTTPS POST request to the specified url, containing a JSON-serialized Update. In case of an unsuccessful request, we will give up after a reasonable amount of attempts.

If you’d like to make sure that the Webhook request comes from Telegram, we recommend using a secret path in the URL, e.g. https://www.example.com/<token>. Since nobody else knows your bot’s token, you can be pretty sure it’s us.

**Note:** The certificate argument should be a file from disk open(filename, 'rb').

**Parameters**

• **url** (str) – HTTPS url to send updates to. Use an empty string to remove webhook integration.

• **certificate** (filelike) – Upload your public key certificate so that the root certificate in use can be checked. See our self-signed guide for details. (https://goo.gl/rw7w6Y)

• **max_connections** (int, optional) – Maximum allowed number of simultaneous HTTPS connections to the webhook for update delivery, 1-100. Defaults to 40. Use lower values to limit the load on your bot’s server, and higher values to increase your bot’s throughput.

• **allowed_updates** (List[str], optional) – List the types of updates you want your bot to receive. For example, specify ["message", “edited_channel_post”, “callback_query"] to only receive updates of these types. See telegram.Update for a complete list of available update types. Specify an empty list to receive all updates regardless of type (default). If not specified, the previous setting will be used. Please note that this parameter doesn’t affect updates created before the call to the set_webhook, so unwanted updates may be received for a short period of time.

• **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **kwargs** (dict) – Arbitrary keyword arguments.

**Note:**

1. You will not be able to receive updates using get_updates for as long as an outgoing webhook is set up.

2. To use a self-signed certificate, you need to upload your public key certificate using certificate parameter. Please upload as InputFile, sending a String will not work.

3. Ports currently supported for Webhooks: 443, 80, 88, 8443.
Returns bool On success, True is returned.

Raises telegram.TelegramError

stopMessageLiveLocation (chat_id=None, message_id=None, inline_message_id=None, reply_markup=None, timeout=None, **kwargs)

Alias for stop_message_live_location

stop_message_live_location (chat_id=None, message_id=None, inline_message_id=None, reply_markup=None, timeout=None, **kwargs)

Use this method to stop updating a live location message sent by the bot or via the bot (for inline bots) before live_period expires.

Parameters

- **chat_id**(int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **message_id**(int, optional) – Required if inline_message_id is not specified. Identifier of the sent message.
- **inline_message_id**(str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.
- **reply_markup**(telegram.ReplyMarkup, optional) – Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.
- **timeout**(int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

Returns On success the edited message.

Return type telegram.Message
to_dict ()

unbanChatMember (chat_id, user_id, timeout=None, **kwargs)

Alias for unban_chat_member

unban_chat_member (chat_id, user_id, timeout=None, **kwargs)

Use this method to unban a previously kicked user in a supergroup.

The user will not return to the group automatically, but will be able to join via link, etc. The bot must be an administrator in the group for this to work.

Parameters

- **chat_id**(int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **user_id**(int) – Unique identifier of the target user.
- **timeout**(int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs**(dict) – Arbitrary keyword arguments.

Returns bool On success, True is returned.

Raises telegram.TelegramError

unpinChatMessage (chat_id, timeout=None, **kwargs)

Alias for unpin_chat_message
unpin_chat_message(chat_id, timeout=None, **kwargs)

Use this method to unpin a message in a supergroup. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

Parameters

- chat_id (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- timeout (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **kwargs (dict) – Arbitrary keyword arguments

Returns

Returns True on success.

Return type bool

Raises telegram.TelegramError

uploadStickerFile(user_id, png_sticker, timeout=20, **kwargs)

Alias for upload_sticker_file

upload_sticker_file(user_id, png_sticker, timeout=20, **kwargs)

Use this method to upload a .png file with a sticker for later use in create_new_sticker_set and add_sticker_to_set methods (can be used multiple times).

Note: The png_sticker argument can be either a file_id, an URL or a file from disk open(filename, 'rb')

Parameters

- user_id (int) – User identifier of sticker file owner.
- png_sticker (str | filelike object) – Png image with the sticker, must be up to 512 kilobytes in size, dimensions must not exceed 512px, and either width or height must be exactly 512px.
- timeout (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **kwargs (dict) – Arbitrary keyword arguments.

Returns

The uploaded File

Return type telegram.File

Raises telegram.TelegramError

username

str – Bot’s username.

class telegram.Chat(id, type, title=None, username=None, first_name=None, last_name=None, all_members_are_administrators=None, bot=None, photo=None, description=None, invite_link=None, pinned_message=None, sticker_set_name=None, can_set_sticker_set=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents a chat.

id

int – Unique identifier for this chat.

type

str – Type of chat.
title
str – Optional. Title, for supergroups, channels and group chats.

username
str – Optional. Username.

first_name
str – Optional. First name of the other party in a private chat.

last_name
str – Optional. Last name of the other party in a private chat.

all_members_are_administrators
bool – Optional.

photo

description
str – Optional. Description, for supergroups and channel chats.

invite_link
str – Optional. Chat invite link, for supergroups and channel chats.

pinned_message

sticker_set_name
str – Optional. For supergroups, name of Group sticker set.

can_set_sticker_set
bool – Optional. True, if the bot can change group the sticker set.

Parameters

- id (int) – Unique identifier for this chat. This number may be greater than 32 bits and some programming languages may have difficulty/silent defects in interpreting it. But it is smaller than 52 bits, so a signed 64 bit integer or double-precision float type are safe for storing this identifier.
- type (str) – Type of chat, can be either ‘private’, ‘group’, ‘supergroup’ or ‘channel’.
- title (str, optional) – Title, for supergroups, channels and group chats.
- username (str, optional) – Username, for private chats, supergroups and channels if available.
- first_name (str, optional) – First name of the other party in a private chat.
- last_name (str, optional) – Last name of the other party in a private chat.
- all_members_are_administrators (bool, optional) – True if a group has All Members Are Admins enabled.
- photo (telegram.ChatPhoto, optional) – Chat photo. Returned only in getChat.
- description (str, optional) – Description, for supergroups and channel chats. Returned only in get_chat.
- invite_link (str, optional) – Chat invite link, for supergroups and channel chats. Returned only in get_chat.
- pinned_message (telegram.Message, optional) – Pinned message, for supergroups. Returned only in get_chat.
- bot (telegram.Bot, optional) – The Bot to use for instance methods.
- sticker_set_name (str, optional) – For supergroups, name of Group sticker set. Returned only in get_chat.
• **can_set_sticker_set** *(bool, optional)* – True, if the bot can change the sticker set. Returned only in `get_chat`.
• **kwargs** *(dict)* – Arbitrary keyword arguments.

```python
CHANNEL = 'channel'
str - 'channel'

GROUP = 'group'
str - 'group'

PRIVATE = 'private'
str - 'private'

SUPERGROUP = 'supergroup'
str - 'supergroup'

classmethod de_json(*data, bot)*

get_administrators(*args, **kwargs)*
Shortcut for:
```
bot.get_chat_administrators(update.message.chat.id, *args, **kwargs)
```

Returns A list of administrators in a chat. An Array of `telegram.ChatMember` objects that contains information about all chat administrators except other bots. If the chat is a group or a supergroup and no administrators were appointed, only the creator will be returned

Return type List[`telegram.ChatMember`]

get_member(*args, **kwargs)*
Shortcut for:
```
bot.get_chat_member(update.message.chat.id, *args, **kwargs)
```

Returns `telegram.ChatMember`

get_members_count(*args, **kwargs)*
Shortcut for:
```
bot.get_chat_members_count(update.message.chat.id, *args, **kwargs)
```

Returns `int`

kick_member(*args, **kwargs)*
Shortcut for:
```
bot.kick_chat_member(update.message.chat.id, *args, **kwargs)
```

Returns If the action was sent successfully.

Return type `bool`

Note: This method will only work if the All Members Are Admins setting is off in the target group. Otherwise members may only be removed by the group’s creator or by the member that added them.

leave(*args, **kwargs)*
Shortcut for:
```
bot.leave_chat(update.message.chat.id, *args, **kwargs)
```
Returns `bool` If the action was sent successfully.

**link**

 Casting propensity. If the chat has a `username`, returns a t.me link of the chat.

**send_action** (*args, **kwargs)

Shortcut for:

```
bot.send_chat_action(update.message.chat.id, *args, **kwargs)
```

**Returns** If the action was sent successfully.

**Return type** `bool`

**send_animation** (*args, **kwargs)

Shortcut for:

```
bot.send_animation(Chat.id, *args, **kwargs)
```

Where `Chat` is the current instance.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

**send_audio** (*args, **kwargs)

Shortcut for:

```
bot.send_audio(Chat.id, *args, **kwargs)
```

Where `Chat` is the current instance.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

**send_document** (*args, **kwargs)

Shortcut for:

```
bot.send_document(Chat.id, *args, **kwargs)
```

Where `Chat` is the current instance.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

**send_message** (*args, **kwargs)

Shortcut for:

```
bot.send_message(Chat.id, *args, **kwargs)
```

Where `Chat` is the current instance.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

**send_photo** (*args, **kwargs)

Shortcut for:

```
bot.send_photo(Chat.id, *args, **kwargs)
```

Where `Chat` is the current instance.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`
send_sticker(*args, **kwargs)
Shortcut for:

```
bot.send_sticker(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

send_video(*args, **kwargs)
Shortcut for:

```
bot.send_video(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

send_video_note(*args, **kwargs)
Shortcut for:

```
bot.send_video_note(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

send_voice(*args, **kwargs)
Shortcut for:

```
bot.send_voice(Chat.id, *args, **kwargs)
```

Where Chat is the current instance.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

unban_member(*args, **kwargs)
Shortcut for:

```
bot.unban_chat_member(update.message.chat.id, *args, **kwargs)
```

**Returns** If the action was sent successfully.

**Return type** `bool`

class telegram.ChatMember(user, status, until_date=None, can_be_edited=None,
can_change_info=None, can_post_messages=None,
can_edit_messages=None, can_delete_messages=None,
can_invite_users=None, can_restrict_members=None,
can_pin_messages=None, can_promote_members=None,
can_send_messages=None, can_send_media_messages=None,
can_send_other_messages=None, can_add_web_page_previews=None,
**kwargs)
```

**Bases:** telegram.base.TelegramObject

This object contains information about one member of the chat.

user

```
telegram.User – Information about the user.
```
status
   str – The member’s status in the chat.

until_date
   datetime.datetime – Optional. Date when restrictions will be lifted for this user.

can_be_edited
   bool – Optional. If the bot is allowed to edit administrator privileges of that user.

can_change_info
   bool – Optional. If the administrator can change the chat title, photo and other settings.

can_post_messages
   bool – Optional. If the administrator can post in the channel.

can_edit_messages
   bool – Optional. If the administrator can edit messages of other users.

can_delete_messages
   bool – Optional. If the administrator can delete messages of other users.

can_invite_users
   bool – Optional. If the administrator can invite new users to the chat.

can_restrict_members
   bool – Optional. If the administrator can restrict, ban or unban chat members.

can_pin_messages
   bool – Optional. If the administrator can pin messages.

can_promote_members
   bool – Optional. If the administrator can add new administrators.

can_send_messages
   bool – Optional. If the user can send text messages, contacts, locations and venues.

can_send_media_messages
   bool – Optional. If the user can send media messages, implies can_send_messages.

can_send_other_messages
   bool – Optional. If the user can send animations, games, stickers and use inline bots, implies can_send_media_messages.

can_add_web_page_previews
   bool – Optional. If user may add web page previews to his messages, implies can_send_media_messages

Parameters

- user(telegram.User) – Information about the user.
- until_date(datetime.datetime, optional) – Restricted and kicked only. Date when restrictions will be lifted for this user.
- can_be_edited(bool, optional) – Administrators only. True, if the bot is allowed to edit administrator privileges of that user.
- can_change_info(bool, optional) – Administrators only. True, if the administrator can change the chat title, photo and other settings.
- can_post_messages(bool, optional) – Administrators only. True, if the administrator can post in the channel, channels only.
- can_edit_messages(bool, optional) – Administrators only. True, if the administrator can edit messages of other users, channels only.
• **can_delete_messages** *(bool, optional) – Administrators only. True, if the administrator can delete messages of other user.*

• **can_invite_users** *(bool, optional) – Administrators only. True, if the administrator can invite new users to the chat.*

• **can_restrict_members** *(bool, optional) – Administrators only. True, if the administrator can restrict, ban or unban chat members.*

• **can_pin_messages** *(bool, optional) – Administrators only. True, if the administrator can pin messages, supergroups only.*

• **can_promote_members** *(bool, optional) – Administrators only. True, if the administrator can add new administrators with a subset of his own privileges or demote administrators that he has promoted, directly or indirectly (promoted by administrators that were appointed by the user).*

• **can_send_messages** *(bool, optional) – Restricted only. True, if the user can send text messages, contacts, locations and venues.*

• **can_send_media_messages** *(bool, optional) – Restricted only. True, if the user can send audios, documents, photos, videos, video notes and voice notes, implies can_send_messages.*

• **can_send_other_messages** *(bool, optional) – Restricted only. True, if the user can send animations, games, stickers and use inline bots, implies can_send_media_messages.*

• **can_add_web_page_previews** *(bool, optional) – Restricted only. True, if user may add web page previews to his messages, implies can_send_media_messages.*

```python
ADMINISTRATOR = 'administrator'
str – 'administrator'

CREATOR = 'creator'
str – 'creator'

KICKED = 'kicked'
str – 'kicked'

LEFT = 'left'
str – 'left'

MEMBER = 'member'
str – 'member'

REstricted = 'restricted'
str – 'restricted'

classmethod de_json(data, bot)
to_dict()

class telegram.ChatAction
Bases: object

Helper class to provide constants for different chat actions.

FIND_LOCATION = 'find_location'
str – 'find_location'

RECORD_AUDIO = 'record_audio'
str – 'record_audio'

RECORD_VIDEO = 'record_video'
str – 'record_video'

RECORD_VIDEO_NOTE = 'record_video_note'
str – 'record_video_note'
```
class telegram.ChosenInlineResult(result_id, from_user, query, location=None, inline_message_id=None, **kwargs)

Represents a result of an inline query that was chosen by the user and sent to their chat partner.

Note: In Python from is a reserved word, use from_user instead.

result_id
    str – The unique identifier for the result that was chosen.

from_user
    telegram.User – The user that chose the result.

location
    telegram.Location – Optional. Sender location.

inline_message_id
    str – Optional. Identifier of the sent inline message.

query
    str – The query that was used to obtain the result.

Parameters

- result_id (str) – The unique identifier for the result that was chosen.
- from_user (telegram.User) – The user that chose the result.
- location (telegram.Location, optional) – Sender location, only for bots that require user location.
- inline_message_id (str, optional) – Identifier of the sent inline message. Available only if there is an inline keyboard attached to the message. Will be also received in callback queries and can be used to edit the message.
- query (str) – The query that was used to obtain the result.
- **kwargs (dict) – Arbitrary keyword arguments.

classmethod de_json(data, bot)

class telegram.CallbackQuery(id, from_user, chat_instance, message=None, data=None, inline_message_id=None, game_short_name=None, bot=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents an incoming callback query from a callback button in an inline keyboard.
If the button that originated the query was attached to a message sent by the bot, the field `message` will be present. If the button was attached to a message sent via the bot (in inline mode), the field `inline_message_id` will be present.

**Note:**
- In Python, `from` is a reserved word, use `from_user` instead.
- Exactly one of the fields `data` or `game_short_name` will be present.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>id</code></td>
<td>str – Unique identifier for this query.</td>
</tr>
<tr>
<td><code>from_user</code></td>
<td><code>telegram.User</code> – Sender.</td>
</tr>
<tr>
<td><code>message</code></td>
<td><code>telegram.Message</code> – Optional. Message with the callback button that originated the query.</td>
</tr>
<tr>
<td><code>inline_message_id</code></td>
<td>str – Optional. Identifier of the message sent via the bot in inline mode, that originated the query.</td>
</tr>
<tr>
<td><code>chat_instance</code></td>
<td>str – Optional. Global identifier, uniquely corresponding to the chat to which the message with the callback button was sent.</td>
</tr>
<tr>
<td><code>data</code></td>
<td>str – Optional. Data associated with the callback button.</td>
</tr>
<tr>
<td><code>game_short_name</code></td>
<td>str – Optional. Short name of a Game to be returned.</td>
</tr>
</tbody>
</table>

**Parameters**
- `id`(str) – Unique identifier for this query.
- `from_user`(telegram.User) – Sender.
- `message`(telegram.Message, optional) – Message with the callback button that originated the query. Note that message content and message date will not be available if the message is too old.
- `inline_message_id`(str, optional) – Identifier of the message sent via the bot in inline mode, that originated the query.
- `chat_instance`(str, optional) – Global identifier, uniquely corresponding to the chat to which the message with the callback button was sent. Useful for high scores in games.
- `data`(str, optional) – Data associated with the callback button. Be aware that a bad client can send arbitrary data in this field.
- `game_short_name`(str, optional) – Short name of a Game to be returned, serves as the unique identifier for the game

**Note:** After the user presses an inline button, Telegram clients will display a progress bar until you call `answer`. It is, therefore, necessary to react by calling `telegram.Bot.answer_callback_query` even if no notification to the user is needed (e.g., without specifying any of the optional parameters).

```python
answer(*args, **kwargs)
```

Shortcut for:
bot.answer_callback_query(update.callback_query.id, *args, **kwargs)

**Returns**  On success, True is returned.

**Return type**  bool

**classmethod de_json**(data, bot)

**edit_message_caption**(caption, *args, **kwargs)

Shortcut for either:

bot.edit_message_caption(caption=caption,
    chat_id=update.callback_query.message.chat_id,
    message_id=update.callback_query.message.message_id,
    *args, **kwargs)

or:

bot.edit_message_caption(caption=caption,
    inline_message_id=update.callback_query.inline_message_id,
    *args, **kwargs)

**Returns**  On success, if edited message is sent by the bot, the edited Message is returned, otherwise True is returned.

**Return type**  telegram.Message

**edit_message_reply_markup**(reply_markup, *args, **kwargs)

Shortcut for either:

bot.edit_message_replyMarkup(chat_id=update.callback_query.message.chat_id,
    message_id=update.callback_query.message.message_id,
    reply_markup=reply_markup,
    *args, **kwargs)

or:

bot.edit_message_reply_markup(inline_message_id=update.callback_query.inline_message_id,
    reply_markup=reply_markup,
    *args, **kwargs)

**Returns**  On success, if edited message is sent by the bot, the edited Message is returned, otherwise True is returned.

**Return type**  telegram.Message

**edit_message_text**(text, *args, **kwargs)

Shortcut for either:

bot.edit_message_text(text, chat_id=update.callback_query.message.chat_id,
    message_id=update.callback_query.message.message_id,
    *args, **kwargs)

or:

bot.edit_message_text(text, inline_message_id=update.callback_query.inline_message_id,
    *args, **kwargs)
Returns On success, if edited message is sent by the bot, the edited Message is returned, otherwise True is returned.

Return type telegram.Message

class telegram.Contact(phone_number, first_name, last_name=None, user_id=None, vcard=None, **kwargs)
Bases: telegram.base.TelegramObject

This object represents a phone contact.

phone_number
  str – Contact’s phone number.

first_name
  str – Contact’s first name.

last_name
  str – Optional. Contact’s last name.

user_id
  int – Optional. Contact’s user identifier in Telegram.

vcard
  str – Optional. Additional data about the contact in the form of a vCard.

Parameters

* phone_number (str) – Contact’s phone number.
* first_name (str) – Contact’s first name.
* last_name (str, optional) – Contact’s last name.
* user_id (int, optional) – Contact’s user identifier in Telegram.
* vcard (str, optional) – Additional data about the contact in the form of a vCard.
* **kwargs (dict) – Arbitrary keyword arguments.

classmethod de_json(data, bot)

class telegram.Document(file_id, thumb=None, file_name=None, mime_type=None, file_size=None, bot=None, **kwargs)
Bases: telegram.base.TelegramObject

This object represents a general file (as opposed to photos, voice messages and audio files).

file_id
  str – Unique file identifier.

thumb

file_name
  str – Original filename.

mime_type
  str – Optional. MIME type of the file.

file_size
  int – Optional. File size.

bot

Parameters

* file_id (str) – Unique file identifier
• `thumb` ([`telegram.PhotoSize`], optional) – Document thumbnail as defined by sender.

• `file_name` (str, optional) – Original filename as defined by sender.

• `mime_type` (str, optional) – MIME type of the file as defined by sender.

• `file_size` (int, optional) – File size.

• `bot` ([`telegram.Bot`], optional) – The Bot to use for instance methods.

• `**kwargs` (dict) – Arbitrary keyword arguments.

classmethod `de_json` *(data, bot)*

`get_file` *(timeout=None, **kwargs)*

Convenience wrapper over `telegram.Bot.get_file`

Parameters

• `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• `**kwargs` (dict) – Arbitrary keyword arguments.

Returns `telegram.File`

Raises `telegram.TelegramError`

class `telegram.File` *(file_id, bot=None, file_size=None, file_path=None, **kwargs)*

Bases: `telegram.base.TelegramObject`

This object represents a file ready to be downloaded. The file can be downloaded with `download`. It is guaranteed that the link will be valid for at least 1 hour. When the link expires, a new one can be requested by calling `get_file`.

**Note:** Maximum file size to download is 20 MB

`file_id`

str – Unique identifier for this file.

`file_size`

str – Optional. File size.

`file_path`

str – Optional. File path. Use `download` to get the file.

Parameters

• `file_id` (str) – Unique identifier for this file.

• `file_size` (int, optional) – Optional. File size, if known.

• `file_path` (str, optional) – File path. Use `download` to get the file.

• `bot` ([`telegram.Bot`], optional) – Bot to use with shortcut method.

• `**kwargs` (dict) – Arbitrary keyword arguments.

**Note:** If you obtain an instance of this class from `telegram.PassportFile.get_file`, then it will automatically be decrypted as it downloads when you call `download()`.

classmethod `de_json` *(data, bot)*
download (custom_path=None, out=None, timeout=None)
Download this file. By default, the file is saved in the current working directory with its original filename as reported by Telegram. If a custom_path is supplied, it will be saved to that path instead. If out is defined, the file contents will be saved to that object using the out.write method.

Note: custom_path and out are mutually exclusive.

Parameters
• custom_path (str, optional) – Custom path.
• out (io.BufferedWriter, optional) – A file-like object. Must be opened for writing in binary mode, if applicable.
• timeout (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

Returns The same object as out if specified. Otherwise, returns the filename downloaded to.
Return type str | io.BufferedWriter
 Raises ValueError – If both custom_path and out are passed.

download_as_bytearray (buf=None)
Download this file and return it as a bytearray.

Parameters buf (bytearray, optional) – Extend the given bytearray with the downloaded data.

Returns The same object as buf if it was specified. Otherwise a newly allocated bytearray.
Return type bytearray

set_credentials (credentials)

class telegram.ForceReply (force_reply=True, selective=False, **kwargs)
Bases: telegram.replymarkup.ReplyMarkup

Upon receiving a message with this object, Telegram clients will display a reply interface to the user (act as if the user has selected the bot’s message and tapped ‘Reply’). This can be extremely useful if you want to create user-friendly step-by-step interfaces without having to sacrifice privacy mode.

force_reply
True – Shows reply interface to the user.

selective
bool – Optional. Force reply from specific users only.

Parameters
• selective (bool, optional) – Use this parameter if you want to force reply from specific users only. Targets:
  1. users that are @mentioned in the text of the Message object
  2. if the bot’s message is a reply (has reply_to_message_id), sender of the original message.
• **kwargs (dict) – Arbitrary keyword arguments.
class telegram.InlineKeyboardButton(text, url=None, callback_data=None, switch_inline_query=None, switch_inline_query_current_chat=None, callback_game=None, pay=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents one button of an inline keyboard.

Note: You must use exactly one of the optional fields. Mind that callback_game is not working as expected. Putting a game short name in it might, but is not guaranteed to work.

**text**

str – Label text on the button.

**url**

str – Optional. HTTP url to be opened when button is pressed.

**callback_data**

str – Optional. Data to be sent in a callback query to the bot when button is pressed, 1-64 bytes.

**switch_inline_query**

str – Optional. Will prompt the user to select one of their chats, open that chat and insert the bot’s username and the specified inline query in the input field.

**switch_inline_query_current_chat**

str – Optional. Will insert the bot’s username and the specified inline query in the current chat’s input field.

**callback_game**

telegram.CallbackGame – Optional. Description of the game that will be launched when the user presses the button.

**pay**

bool – Optional. Specify True, to send a Pay button.

Parameters

- **text** (str) – Label text on the button.
- **url** (str) – HTTP url to be opened when button is pressed.
- **callback_data** (str, optional) – Data to be sent in a callback query to the bot when button is pressed, 1-64 bytes.
- **switch_inline_query** (str, optional) – If set, pressing the button will prompt the user to select one of their chats, open that chat and insert the bot’s username and the specified inline query in the input field. Can be empty, in which case just the bot’s username will be inserted. This offers an easy way for users to start using your bot in inline mode when they are currently in a private chat with it. Especially useful when combined with switch_pm* actions - in this case the user will be automatically returned to the chat they switched from, skipping the chat selection screen.
- **switch_inline_query_current_chat** (str, optional) – If set, pressing the button will insert the bot’s username and the specified inline query in the current chat’s input field. Can be empty, in which case only the bot’s username will be inserted. This offers a quick way for the user to open your bot in inline mode in the same chat - good for selecting something from multiple options.
- **callback_game** (telegram.CallbackGame, optional) – Description of the game that will be launched when the user presses the button. This type of button must always be the first button in the first row.
- **pay** (bool, optional) – Specify True, to send a Pay button. This type of button must always be the first button in the first row.
**kwargs (dict) – Arbitrary keyword arguments.

class telegram.InlineKeyboardMarkup(inline_keyboard, **kwargs)
   Bases: telegram.replymarkup.ReplyMarkup

This object represents an inline keyboard that appears right next to the message it belongs to.

inline_keyboard
   List[List[telegram.InlineKeyboardButton]] – Array of button rows, each represented by an Array of InlineKeyboardButton objects.

Parameters

   • inline_keyboard (List[List[telegram.InlineKeyboardButton]]) – Array of button rows, each represented by an Array of InlineKeyboardButton objects.
   • **kwargs (dict) – Arbitrary keyword arguments.

classmethod from_button(button, **kwargs)
   Shortcut for:

   InlineKeyboardMarkup([[button]], **kwargs)

Return an InlineKeyboardMarkup from a single InlineKeyboardButton

Parameters

   • button (telegram.InlineKeyboardButton) – The button to use in the markup
   • **kwargs (dict) – Arbitrary keyword arguments.

classmethod from_column(button_column, **kwargs)
   Shortcut for:

   InlineKeyboardMarkup([[button] for button in button_column], **kwargs)

Return an InlineKeyboardMarkup from a single column of InlineKeyboardButtons

Parameters

   • button_column (List[telegram.InlineKeyboardButton]) – The button to use in the markup
   • **kwargs (dict) – Arbitrary keyword arguments.

classmethod from_row(button_row, **kwargs)
   Shortcut for:

   InlineKeyboardMarkup([button_row], **kwargs)

Return an InlineKeyboardMarkup from a single row of InlineKeyboardButtons

Parameters

   • button_row (List[telegram.InlineKeyboardButton]) – The button to use in the markup
   • **kwargs (dict) – Arbitrary keyword arguments.

to_dict()

class telegram.InlineQuery(id, from_user, query, offset, location=None, bot=None, **kwargs)
   Bases: telegram.base.TelegramObject

This object represents an incoming inline query. When the user sends an empty query, your bot could return some default or trending results.

Note:
• In Python `from` is a reserved word, use `from_user` instead.

**id**

str – Unique identifier for this query.

**from_user**

*telegram.User* – Sender.

**location**

*telegram.Location* – Optional. Sender location, only for bots that request user location.

**query**

str – Text of the query (up to 512 characters).

**offset**

str – Offset of the results to be returned, can be controlled by the bot.

**Parameters**

- **id**(str) – Unique identifier for this query.
- **from_user**(telegram.User) – Sender.
- **location**(telegram.Location, optional) – Sender location, only for bots that request user location.
- **query**(str) – Text of the query (up to 512 characters).
- **offset**(str) – Offset of the results to be returned, can be controlled by the bot.
- **bot**(telegram.Bot, optional) – The Bot to use for instance methods.
- ****kwargs**(dict) – Arbitrary keyword arguments.

```python
answer(*args, **kwargs)
```

Shortcut for:

```python
bot.answer_inline_query(update.inline_query.id, *args, **kwargs)
```

**Parameters**

- **results**(List[telegram.InlineQueryResult]) – A list of results for the inline query.
- **cache_time**(int, optional) – The maximum amount of time in seconds that the result of the inline query may be cached on the server. Defaults to 300.
- **is_personal**(bool, optional) – Pass True, if results may be cached on the server side only for the user that sent the query. By default, results may be returned to any user who sends the same query.
- **next_offset**(str, optional) – Pass the offset that a client should send in the next query with the same text to receive more results. Pass an empty string if there are no more results or if you don’t support pagination. Offset length can’t exceed 64 bytes.
- **switch_pm_text**(str, optional) – If passed, clients will display a button with specified text that switches the user to a private chat with the bot and sends the bot a start message with the parameter switch_pm_parameter.
- **switch_pm_parameter**(str, optional) – Deep-linking parameter for the /start message sent to the bot when user presses the switch button. 1-64 characters, only A-Z, a-z, 0-9, _, - and - are allowed.

```python
classmethod de_json(data, bot)
```
class telegram.InlineQueryResult

Bases: telegram.base.TelegramObject

Baseclass for the InlineQueryResult* classes.

type
   str – Type of the result.

id
   str – Unique identifier for this result, 1-64 Bytes.

Parameters

   • type (str) – Type of the result.
   • id (str) – Unique identifier for this result, 1-64 Bytes.
   • **kwargs (dict) – Arbitrary keyword arguments.

class telegram.InlineQueryResult

Bases: telegram.base.TelegramObject

Baseclass for the InlineQueryResult* classes.

type
   str – Type of the result.

id
   str – Unique identifier for this result, 1-64 Bytes.

Parameters

   • type (str) – Type of the result.
   • id (str) – Unique identifier for this result, 1-64 Bytes.
   • **kwargs (dict) – Arbitrary keyword arguments.

class telegram.InlineQueryResultArticle

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

This object represents a Telegram InlineQueryResultArticle.

type
   str – ‘article’.

id
   str – Unique identifier for this result, 1-64 Bytes.

title
   str – Title of the result.

input_message_content
   telegram.InputMessageContent – Content of the message to be sent.

reply_markup
   telegram.ReplyMarkup – Optional. Inline keyboard attached to the message.

url
   str – Optional. URL of the result.

hide_url
   bool – Optional. Pass True, if you don’t want the URL to be shown in the message.

description
   str – Optional. Short description of the result.
thumb_url
    str – Optional. Url of the thumbnail for the result.

thumb_width
    int – Optional. Thumbnail width.

thumb_height
    int – Optional. Thumbnail height.

Parameters

• id (str) – Unique identifier for this result, 1-64 Bytes.

• title (str) – Title of the result.

• input_message_content (telegram.InputMessageContent) – Content of the message to be sent.

• reply_markup (telegram.ReplyMarkup, optional) – Inline keyboard attached to the message

• url (str, optional) – URL of the result.

• hide_url (bool, optional) – Pass True, if you don’t want the URL to be shown in the message.

• description (str, optional) – Short description of the result.

• thumb_url (str, optional) – Url of the thumbnail for the result.

• thumb_width (int, optional) – Thumbnail width.

• thumb_height (int, optional) – Thumbnail height.

• **kwargs (dict) – Arbitrary keyword arguments.

class telegram.InlineQueryResultAudio(id, audio_url, title, performer=None, audio_duration=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to an mp3 audio file. By default, this audio file will be sent by the user. Alternatively, you can use input_message_content to send a message with the specified content instead of the audio.

type
    str – ‘audio’.

id
    str – Unique identifier for this result, 1-64 bytes.

audio_url
    str – A valid URL for the audio file.

title
    str – Title.

performer
    str – Optional. Caption, 0-200 characters.

audio_duration
    str – Optional. Performer.

caption
    str – Optional. Audio duration in seconds.
parse_mode
str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

reply_markup
telemetry.InlineKeyboardMarkup – Optional. Inline keyboard attached to the message.

input_message_content
telemetry.InputMessageContent – Optional. Content of the message to be sent instead of the audio.

Parameters

• id (str) – Unique identifier for this result, 1-64 bytes.
• audio_url (str) – A valid URL for the audio file.
• title (str) – Title.
• performer (str, optional) – Caption, 0-200 characters.
• audio_duration (str, optional) – Performer.
• caption (str, optional) – Audio duration in seconds.
• parse_mode (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
• reply_markup (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.
• input_message_content (telegram.InputMessageContent, optional) – Content of the message to be sent instead of the audio.
• **kwargs (dict) – Arbitrary keyword arguments.
**input_message_content**

*telegram.InputMessageContent* – Optional. Content of the message to be sent instead of the audio.

**Parameters**

- **id** *(str)* – Unique identifier for this result, 1-64 bytes.
- **audio_file_id** *(str)* – A valid file identifier for the audio file.
- **caption** *(str, optional)* – Caption, 0-1024 characters
- **parse_mode** *(str, optional)* – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.
- **reply_markup** *(telegram.InlineKeyboardMarkup, optional)* – Inline keyboard attached to the message.
- **input_message_content** *(telegram.InputMessageContent, optional)* – Content of the message to be sent instead of the audio.
- ****kwargs** *(dict)* – Arbitrary keyword arguments.

```python
class telegram.InlineQueryResultCachedDocument(id, title, document_file_id, description=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)
```

Bases: *telegram.inline.inlinequeryresult.InlineQueryResult*

Represents a link to a file stored on the Telegram servers. By default, this file will be sent by the user with an optional caption. Alternatively, you can use **input_message_content** to send a message with the specified content instead of the file.

**type** *(str)* – ‘document’.

**id** *(str)* – Unique identifier for this result, 1-64 bytes.

**title** *(str)* – Title for the result.

**document_file_id** *(str)* – A valid file identifier for the file.

**description** *(str)* – Optional. Short description of the result.

**caption** *(str)* – Optional. Caption, 0-1024 characters

**parse_mode** *(str)* – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption.. See the constants in *telegram.ParseMode* for the available modes.

**reply_markup** *(telegram.InlineKeyboardMarkup)* – Optional. Inline keyboard attached to the message.

**input_message_content** *(telegram.InputMessageContent)* – Optional. Content of the message to be sent instead of the file.

**Parameters**

- **id** *(str)* – Unique identifier for this result, 1-64 bytes.
• **title** (*str*) – Title for the result.
• **document_file_id** (*str*) – A valid file identifier for the file.
• **description** (*str*, optional) – Short description of the result.
• **caption** (*str*, optional) – Caption, 0-1024 characters
• **parse_mode** (*str*, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption.. See the constants in *telegram.ParseMode* for the available modes.
• **reply_markup** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.
• **input_message_content** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the file.
• ****kwargs** (*dict*) – Arbitrary keyword arguments.

class telegram.InlineQueryResultCachedGif

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to an animated GIF file stored on the Telegram servers. By default, this animated GIF file will be sent by the user with an optional caption. Alternatively, you can use *input_message_content* to send a message with specified content instead of the animation.

**type**

*str* – ‘gif’.

**id**

*str* – Unique identifier for this result, 1-64 bytes.

**gif_file_id**

*str* – A valid file identifier for the GIF file.

**title**

*str* – Optional. Title for the result.

**caption**

*str* – Optional. Caption, 0-1024 characters

**parse_mode**

*str* – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.

**reply_markup**

*telegram.InlineKeyboardMarkup* – Optional. Inline keyboard attached to the message.

**input_message_content**

*telegram.InputMessageContent* – Optional. Content of the message to be sent instead of the gif.

**Parameters**

• **id** (*str*) – Unique identifier for this result, 1-64 bytes.
• **gif_file_id** (*str*) – A valid file identifier for the GIF file.
• **title** (*str*, optional) – Title for the result.
• **caption** (*str*, optional) – Caption, 0-1024 characters
• **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

• **reply_markup** (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.

• **input_message_content** (telegram.InputMessageContent, optional) – Content of the message to be sent instead of the gif.

• **kwargs** (dict) – Arbitrary keyword arguments.

```python
class telegram.InlineQueryResultCachedMpeg4Gif(id, mpeg4_file_id, title=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)
```

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to a video animation (H.264/MPEG-4 AVC video without sound) stored on the Telegram servers. By default, this animated MPEG-4 file will be sent by the user with an optional caption. Alternatively, you can use **input_message_content** to send a message with the specified content instead of the animation.

**type**

str – 'mpeg4_gif'.

**id**

str – Unique identifier for this result, 1-64 bytes.

**mpeg4_file_id**

str – A valid file identifier for the MP4 file.

**title**

str – Optional. Title for the result.

**caption**

str – Optional. Caption, 0-1024 characters

**parse_mode**

str – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

**reply_markup**

telegram.InlineKeyboardMarkup – Optional. Inline keyboard attached to the message.

**input_message_content**

telegram.InputMessageContent – Optional. Content of the message to be sent instead of the MPEG-4 file.

**Parameters**

• **id** (str) – Unique identifier for this result, 1-64 bytes.

• **mpeg4_file_id** (str) – A valid file identifier for the MP4 file.

• **title** (str, optional) – Title for the result.

• **caption** (str, optional) – Caption, 0-1024 characters

• **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

• **reply_markup** (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.
• **input_message_content** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the MPEG-4 file.

• **kwargs** (*dict*) – Arbitrary keyword arguments.

class telegram.InlineQueryResultCachedPhoto(id, photo_file_id, title=None, description=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)

Represents a link to a photo stored on the Telegram servers. By default, this photo will be sent by the user with an optional caption. Alternatively, you can use **input_message_content** to send a message with the specified content instead of the photo.

type
  *str* – ‘photo’.

id
  *str* – Unique identifier for this result, 1-64 bytes.

photo_file_id
  *str* – A valid file identifier of the photo.

title
  *str* – Optional. Title for the result.

description
  *str* – Optional. Short description of the result.

caption
  *str* – Optional. Caption, 0-1024 characters

parse_mode
  *str* – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.

reply_markup
  *telegram.InlineKeyboardMarkup* – Optional. Inline keyboard attached to the message.

input_message_content
  *telegram.InputMessageContent* – Optional. Content of the message to be sent instead of the photo.

Parameters

• **id** (*str*) – Unique identifier for this result, 1-64 bytes.

• **photo_file_id** (*str*) – A valid file identifier of the photo.

• **title** (*str*, optional) – Title for the result.

• **description** (*str*, optional) – Short description of the result.

• **caption** (*str*, optional) – Caption, 0-1024 characters

• **parse_mode** (*str*, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in *telegram.ParseMode* for the available modes.

• **reply_markup** (*telegram.InlineKeyboardMarkup*, optional) – Inline keyboard attached to the message.

• **input_message_content** (*telegram.InputMessageContent*, optional) – Content of the message to be sent instead of the photo.

• **kwargs** (*dict*) – Arbitrary keyword arguments.
```python
class telegram.InlineQueryResultCachedSticker(id, sticker_file_id, reply_markup=None, input_message_content=None, **kwargs)
```

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to a sticker stored on the Telegram servers. By default, this sticker will be sent by the user. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the sticker.

**type**

str – ‘sticker’.

**id**

str – Unique identifier for this result, 1-64 bytes.

**sticker_file_id**

str – A valid file identifier of the sticker.

**reply_markup**

telegram.InlineKeyboardMarkup – Optional. Inline keyboard attached to the message.

**input_message_content**

telegram.InputMessageContent – Optional. Content of the message to be sent instead of the sticker.

**Parameters**

- `id (str)` –
- `sticker_file_id (str)` –
- `reply_markup (telegram.InlineKeyboardMarkup, optional)` – Inline keyboard attached to the message.
- `input_message_content (telegram.InputMessageContent, optional)` – Content of the message to be sent instead of the sticker.
- `**kwargs (dict)` – Arbitrary keyword arguments.

```python
class telegram.InlineQueryResultCachedVideo(id, video_file_id, title, description=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)
```

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to a video file stored on the Telegram servers. By default, this video file will be sent by the user with an optional caption. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the video.

**type**

str – ‘video’.

**id**

str – Unique identifier for this result, 1-64 bytes.

**video_file_id**

str – A valid file identifier for the video file.

**title**

str – Title for the result.

**description**

str – Optional. Short description of the result.

**caption**

str – Optional. Caption, 0-1024 characters.
parse_mode
str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

reply_markup
telegram.InlineKeyboardMarkup – Optional. Inline keyboard attached to the message.

input_message_content
telegram.InputMessageContent – Optional. Content of the message to be sent instead of the video.

Parameters

• id (str) – Unique identifier for this result, 1-64 bytes.
• video_file_id (str) – A valid file identifier for the video file.
• title (str) – Title for the result.
• description (str, optional) – Short description of the result.
• caption (str, optional) – Caption, 0-1024 characters.
• parse_mode (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
• reply_markup (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.
• input_message_content (telegram.InputMessageContent, optional) – Content of the message to be sent instead of the video.
• **kwargs (dict) – Arbitrary keyword arguments.

class telegram.InlineQueryResultCachedVoice(id, voice_file_id, title, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to a voice message stored on the Telegram servers. By default, this voice message will be sent by the user. Alternatively, you can use input_message_content to send a message with the specified content instead of the voice message.

type
str – ‘voice’.

id
str – Unique identifier for this result, 1-64 bytes.

voice_file_id
str – A valid file identifier for the voice message.

title
str – Voice message title.

caption
str – Optional. Caption, 0-1024 characters.

parse_mode
str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

reply_markup
telegram.InlineKeyboardMarkup – Optional. Inline keyboard attached to the message.
**input_message_content**

`telegram.InputMessageContent` – Optional. Content of the message to be sent instead of the voice.

**Parameters**

- **id** (str) – Unique identifier for this result, 1-64 bytes.
- **voice_file_id** (str) – A valid file identifier for the voice message.
- **title** (str) – Voice message title.
- **caption** (str, optional) – Caption, 0-1024 characters.
- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **reply_markup** (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- **input_message_content** (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the voice.
- ****kwargs** (dict) – Arbitrary keyword arguments.

**class** `telegram.InlineQueryResultContact`

Represents a contact with a phone number. By default, this contact will be sent by the user. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the contact.

**type**

str – ‘contact’.

**id**

str – Unique identifier for this result, 1-64 bytes.

**phone_number**

str – Contact’s phone number.

**first_name**

str – Contact’s first name.

**last_name**

str – Optional. Contact’s last name.

**vcard**

str – Optional. Additional data about the contact in the form of a vCard, 0-2048 bytes.

**reply_markup**

`telegram.InlineKeyboardMarkup` – Optional. Inline keyboard attached to the message.

**input_message_content**

`telegram.InputMessageContent` – Optional. Content of the message to be sent instead of the contact.

**thumb_url**

str – Optional. Url of the thumbnail for the result.

**thumb_width**

int – Optional. Thumbnail width.
thumb_height
    int – Optional. Thumbnail height.

Parameters

- **id** (str) – Unique identifier for this result, 1-64 bytes.
- **phone_number** (str) – Contact’s phone number.
- **first_name** (str) – Contact’s first name.
- **last_name** (str, optional) – Contact’s last name.
- **vcard** (str, optional) – Additional data about the contact in the form of a vCard, 0-2048 bytes.
- **reply_markup** (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.
- **input_message_content** (telegram.InputMessageContent, optional) – Content of the message to be sent instead of the contact.
- **thumb_url** (str, optional) – Url of the thumbnail for the result.
- **thumb_width** (int, optional) – Thumbnail width.
- **thumb_height** (int, optional) – Thumbnail height.
- ****kwargs** (dict) – Arbitrary keyword arguments.

class telegram.InlineQueryResultDocument (id, document_url, title, mime_type, caption=None, description=None, reply_markup=None, input_message_content=None, thumb_url=None, thumb_width=None, thumb_height=None, parse_mode=None, **kwargs)

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to a file. By default, this file will be sent by the user with an optional caption. Alternatively, you can use **input_message_content** to send a message with the specified content instead of the file. Currently, only .PDF and .ZIP files can be sent using this method.

type
    str – ‘document’.

id
    str – Unique identifier for this result, 1-64 bytes.

title
    str – Title for the result.

caption
    str – Optional. Caption, 0-1024 characters

parse_mode
    str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in **telegram.ParseMode** for the available modes.

document_url
    str – A valid URL for the file.

mime_type
    str – Mime type of the content of the file, either “application/pdf” or “application/zip”.

description
    str – Optional. Short description of the result.
reply_markup
    telegram.InlineKeyboardMarkup – Optional. Inline keyboard attached to the message.

input_message_content
    telegram.InputMessageContent – Optional. Content of the message to be sent instead of the file.

thumb_url
    str – Optional. URL of the thumbnail (jpeg only) for the file.

thumb_width
    int – Optional. Thumbnail width.

thumb_height
    int – Optional. Thumbnail height.

Parameters

- id (str) – Unique identifier for this result, 1-64 bytes.
- title (str) – Title for the result.
- caption (str, optional) – Caption, 0-1024 characters
- parse_mode (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
- document_url (str) – A valid URL for the file.
- mime_type (str) – Mime type of the content of the file, either “application/pdf” or “application/zip”.
- description (str, optional) – Short description of the result.
- reply_markup (telegram.InlineKeyboardMarkup) – Optional. Inline keyboard attached to the message.
- input_message_content (telegram.InputMessageContent) – Optional. Content of the message to be sent instead of the file.
- thumb_url (str, optional) – URL of the thumbnail (jpeg only) for the file.
- thumb_width (int, optional) – Thumbnail width.
- thumb_height (int, optional) – Thumbnail height.
- **kwargs (dict) – Arbitrary keyword arguments.

class telegram.InlineQueryResultGif(id, gif_url, thumb_url, gif_width=None, gif_height=None, title=None, caption=None, reply_markup=None, input_message_content=None, gif_duration=None, parse_mode=None, **kwargs)

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to an animated GIF file. By default, this animated GIF file will be sent by the user with optional caption. Alternatively, you can use input_message_content to send a message with the specified content instead of the animation.

type
    str – ‘gif’.

id
    str – Unique identifier for this result, 1-64 bytes.

gif_url
    str – A valid URL for the GIF file. File size must not exceed 1MB.

gif_width
    int – Optional. Width of the GIF.
gif_height
    int – Optional. Height of the GIF.

gif_duration
    int – Optional. Duration of the GIF.

thumb_url
    str – URL of the static thumbnail for the result (jpeg or gif).

title
    str – Optional. Title for the result.

caption
    str – Optional. Caption, 0-1024 characters

parse_mode
    str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-
    width text or inline URLs in the media caption. See the constants in telegram.ParseMode for
    the available modes.

reply_markup
    telegram.InlineKeyboardMarkup – Optional. Inline keyboard attached to the message.

input_message_content
    telegram.InputMessageContent – Optional. Content of the message to be sent instead of the
    gif.

Parameters

• id (str) – Unique identifier for this result, 1-64 bytes.
• gif_url (str) – A valid URL for the GIF file. File size must not exceed 1MB.
• gif_width (int, optional) – Width of the GIF.
• gif_height (int, optional) – Height of the GIF.
• gif_duration (int, optional) – Duration of the GIF
• thumb_url (str) – URL of the static thumbnail for the result (jpeg or gif).
• title (str, optional) – Title for the result.caption (str, optional):
• caption (str, optional) – Caption, 0-1024 characters
• parse_mode (str, optional) – Send Markdown or HTML, if you want Telegram
    apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the
    constants in telegram.ParseMode for the available modes.
• reply_markup (telegram.InlineKeyboardMarkup, optional) – Inline key-
    board attached to the message.
• input_message_content (telegram.InputMessageContent, optional) –
    Content of the message to be sent instead of the gif.
• **kwargs (dict) – Arbitrary keyword arguments.

class telegram.InlineQueryResultLocation

    id (str) – Unique identifier for this result, 1-64 bytes.
    latitude (float) – Latitude of the location.
    longitude (float) – Longitude of the location.
    title (str, optional) – Title for the result.
    live_period (int, optional) – Duration of the live location (1-180 seconds).
    reply_markup (telegram.InlineKeyboardMarkup, optional) – Inline keyboard
    attached to the message.
    input_message_content (telegram.InputMessageContent, optional) –
    Content of the message to be sent instead of the location.
    thumb_url (str) – URL of the static thumbnail for the result (jpeg or gif).
    thumb_width (int, optional) – Width of the thumbnail.
    thumb_height (int, optional) – Height of the thumbnail.

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a location on a map. By default, the location will be sent by the user. Alternatively, you can use
input_message_content to send a message with the specified content instead of the location.

type
    str – ‘location’.
id
str – Unique identifier for this result, 1-64 bytes.

latitude
float – Location latitude in degrees.

longitude
float – Location longitude in degrees.

title
str – Location title.

live_period
int – Optional. Period in seconds for which the location can be updated, should be between 60 and 86400.

reply_markup
telegram.InlineKeyboardMarkup – Optional. Inline keyboard attached to the message.

input_message_content
telegram.InputMessageContent – Optional. Content of the message to be sent instead of the location.

thumb_url
str – Optional. Url of the thumbnail for the result.

thumb_width
int – Optional. Thumbnail width.

thumb_height
int – Optional. Thumbnail height.

Parameters

• id (str) – Unique identifier for this result, 1-64 bytes.
• latitude (float) – Location latitude in degrees.
• longitude (float) – Location longitude in degrees.
• title (str) – Location title.
• live_period (int, optional) – Period in seconds for which the location can be updated, should be between 60 and 86400.
• reply_markup (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.
• input_message_content (telegram.InputMessageContent, optional) – Content of the message to be sent instead of the location.
• thumb_url (str, optional) – Url of the thumbnail for the result.
• thumb_width (int, optional) – Thumbnail width.
• thumb_height (int, optional) – Thumbnail height.
• **kwargs (dict) – Arbitrary keyword arguments.

class telegram.InlineQueryResultMpeg4Gif (id, mpeg4_url, thumb_url, mpeg4_width=None, mpeg4_height=None, title=None, caption=None, reply_markup=None, input_message_content=None, mpeg4_duration=None, parse_mode=None, **kwargs)

Bases: telegram.inline.inlinequeryresult.InlineQueryResult
Represents a link to a video animation (H.264/MPEG-4 AVC video without sound). By default, this animated MPEG-4 file will be sent by the user with optional caption. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the animation.

```python

type
  str – ‘mpeg4_gif’.

id
  str – Unique identifier for this result, 1-64 bytes.

mpeg4_url
  str – A valid URL for the MP4 file. File size must not exceed 1MB.

mpeg4_width
  int – Optional. Video width.

mpeg4_height
  int – Optional. Video height.

mpeg4_duration
  int – Optional. Video duration.

thumb_url
  str – URL of the static thumbnail (jpeg or gif) for the result.

title
  str – Optional. Title for the result.

caption
  str – Optional. Caption, 0-1024 characters

parse_mode
  str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

reply_markup
  `telegram.InlineKeyboardMarkup` – Optional. Inline keyboard attached to the message.

input_message_content
  `telegram.InputMessageContent` – Optional. Content of the message to be sent instead of the MPEG-4 file.
```

Parameters

- id (str) – Unique identifier for this result, 1-64 bytes.
- mpeg4_url (str) – A valid URL for the MP4 file. File size must not exceed 1MB.
- mpeg4_width (int, optional) – Video width.
- mpeg4_height (int, optional) – Video height.
- mpeg4_duration (int, optional) – Video duration.
- thumb_url (str) – URL of the static thumbnail (jpeg or gif) for the result.
- title (str, optional) – Title for the result.
- caption (str, optional) – Caption, 0-1024 characters
- parse_mode (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- reply_markup (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.
- input_message_content (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the MPEG-4 file.
• **kwargs (dict) – Arbitrary keyword arguments.

```python
class telegram.InlineQueryResultPhoto(id, photo_url, thumb_url, photo_width=None, photo_height=None, title=None, description=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)
```

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to a photo. By default, this photo will be sent by the user with optional caption. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the photo.

**type**

str – ‘photo’.

**id**

str – Unique identifier for this result, 1-64 bytes.

**photo_url**

str – A valid URL of the photo. Photo must be in jpeg format. Photo size must not exceed 5MB.

**thumb_url**

str – URL of the thumbnail for the photo.

**photo_width**

int – Optional. Width of the photo.

**photo_height**

int – Optional. Height of the photo.

**title**

str – Optional. Title for the result.

**description**

str – Optional. Short description of the result.

**caption**

str – Optional. Caption, 0-1024 characters

**parse_mode**

str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

**reply_markup**

```python
telegram.InlineKeyboardMarkup – Optional. Inline keyboard attached to the message.
```

**input_message_content**

```python
telegram.InputMessageContent – Optional. Content of the message to be sent instead of the photo.
```

**Parameters**

- **id** (str) – Unique identifier for this result, 1-64 bytes.
- **photo_url** (str) – A valid URL of the photo. Photo must be in jpeg format. Photo size must not exceed 5MB.
- **thumb_url** (str) – URL of the thumbnail for the photo.
- **photo_width** (int, optional) – Width of the photo.
- **photo_height** (int, optional) – Height of the photo.
- **title** (str, optional) – Title for the result.
- **description** (str, optional) – Short description of the result.
• `caption` (str, optional) – Caption, 0-1024 characters

• `parse_mode` (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

• `reply_markup` (`telegram.InlineKeyboardMarkup`, optional) – Inline keyboard attached to the message.

• `input_message_content` (`telegram.InputMessageContent`, optional) – Content of the message to be sent instead of the photo.

• `**kwargs` (dict) – Arbitrary keyword arguments.

class `telegram.InlineQueryResultVenue`(
    id, latitude, longitude, title,
    address, foursquare_id=None, reply_markup=None,
    input_message_content=None, thumb_url=None,
    thumb_width=None, thumb_height=None,
    **kwargs)

Bases: `telegram.inline.inlinequeryresult.InlineQueryResult`

Represents a venue. By default, the venue will be sent by the user. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the venue.

def type
    str – ‘venue’.

id
    str – Unique identifier for this result, 1-64 Bytes.

latitude
    float – Latitude of the venue location in degrees.

longitude
    float – Longitude of the venue location in degrees.

title
    str – Title of the venue.

address
    str – Address of the venue.

foursquare_id
    str – Optional. Foursquare identifier of the venue if known.

foursquare_type
    str – Optional. Foursquare type of the venue, if known. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”.)

reply_markup
    `telegram.InlineKeyboardMarkup` – Optional. Inline keyboard attached to the message.

input_message_content
    `telegram.InputMessageContent` – Optional. Content of the message to be sent instead of the venue.

thumb_url
    str – Optional. Url of the thumbnail for the result.

thumb_width
    int – Optional. Thumbnail width.

thumb_height
    int – Optional. Thumbnail height.
• id (str) – Unique identifier for this result, 1-64 Bytes.
• latitude (float) – Latitude of the venue location in degrees.
• longitude (float) – Longitude of the venue location in degrees.
• title (str) – Title of the venue.
• address (str) – Address of the venue.
• foursquare_id (str, optional) – Foursquare identifier of the venue if known.
• foursquare_type (str, optional) – Foursquare type of the venue, if known. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”).
• reply_markup (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.
• input_message_content (telegram.InputMessageContent, optional) – Content of the message to be sent instead of the location.
• thumb_url (str, optional) – Url of the thumbnail for the result.
• thumb_width (int, optional) – Thumbnail width.
• thumb_height (int, optional) – Thumbnail height.
• **kwargs (dict) – Arbitrary keyword arguments.

```python
class telegram.InlineQueryResultVideo(id, video_url, mime_type, thumb_url, title, caption=None, video_width=None, video_height=None, video_duration=None, description=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)
```

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to a page containing an embedded video player or a video file. By default, this video file will be sent by the user with an optional caption. Alternatively, you can use input_message_content to send a message with the specified content instead of the video.

type
    str – ‘video’.

id
    str – Unique identifier for this result, 1-64 bytes.

video_url
    str – A valid URL for the embedded video player or video file.

mime_type
    str – Mime type of the content of video url, “text/html” or “video/mp4”.

thumb_url
    str – URL of the thumbnail (jpeg only) for the video.

title
    str – Title for the result.

caption
    str – Optional. Caption, 0-1024 characters

parse_mode
    str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

video_width
    int – Optional. Video width.
video_height
int – Optional. Video height.

video_duration
int – Optional. Video duration in seconds.

description
str – Optional. Short description of the result.

reply_markup
telegram.InlineKeyboardMarkup – Optional. Inline keyboard attached to the message.

input_message_content
telegram.InputMessageContent – Optional. Content of the message to be sent instead of the video.

Parameters

• id (str) – Unique identifier for this result, 1-64 bytes.
• video_url (str) – A valid URL for the embedded video player or video file.
• mime_type (str) – Mime type of the content of video url, “text/html” or “video/mp4”.
• thumb_url (str) – URL of the thumbnail (jpeg only) for the video.
• title (str) – Title for the result.
• caption (str, optional) – Caption, 0-1024 characters.
• parse_mode (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
• video_width (int, optional) – Video width.
• video_height (int, optional) – Video height.
• video_duration (int, optional) – Video duration in seconds.
• description (str, optional) – Short description of the result.
• reply_markup (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.
• input_message_content (telegram.InputMessageContent, optional) – Content of the message to be sent instead of the video.
• **kwargs (dict) – Arbitrary keyword arguments.

class telegram.InlineQueryResultVoice (id, voice_url, title, voice_duration=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, **kwargs)

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to a voice recording in an .ogg container encoded with OPUS. By default, this voice recording will be sent by the user. Alternatively, you can use input_message_content to send a message with the specified content instead of the the voice message.

type
str – ‘voice’.

id
str – Unique identifier for this result, 1-64 bytes.

voice_url
str – A valid URL for the voice recording.
title
str – Voice message title.

caption
str – Optional. Caption, 0-1024 characters.

parse_mode
str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-
width text or inline URLs in the media caption. See the constants in telegram.ParseMode for
the available modes.

voice_duration
int – Optional. Recording duration in seconds.

reply_markup
telegram.InlineKeyboardMarkup – Optional. Inline keyboard attached to the message.

input_message_content
telegram.InputMessageContent – Optional. Content of the message to be sent instead of the
voice.

Parameters

- **id**(str) – Unique identifier for this result, 1-64 bytes.
- **voice_url**(str) – A valid URL for the voice recording.
- **title**(str) – Voice message title.
- **caption**(str, optional) – Caption, 0-1024 characters.
- **parse_mode**(str, optional) – Send Markdown or HTML, if you want Telegram
  apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the
  constants in telegram.ParseMode for the available modes.
- **voice_duration**(int, optional) – Recording duration in seconds.
- **reply_markup**(telegram.InlineKeyboardMarkup, optional) – Inline key-
  board attached to the message.
- **input_message_content**(telegram.InputMessageContent, optional) –
  Content of the message to be sent instead of the voice.
- ****kwargs**(dict) – Arbitrary keyword arguments.

class telegram.InlineQueryResultGame(id, game_short_name, reply_markup=None, **kwargs)
Bases: telegram.inline.inlinequeryresult.InlineQueryResult
Represents a Game.

type
str – ‘game’.

id
str – Unique identifier for this result, 1-64 bytes.

game_short_name
str – Short name of the game.

reply_markup
telegram.InlineKeyboardMarkup – Optional. Inline keyboard attached to the message.

Parameters

- **id**(str) – Unique identifier for this result, 1-64 bytes.
- **game_short_name**(str) – Short name of the game.
• reply_markup (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.
• **kwargs (dict) – Arbitrary keyword arguments.

class telegram.InputContactMessageContent (phone_number, first_name, last_name=None, vcard=None, **kwargs)  
Bases: telegram.inline.inputmessagecontent.InputMessageContent

Represents the content of a contact message to be sent as the result of an inline query.

phone_number  
str – Contact’s phone number.

first_name  
str – Contact’s first name.

last_name  
str – Optional. Contact’s last name.

vcard  
str – Optional. Additional data about the contact in the form of a vCard, 0-2048 bytes.

Parameters

• phone_number (str) – Contact’s phone number.
• first_name (str) – Contact’s first name.
• last_name (str, optional) – Contact’s last name.
• vcard (str, optional) – Additional data about the contact in the form of a vCard, 0-2048 bytes.
• **kwargs (dict) – Arbitrary keyword arguments.

class telegram.InputFile (obj, filename=None, attach=None)  
Bases: object

This object represents a Telegram InputFile.

input_file_content  
bytes – The binary content of the file to send.

filename  
str – Optional. Filename for the file to be sent.

attach  
str – Optional, attach id for sending multiple files.

Parameters

• obj (File handler) – An open file descriptor.
• filename (str, optional) – Filename for this InputFile.
• attach (bool, optional) – Whether this should be send as one file or is part of a collection of files.

Raises
TelegramError

field_tuple

static is_file (obj)

static is_image (stream)

Check if the content file is an image by analyzing its headers.

Parameters stream (str) – A str representing the content of a file.

Returns The str mime-type of an image.
class telegram.InputLocationMessageContent(latitude, longitude, live_period=None, **kwargs)

Represents the content of a location message to be sent as the result of an inline query.

latitude

float – Latitude of the location in degrees.

longitude

float – Longitude of the location in degrees.

Parameters

- latitude (float) – Latitude of the location in degrees.
- longitude (float) – Longitude of the location in degrees.
- live_period (int, optional) – Period in seconds for which the location can be updated, should be between 60 and 86400.
- **kwargs (dict) – Arbitrary keyword arguments.

class telegram.InputMessageContent

Bases: telegram.base.TelegramObject

Base class for Telegram InputMessageContent Objects.


class telegram.InputTextMessageContent(message_text, parse_mode=None, disable_web_page_preview=None, **kwargs)

Represents the content of a text message to be sent as the result of an inline query.

message_text

str – Text of the message to be sent, 1-4096 characters.

parse_mode

str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot’s message.

disable_web_page_preview

bool – Optional. Disables link previews for links in the sent message.

Parameters

- message_text (str) – Text of the message to be sent, 1-4096 characters. Also found as telegram.constants.MAX_MESSAGE_LENGTH.
- parse_mode (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot’s message.
- disable_web_page_preview (bool, optional) – Disables link previews for links in the sent message.
- **kwargs (dict) – Arbitrary keyword arguments.

class telegram.InputVenueMessageContent(latitude, longitude, title, address, foursquare_id=None, foursquare_type=None, **kwargs)

Bases: telegram.inline.inputmessagecontent.InputMessageContent

For more details, see: telegram.InputContactMessageContent, telegram.InputLocationMessageContent, telegram.InputTextMessageContent and telegram.InputVenueMessageContent.
Represents the content of a venue message to be sent as the result of an inline query.

**latitude**
float – Latitude of the location in degrees.

**longitude**
float – Longitude of the location in degrees.

**title**
str – Name of the venue.

**address**
str – Address of the venue.

**foursquare_id**
str – Optional. Foursquare identifier of the venue, if known.

**foursquare_type**
str – Optional. Foursquare type of the venue, if known. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”.)

**Parameters**

- **latitude** (float) – Latitude of the location in degrees.
- **longitude** (float) – Longitude of the location in degrees.
- **title** (str) – Name of the venue.
- **address** (str) – Address of the venue.
- **foursquare_id** (str, optional) – Foursquare identifier of the venue, if known.
- **foursquare_type** (str, optional) – Foursquare type of the venue, if known. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”.)
- ****kwargs (dict) – Arbitrary keyword arguments.

```python
class telegram.KeyboardButton(text, request_contact=None, request_location=None, **kwargs)
```

This object represents one button of the reply keyboard. For simple text buttons String can be used instead of this object to specify text of the button.

**Note:** Optional fields are mutually exclusive.

**text**
str – Text of the button.

**request_contact**
bool – Optional. If the user’s phone number will be sent.

**request_location**
bool – Optional. If the user’s current location will be sent.

**Parameters**

- **text** (str) – Text of the button. If none of the optional fields are used, it will be sent to the bot as a message when the button is pressed.
- **request_contact** (bool, optional) – If True, the user’s phone number will be sent as a contact when the button is pressed. Available in private chats only.
- **request_location** (bool, optional) – If True, the user’s current location will be sent when the button is pressed. Available in private chats only.
Note: *request_contact* and *request_location* options will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

```python
class telegram.Location(longitude, latitude, **kwargs)
Bases: telegram.base.TelegramObject
```

This object represents a point on the map.

- **longitude**
  - float – Longitude as defined by sender.

- **latitude**
  - float – Latitude as defined by sender.

**Parameters**

- **longitude** (float) – Longitude as defined by sender.
- **latitude** (float) – Latitude as defined by sender.
- ****kwargs (dict) – Arbitrary keyword arguments.

```python
classmethod de_json(data, bot)
```

```python
class telegram.EncryptedCredentials(data, hash, secret, bot=None, **kwargs)
Bases: telegram.base.TelegramObject
```

Contains data required for decrypting and authenticating EncryptedPassportElement. See the Telegram Passport Documentation for a complete description of the data decryption and authentication processes.

- **data**
  - telegram.Credentials or str – Decrypted data with unique user’s nonce, data hashes and secrets used for EncryptedPassportElement decryption and authentication or base64 encrypted data.

- **hash**
  - str – Base64-encoded data hash for data authentication.

- **secret**
  - str – Decrypted or encrypted secret used for decryption.

**Parameters**

- **data** (telegram.Credentials or str) – Decrypted data with unique user’s nonce, data hashes and secrets used for EncryptedPassportElement decryption and authentication or base64 encrypted data.
- **hash** (str) – Base64-encoded data hash for data authentication.
- **secret** (str) – Decrypted or encrypted secret used for decryption.
- ****kwargs (dict) – Arbitrary keyword arguments.

Note: This object is decrypted only when originating from *telegram.PassportData.decrypted_credentials*.

```python
classmethod de_json(data, bot)
```

```python
decrypted_data
telegram.Credentials –
```

Lazily decrypt and return credentials data. This object also contains the user specified nonce as *decrypted_data.nonce*.  

Note: This object is decrypted only when originating from *telegram.PassportData.decrypted_credentials*.

---

254 Chapter 1. telegram package
Raises telegram.TelegramDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

decrypted_secret
str – Lazily decrypt and return secret.

Raises telegram.TelegramDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

class telegram.PassportFile (file_id, file_date, file_size=None, bot=None, credentials=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents a file uploaded to Telegram Passport. Currently all Telegram Passport files are in JPEG format when decrypted and don’t exceed 10MB.

file_id
str – Unique identifier for this file.

file_size
int – File size.

file_date
int – Unix time when the file was uploaded.

bot

Parameters

• file_id (str) – Unique identifier for this file.
• file_size (int) – File size.
• file_date (int) – Unix time when the file was uploaded.
• bot (telegram.Bot, optional) – The Bot to use for instance methods.
• **kwargs (dict) – Arbitrary keyword arguments.

classmethod de_json (data, bot)
classmethod de_json_decrypted (data, bot, credentials)
classmethod de_list (data, bot)
classmethod de_list_decrypted (data, bot, credentials)

get_file (timeout=None, **kwargs)


Parameters

• timeout (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
• **kwargs (dict) – Arbitrary keyword arguments.

Returns telegram.File

Raises telegram.TelegramError
class telegram.EncryptedPassportElement(type, data=None, phone_number=None, email=None, files=None, front_side=None, reverse_side=None, selfie=None, translation=None, hash=None, bot=None, credentials=None, **kwargs)

Bases: telegram.base.TelegramObject

Contains information about documents or other Telegram Passport elements shared with the bot by the user. The data has been automatically decrypted by python-telegram-bot.

**type**


**data**

telegram.PersonalDetails or telegram.IdDocument or telegram.ResidentialAddress or str – Optional. Decrypted or encrypted data, available for “personal_details”, “passport”, “driver_license”, “identity_card”, “identity_passport” and “address” types.

**phone_number**

str – Optional. User’s verified phone number, available only for “phone_number” type.

**email**

str – Optional. User’s verified email address, available only for “email” type.

**files**


**front_side**

telegram.PassportFile – Optional. Encrypted/decrypted file with the front side of the document, provided by the user. Available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

**reverse_side**

telegram.PassportFile – Optional. Encrypted/decrypted file with the reverse side of the document, provided by the user. Available for “driver_license” and “identity_card”.

**selfie**

telegram.PassportFile – Optional. Encrypted/decrypted file with the selfie of the user holding a document, provided by the user; available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

**translation**


**hash**

str – Base64-encoded element hash for using in telegram.PassportElementErrorUnspecified.

**bot**


**Parameters**


256 Chapter 1. telegram package
• **data** ([`telegram.PersonalDetails` or `telegram.IdDocument` or `telegram.ResidentialAddress` or `str`, optional]) – Decrypted or encrypted data, available for “personal_details”, “passport”, “driver_license”, “identity_card”, “identity_passport” and “address” types.

• **phone_number** (`str`, optional) – User’s verified phone number, available only for “phone_number” type.

• **email** (`str`, optional) – User’s verified email address, available only for “email” type.

• **files** ([`List[telegram.PassportFile]`, optional]) – Array of encrypted/decrypted files with documents provided by the user, available for “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.

• **front_side** ([`telegram.PassportFile`], optional) – Encrypted/decrypted file with the front side of the document, provided by the user. Available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

• **reverse_side** ([`telegram.PassportFile`], optional) – Encrypted/decrypted file with the reverse side of the document, provided by the user. Available for “driver_license” and “identity_card”.

• **selfie** ([`telegram.PassportFile`], optional) – Encrypted/decrypted file with the selfie of the user holding a document, provided by the user; available for “passport”, “driver_license”, “identity_card” and “internal_passport”.


• **hash** (`str`) – Base64-encoded element hash for using in `telegram.PassportElementErrorUnspecified`.

• **bot** ([`telegram.Bot`, optional]) – The Bot to use for instance methods.

• ****kwargs** (`dict`) – Arbitrary keyword arguments.

---

**Note:** This object is decrypted only when originating from `telegram.PassportData.decrypted_data`.

```python
classmethod de_json(data, bot)
```

```python
classmethod de_json_decrypted(data, bot, credentials)
```

```python
classmethod de_list(data, bot)
```

```python
to_dict()
```

```python
class telegram.PassportData(data, credentials, bot=None, **kwargs)
```

Bases: `telegram.base.TelegramObject`

Contains information about Telegram Passport data shared with the bot by the user.

```python
data
  List[telegram.EncryptedPassportElement] – Array with encrypted information about documents and other Telegram Passport elements that was shared with the bot.
```

```python
credentials
  telegram.EncryptedCredentials – Encrypted credentials.
```

```python
bot
```
Parameters

- **data** (`List[telegram.EncryptedPassportElement]`) – Array with encrypted information about documents and other Telegram Passport elements that was shared with the bot.
- **credentials** (`str`) – Encrypted credentials.
- **bot** (`telegram.Bot`, optional) – The Bot to use for instance methods.
- **kwargs** (`dict`) – Arbitrary keyword arguments.

Note: To be able to decrypt this object, you must pass your private_key to either `telegram.Updater` or `telegram.Bot`. Decrypted data is then found in `decrypted_data` and the payload can be found in `decrypted_credentials`'s attribute `telegram.Credentials.payload`.

```python
classmethod de_json(data, bot)

decrypted_credentials

Lazily decrypt and return credentials that were used to decrypt the data. This object also contains the user specified payload as `decrypted_data.payload`.

Raises `telegram.TelegramDecryptionError` – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

decrypted_data

Lazily decrypt and return information about documents and other Telegram Passport elements which were shared with the bot.

Raises `telegram.TelegramDecryptionError` – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

to_dict()
```

```python
class telegram.Message

Bases: telegram.base.TelegramObject

This object represents a message.

Note:

- In Python `from` is a reserved word, use `from_user` instead.
```
message_id
int – Unique message identifier inside this chat.

from_user

date
datetime.datetime – Date the message was sent.

chat
telegram.Chat – Conversation the message belongs to.

forward_from
telegram.User – Optional. Sender of the original message.

forward_from_chat
telegram.Chat – Optional. Information about the original channel.

forward_from_message_id
int – Optional. Identifier of the original message in the channel.

forward_date
datetime.datetime – Optional. Date the original message was sent.

reply_to_message
telegram.Message – Optional. The original message.

edit_date
datetime.datetime – Optional. Date the message was last edited.

media_group_id
str – Optional. The unique identifier of a media message group this message belongs to.

text
str – Optional. The actual UTF-8 text of the message.

entities
List[telegram.MessageEntity] – Optional. Special entities like usernames, URLs, bot commands, etc. that appear in the text. See Message.parse_entity and parse_entities methods for how to use properly.

caption_entities
List[telegram.MessageEntity] – Optional. Special entities like usernames, URLs, bot commands, etc. that appear in the caption. See Message.parse_caption_entity and parse_caption_entities methods for how to use properly.

audio

document

animation
telegram.Animation – For backward compatibility, when this field is set, the document field will also be set.

game
telegram.Game – Optional. Information about the game.

photo

sticker
telegram.Sticker – Optional. Information about the sticker.

video
telegram.Video – Optional. Information about the video.
voice


video_note

telegram.VideoNote – Optional. Information about the video message.

caption

str – Optional. Caption for the document, photo or video, 0-1024 characters.

caption

str – Optional. Caption for the document, photo or video, 0-1024 characters.

location

telegram.Location – Optional. Information about the location.

venue

telegram.Venue – Optional. Information about the venue.

left_chat_member

telegram.User – Optional. Information about the user that left the group. (this member may be the bot itself).

new_chat_title

str – Optional. A chat title was changed to this value.

new_chat_photo

List[telegram.PhotoSize] – Optional. A chat photo was changed to this value.

delete_chat_photo

bool – Optional. The chat photo was deleted.

group_chat_created

bool – Optional. The group has been created.

supergroup_chat_created

bool – Optional. The supergroup has been created.

channel_chat_created

bool – Optional. The channel has been created.

migrate_to_chat_id

int – Optional. The group has been migrated to a supergroup with the specified identifier.

migrate_from_chat_id

int – Optional. The supergroup has been migrated from a group with the specified identifier.

pinned_message

telegram.message – Optional. Specified message was pinned.

invoice

telegram.Invoice – Optional. Information about the invoice.

successful_payment

telegram.SuccessfulPayment – Optional. Information about the payment.

connected_website

str – Optional. The domain name of the website on which the user has logged in.

forward_signature

str – Optional. Signature of the post author for messages forwarded from channels.

author_signature

str – Optional. Signature of the post author for messages in channels.
passport_data
telegram.PassportData – Optional. Telegram Passport data

bot

Parameters

• message_id (int) – Unique message identifier inside this chat.

• from_user (telegram.User, optional) – Sender, can be empty for messages sent to channels.

• date (datetime.datetime) – Date the message was sent in Unix time. Converted to datetime.datetime.

• chat (telegram.Chat) – Conversation the message belongs to.

• forward_from (telegram.User, optional) – For forwarded messages, sender of the original message.

• forward_from_chat (telegram.Chat, optional) – For messages forwarded from a channel, information about the original channel.

• forward_from_message_id (int, optional) – For forwarded channel posts, identifier of the original message in the channel.

• forward_date (datetime.datetime, optional) – For forwarded messages, date the original message was sent in Unix time. Converted to datetime.datetime.

• reply_to_message (telegram.Message, optional) – For replies, the original message. Note that the Message object in this field will not contain further reply_to_message fields even if it itself is a reply.

• edit_date (datetime.datetime, optional) – Date the message was last edited in Unix time. Converted to datetime.datetime.

• media_group_id (str, optional) – The unique identifier of a media message group this message belongs to.

• text (str, optional) – For text messages, the actual UTF-8 text of the message, 0-4096 characters. Also found as telegram.constants.MAX_MESSAGE_LENGTH.

• entities (List[telegram.MessageEntity], optional) – For text messages, special entities like usernames, URLs, bot commands, etc. that appear in the text. See attr:parse_entity and attr:parse_entities methods for how to use properly.

• caption_entities (List[telegram.MessageEntity]) – Optional. For Messages with a Caption. Special entities like usernames, URLs, bot commands, etc. that appear in the caption. See Message.parse_caption_entity and parse_caption_entities methods for how to use properly.

• audio (telegram.Audio, optional) – Message is an audio file, information about the file.

• document (telegram.Document, optional) – Message is a general file, information about the file.

• animation (telegram.Animation, optional) – Message is an animation, information about the animation. For backward compatibility, when this field is set, the document field will also be set.

• game (telegram.Game, optional) – Message is a game, information about the game.

• photo (List[telegram.PhotoSize], optional) – Message is a photo, available sizes of the photo.
• **sticker** (*telegram.Sticker*, optional) – Message is a sticker, information about the sticker.

• **video** (*telegram.Video*, optional) – Message is a video, information about the video.

• **voice** (*telegram.Voice*, optional) – Message is a voice message, information about the file.

• **video_note** (*telegram.VideoNote*, optional) – Message is a video note, information about the video message.

• **new_chat_members** (List[*telegram.User*], optional) – New members that were added to the group or supergroup and information about them (the bot itself may be one of these members).

• **caption** (*str*, optional) – Caption for the document, photo or video, 0-1024 characters.

• **contact** (*telegram.Contact*, optional) – Message is a shared contact, information about the contact.

• **location** (*telegram.Location*, optional) – Message is a shared location, information about the location.

• **venue** (*telegram.Venue*, optional) – Message is a venue, information about the venue.

• **left_chat_member** (*telegram.User*, optional) – A member was removed from the group, information about them (this member may be the bot itself).

• **new_chat_title** (*str*, optional) – A chat title was changed to this value.

• **new_chat_photo** (List[*telegram.PhotoSize*], optional) – A chat photo was change to this value.

• **delete_chat_photo** (*bool*, optional) – Service message: The chat photo was deleted.

• **group_chat_created** (*bool*, optional) – Service message: The group has been created.

• **supergroup_chat_created** (*bool*, optional) – Service message: The supergroup has been created. This field can’t be received in a message coming through updates, because bot can’t be a member of a supergroup when it is created. It can only be found in `reply_to_message` if someone replies to a very first message in a directly created supergroup.

• **channel_chat_created** (*bool*, optional) – Service message: The channel has been created. This field can’t be received in a message coming through updates, because bot can’t be a member of a channel when it is created. It can only be found in `reply_to_message` if someone replies to a very first message in a channel.

• **migrate_to_chat_id** (*int*, optional) – The group has been migrated to a supergroup with the specified identifier. This number may be greater than 32 bits and some programming languages may have difficulty/silent defects in interpreting it. But it is smaller than 52 bits, so a signed 64 bit integer or double-precision float type are safe for storing this identifier.

• **migrate_from_chat_id** (*int*, optional) – The supergroup has been migrated from a group with the specified identifier. This number may be greater than 32 bits and some programming languages may have difficulty/silent defects in interpreting it. But it is smaller than 52 bits, so a signed 64 bit integer or double-precision float type are safe for storing this identifier.
• *pinned_message* (telegram.message, optional) – Specified message was pinned. Note that the Message object in this field will not contain further attr:*reply_to_message* fields even if it is itself a reply.

• *invoice* (telegram.Invoice, optional) – Message is an invoice for a payment, information about the invoice.

• *successful_payment* (telegram.SuccessfulPayment, optional) – Message is a service message about a successful payment, information about the payment.

• *connected_website* (str, optional) – The domain name of the website on which the user has logged in.

• *forward_signature* (str, optional) – Signature of the post author for messages forwarded from channels.

• *author_signature* (str, optional) – Signature of the post author for messages in channels.

• *passport_data* (telegram.PassportData, optional) – Telegram Passport data

```python
ATTACHMENT_TYPES = ['audio', 'game', 'animation', 'document', 'photo', 'sticker', 'video', 'voice', 'video_note', 'contact', 'location', 'venue', 'invoice', 'successful_payment']
MESSAGE_TYPES = ['text', 'new_chat_members', 'left_chat_member', 'new_chat_title', 'new_chat_photo', ...
```

- **caption_html**
  Creates an HTML-formatted string from the markup entities found in the message’s caption.

  Use this if you want to retrieve the message caption with the caption entities formatted as HTML in the same way the original message was formatted.

  Returns  Message caption with caption entities formatted as HTML.

  Return type  str

- **caption_html_urlled**
  Creates an HTML-formatted string from the markup entities found in the message’s caption.

  Use this if you want to retrieve the message caption with the caption entities formatted as HTML. This also formats telegram.MessageEntity.URL as a hyperlink.

  Returns  Message caption with caption entities formatted as HTML.

  Return type  str

- **caption_markdown**
  Creates a Markdown-formatted string from the markup entities found in the message’s caption.

  Use this if you want to retrieve the message caption with the caption entities formatted as Markdown in the same way the original message was formatted.

  Returns  Message caption with caption entities formatted as Markdown.

  Return type  str

- **caption_markdown_urlled**
  Creates a Markdown-formatted string from the markup entities found in the message’s caption.

  Use this if you want to retrieve the message caption with the caption entities formatted as Markdown. This also formats telegram.MessageEntity.URL as a hyperlink.

  Returns  Message caption with caption entities formatted as Markdown.

  Return type  str

- **chat_id**

- **classmethod de_json**(data, bot)

1.49. Module contents 263
**delete** (*args, **kwargs*)

Shortcut for:

```
bot.delete_message(chat_id=message.chat_id,
                   message_id=message.message_id,
                   *args,
                   **kwargs)
```

**Returns** On success, True is returned.

**Return type** bool

**edit_caption** (*args, **kwargs*)

Shortcut for:

```
bot.edit_message_caption(chat_id=message.chat_id,
                         message_id=message.message_id,
                         *args,
                         **kwargs)
```

**Note:** You can only edit messages that the bot sent itself, therefore this method can only be used on the return value of the bot.send_* family of methods.

**Returns** On success, instance representing the edited message.

**Return type** telegram.Message

**edit_media** (media, *args, **kwargs*)

Shortcut for:

```
bot.edit_message_media(chat_id=message.chat_id,
                       message_id=message.message_id,
                       *args,
                       **kwargs)
```

**Note:** You can only edit messages that the bot sent itself, therefore this method can only be used on the return value of the bot.send_* family of methods.

**Returns** On success, instance representing the edited message.

**Return type** telegram.Message

**edit_reply_markup** (*args, **kwargs*)

Shortcut for:

```
bot.edit_message_reply_markup(chat_id=message.chat_id,
                              message_id=message.message_id,
                              *args,
                              **kwargs)
```

**Note:** You can only edit messages that the bot sent itself, therefore this method can only be used on the return value of the bot.send_* family of methods.

**Returns** On success, instance representing the edited message.

**Return type** telegram.Message
edit_text (*args, **kwargs)
Shortcut for:

```python
bot.edit_message_text(chat_id=message.chat_id,
message_id=message.message_id,
*args,
**kwargs)
```

**Note:** You can only edit messages that the bot sent itself, therefore this method can only be used on
the return value of the `bot.send_` family of methods.

**Returns** On success, instance representing the edited message.

**Return type** telegram.Message

effective_attachment
telegram.Audio or telegram.Contact or telegram.Document or telegram.Animation
or telegram.Game or telegram.Invoice or telegram.Location or List[telegram.PhotoSize]
or telegram.Sticker or telegram.SuccessfulPayment or telegram.Venue or telegram.Video
or telegram.VideoNote or telegram.Voice: The attachment that this message was sent with. May be
None if no attachment was sent.

forward (chat_id, disable_notification=False)
Shortcut for:

```python
bot.forward_message(chat_id=chat_id,
from_chat_id=update.message.chat_id,
disable_notification=disable_notification,
message_id=update.message.message_id)
```

**Returns** On success, instance representing the message forwarded.

**Return type** telegram.Message

link
str – Convenience property. If the chat of the message is a supergroup or a channel and has a `Chat`
username, returns a `t.me` link of the message.

parse_caption_entities (types=None)
Returns a dict that maps `telegram.MessageEntity` to str. It contains entities from this
message’s caption filtered by their `telegram.MessageEntity.type` attribute as the key, and
the text that each entity belongs to as the value of the dict.

**Note:** This method should always be used instead of the `caption_entities` attribute,
since it calculates the correct substring from the message text based on UTF-16 codepoints. See
`parse_entity` for more info.

**Parameters** types (List[str], optional) – List of `telegram.MessageEntity` types
as strings. If the `type` attribute of an entity is contained in this list, it will be returned.
Defaults to a list of all types. All types can be found as constants in `telegram.MessageEntity`.

**Returns** A dictionary of entities mapped to the text that belongs to them, calculated based
on UTF-16 codepoints.

**Return type** Dict[telegram.MessageEntity, str]

parse_caption_entity (entity)
Returns the text from a given `telegram.MessageEntity`.

1.49. Module contents 265
Note: This method is present because Telegram calculates the offset and length in UTF-16 codepoint pairs, which some versions of Python don’t handle automatically. (That is, you can’t just slice Message.caption with the offset and length.)

Parameters

- **entity** (*telegram.MessageEntity*) – The entity to extract the text from. It must

  * an entity that belongs to this message. *(be)*

Returns The text of the given entity

Return type str

```
parse_entities(types=None)
```

Returns a dict that maps *telegram.MessageEntity* to str. It contains entities from this message filtered by their *telegram.MessageEntity.type* attribute as the key, and the text that each entity belongs to as the value of the dict.

Note: This method should always be used instead of the `entities` attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See *parse_entity* for more info.

Parameters

- **types** (List[*str*], optional) – List of *telegram.MessageEntity* types as strings. If the type attribute of an entity is contained in this list, it will be returned. Defaults to a list of all types. All types can be found as constants in *telegram.MessageEntity*.

Returns A dictionary of entities mapped to the text that belongs to them, calculated based on UTF-16 codepoints.

Return type Dict[*telegram.MessageEntity, str*]

```
prefix_entity(entity)
```

Returns the text from a given *telegram.MessageEntity*.

Note: This method is present because Telegram calculates the offset and length in UTF-16 codepoint pairs, which some versions of Python don’t handle automatically. (That is, you can’t just slice Message.text with the offset and length.)

Parameters

- **entity** (*telegram.MessageEntity*) – The entity to extract the text from. It must

  * an entity that belongs to this message. *(be)*

Returns The text of the given entity

Return type str

```
reply_animation(*args, **kwargs)
```

Shortcut for:

```
bot.send_animation(update.message.chat_id, *args, **kwargs)
```
Keyword Arguments `quote` (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

```python
reply_audio(*args, **kwargs)
```
Shortcut for:
```python
def reply_audio(update: Update, *args, **kwargs):
    bot.send_audio(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments `quote` (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

```python
reply_contact(*args, **kwargs)
```
Shortcut for:
```python
def reply_contact(update: Update, *args, **kwargs):
    bot.send_contact(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments `quote` (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

```python
reply_document(*args, **kwargs)
```
Shortcut for:
```python
def reply_document(update: Update, *args, **kwargs):
    bot.send_document(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments `quote` (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

```python
reply_html(*args, **kwargs)
```
Shortcut for:
```python
def reply_html(update: Update, *args, **kwargs):
    bot.send_message(update.message.chat_id, parse_mode=ParseMode.HTML, *args, **kwargs)
```

Sends a message with HTML formatting.

Keyword Arguments `quote` (bool, optional) – If set to True, the message is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

```python
reply_location(*args, **kwargs)
```
Shortcut for:
```python
def reply_location(update: Update, *args, **kwargs):
    bot.send_location(update.message.chat_id, *args, **kwargs)
```
Keyword Arguments quote (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type telegram.Message

reply_markdown (*args, **kwargs)
Shortcut for:
```
bot.send_message(update.message.chat_id, parse_mode=ParseMode.MARKDOWN, ...
+*args,
+**kwargs)
```

Sends a message with markdown formatting.

Keyword Arguments quote (bool, optional) – If set to True, the message is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

reply_media_group (*args, **kwargs)
Shortcut for:
```
bot.reply_media_group(update.message.chat_id, +args, +**kwargs)
```

Keyword Arguments quote (bool, optional) – If set to True, the media group is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns An array of the sent Messages.

Return type List[telegram.Message]

Raises telegram.TelegramError

reply_photo (*args, **kwargs)
Shortcut for:
```
bot.send_photo(update.message.chat_id, +args, +**kwargs)
```

Keyword Arguments quote (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type telegram.Message

reply_sticker (*args, **kwargs)
Shortcut for:
```
bot.send_sticker(update.message.chat_id, +args, +**kwargs)
```

Keyword Arguments quote (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type telegram.Message

reply_text (*args, **kwargs)
Shortcut for:
bot.send_message(update.message.chat_id, *args, **kwargs)

**Keyword Arguments**

*quote* *(bool, optional)* – If set to True, the message is sent as an actual reply to this message. If *reply_to_message_id* is passed in *kwargs*, this parameter will be ignored. Default: True in group chats and False in private chats.

**reply_venue** (*args, **kwargs*)
Shortcut for:

bot.send_venue(update.message.chat_id, *args, **kwargs)

**Keyword Arguments**

*quote* *(bool, optional)* – If set to True, the photo is sent as an actual reply to this message. If *reply_to_message_id* is passed in *kwargs*, this parameter will be ignored. Default: True in group chats and False in private chats.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

**reply_video** (*args, **kwargs*)
Shortcut for:

bot.send_video(update.message.chat_id, *args, **kwargs)

**Keyword Arguments**

*quote* *(bool, optional)* – If set to True, the photo is sent as an actual reply to this message. If *reply_to_message_id* is passed in *kwargs*, this parameter will be ignored. Default: True in group chats and False in private chats.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

**reply_video_note** (*args, **kwargs*)
Shortcut for:

bot.send_video_note(update.message.chat_id, *args, **kwargs)

**Keyword Arguments**

*quote* *(bool, optional)* – If set to True, the photo is sent as an actual reply to this message. If *reply_to_message_id* is passed in *kwargs*, this parameter will be ignored. Default: True in group chats and False in private chats.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

**reply_voice** (*args, **kwargs*)
Shortcut for:

bot.send_voice(update.message.chat_id, *args, **kwargs)

**Keyword Arguments**

*quote* *(bool, optional)* – If set to True, the photo is sent as an actual reply to this message. If *reply_to_message_id* is passed in *kwargs*, this parameter will be ignored. Default: True in group chats and False in private chats.

**Returns** On success, instance representing the message posted.

**Return type** `telegram.Message`

text_html

Creates an HTML-formatted string from the markup entities found in the message.

Use this if you want to retrieve the message text with the entities formatted as HTML in the same way the original message was formatted.
Returns    Message text with entities formatted as HTML.
Return type  str

text_html_urled
Creates an HTML-formatted string from the markup entities found in the message.
Use this if you want to retrieve the message text with the entities formatted as HTML. This also formats
telegram.MessageEntity.URL as a hyperlink.
Returns    Message text with entities formatted as HTML.
Return type  str

text_markdown
Creates a Markdown-formatted string from the markup entities found in the message.
Use this if you want to retrieve the message text with the entities formatted as Markdown in the same
way the original message was formatted.
Returns    Message text with entities formatted as Markdown.
Return type  str

text_markdown_urled
Creates a Markdown-formatted string from the markup entities found in the message.
Use this if you want to retrieve the message text with the entities formatted as Markdown. This also
formats telegram.MessageEntity.URL as a hyperlink.
Returns    Message text with entities formatted as Markdown.
Return type  str

to_dict()
ALL_TYPES = ['mention', 'hashtag', 'cashtag', 'phone_number', 'bot_command', 'url',
List[str]] – List of all the types.

BOLD = 'bold'
str – 'bold'

BOT_COMMAND = 'bot_command'
str – 'bot_command'

CASHTAG = 'cashtag'
str – 'cashtag'

CODE = 'code'
str – 'code'

EMAIL = 'email'
str – 'email'

HASHTAG = 'hashtag'
str – 'hashtag'

ITALIC = 'italic'
str – 'italic'

MENTION = 'mention'
str – 'mention'

PHONE_NUMBER = 'phone_number'
str – 'phone_number'

PRE = 'pre'
str – 'pre'

TEXT_LINK = 'text_link'
str – 'text_link'

TEXT_MENTION = 'text_mention'
str – 'text_mention'

URL = 'url'
str – 'url'

classmethod de_json(data, bot)
classmethod de_list(data, bot)

class telegram.ParseMode
Bases: object

This object represents a Telegram Message Parse Modes.

HTML = 'HTML'
str – 'HTML'

MARKDOWN = 'Markdown'
str – 'Markdown’

class telegram.PhotoSize(file_id, width, height, file_size=None, bot=None, **kwargs)
Bases: telegram.base.TelegramObject

This object represents one size of a photo or a file/sticker thumbnail.

file_id
str – Unique identifier for this file.

width
int – Photo width.

height
int – Photo height.
file_size
    int – Optional. File size.

bot

Parameters
- **file_id** (str) – Unique identifier for this file.
- **width** (int) – Photo width.
- **height** (int) – Photo height.
- **file_size** (int, optional) – File size.
- **bot** (telegram.Bot, optional) – The Bot to use for instance methods.
- ****kwargs (dict) – Arbitrary keyword arguments.

classmethod de_json(data, bot)

classmethod de_list(data, bot)

get_file(timeout=None, **kwargs)
    Convenience wrapper over telegram.Bot.get_file

Parameters
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- ****kwargs (dict) – Arbitrary keyword arguments.

Returns
    telegram.File

Raises
    telegram.TelegramError

class telegram.ReplyKeyboardRemove(selective=False, **kwargs)
    Bases: telegram.replymarkup.ReplyMarkup

Upon receiving a message with this object, Telegram clients will remove the current custom keyboard and display the default letter-keyboard. By default, custom keyboards are displayed until a new keyboard is sent by a bot. An exception is made for one-time keyboards that are hidden immediately after the user presses a button (see telegram.ReplyKeyboardMarkup).

remove_keyboard
    True – Requests clients to remove the custom keyboard.

selective
    bool – Optional. Use this parameter if you want to remove the keyboard for specific users only.

Example

A user votes in a poll, bot returns confirmation message in reply to the vote and removes the keyboard for that user, while still showing the keyboard with poll options to users who haven’t voted yet.

Parameters
- **selective** (bool, optional) – Use this parameter if you want to remove the keyboard for specific users only. Targets:
  1. users that are @mentioned in the text of the Message object
  2. if the bot’s message is a reply (has reply_to_message_id), sender of the original message.
- ****kwargs (dict) – Arbitrary keyword arguments.
class telegram.ReplyKeyboardMarkup:

    resize_keyboard = bool – Optional. Requests clients to resize the keyboard.
    one_time_keyboard = bool – Optional. Requests clients to hide the keyboard as soon as it’s been used.
    selective = bool – Optional. Show the keyboard to specific users only.

Example

A user requests to change the bot’s language, bot replies to the request with a keyboard to select the new language. Other users in the group don’t see the keyboard.

Parameters

    • keyboard (List[List[str | telegram.KeyboardButton]]) – Array of button rows, each represented by an Array of telegram.KeyboardButton objects.
    • resize_keyboard (bool, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to False, in which case the custom keyboard is always of the same height as the app’s standard keyboard. Defaults to False.
    • one_time_keyboard (bool, optional) – Requests clients to hide the keyboard as soon as it’s been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to False.
    • selective (bool, optional) – Use this parameter if you want to show the keyboard to specific users only. Targets:
      1. users that are @mentioned in the text of the Message object
      2. if the bot’s message is a reply (has reply_to_message_id), sender of the original message.
      Defaults to False.
    • **kwargs (dict) – Arbitrary keyword arguments.

classmethod from_button:

    button = telegram.KeyboardButton | str – The button to use in the markup
    resize_keyboard = bool, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to False, in which case the custom keyboard is always of the same height as the app’s standard keyboard. Defaults to False.
• **one_time_keyboard** (bool, optional) – Requests clients to hide the keyboard as soon as it’s been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to False.

• **selective** (bool, optional) – Use this parameter if you want to show the keyboard to specific users only. Targets:
  1. users that are @mentioned in the text of the Message object
  2. if the bot’s message is a reply (has reply_to_message_id), sender of the original message.

  Defaults to False.

• ****kwargs (dict) – Arbitrary keyword arguments.

```
classmethod from_column(button_column, resize_keyboard=False, 
                        one_time_keyboard=False, selective=False, **kwargs)
```

Shortcut for:

```
ReplyKeyboardMarkup([[button] for button in button_column], **kwargs)
```

Return an ReplyKeyboardMarkup from a single column of KeyboardButtons

**Parameters**

• **button_column** (List[telegram.KeyboardButton | str]) – The button to use in the markup

• **resize_keyboard** (bool, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to false, in which case the custom keyboard is always of the same height as the app’s standard keyboard. Defaults to False

• **one_time_keyboard** (bool, optional) – Requests clients to hide the keyboard as soon as it’s been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to False.

• **selective** (bool, optional) – Use this parameter if you want to show the keyboard to specific users only. Targets:
  1. users that are @mentioned in the text of the Message object
  2. if the bot’s message is a reply (has reply_to_message_id), sender of the original message.

  Defaults to False.

• ****kwargs (dict) – Arbitrary keyword arguments.

```
classmethod from_row(button_row, resize_keyboard=False, one_time_keyboard=False, selective=False, **kwargs)
```

Shortcut for:

```
ReplyKeyboardMarkup([button_row], **kwargs)
```

Return an ReplyKeyboardMarkup from a single row of KeyboardButtons

**Parameters**

• **button_row** (List[telegram.KeyboardButton | str]) – The button to use in the markup

• **resize_keyboard** (bool, optional) – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to false, in which case the custom keyboard is always of the same height as the app’s standard keyboard. Defaults to False
one_time_keyboard (bool, optional) – Requests clients to hide the keyboard as soon as it’s been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to False.

selective (bool, optional) – Use this parameter if you want to show the keyboard to specific users only. Targets:
1. users that are @mentioned in the text of the Message object
2. if the bot’s message is a reply (has reply_to_message_id), sender of the original message.

Defaults to False.

**kwargs (dict) – Arbitrary keyword arguments.

to_dict ()

class telegram.ReplyMarkup
    Bases: telegram.base.TelegramObject
    Base class for Telegram ReplyMarkup Objects.


class telegram.Sticker (file_id, width, height, thumb=None, emoji=None, file_size=None, set_name=None, mask_position=None, bot=None, **kwargs)
    Bases: telegram.base.TelegramObject
    This object represents a sticker.

    file_id
        str – Unique identifier for this file.

    width
        int – Sticker width.

    height
        int – Sticker height.

    thumb
        telegram.PhotoSize – Optional. Sticker thumbnail in the .webp or .jpg format.

    emoji
        str – Optional. Emoji associated with the sticker.

    set_name
        str – Optional. Name of the sticker set to which the sticker belongs.

    mask_position
        telegram.MaskPosition – Optional. For mask stickers, the position where the mask should be placed.

    file_size
        int – Optional. File size.

    bot

    Parameters
        • file_id (str) – Unique identifier for this file.
        • width (int) – Sticker width.
        • height (int) – Sticker height.
• **thumb** (*telegram.PhotoSize*, optional) – Sticker thumbnail in the .webp or .jpg format.
• **emoji** (*str*, optional) – Emoji associated with the sticker
• **set_name** (*str*, optional) – Name of the sticker set to which the sticker belongs.
• **mask_position** (*telegram.MaskPosition*, optional) – For mask stickers, the position where the mask should be placed.
• **file_size** (*int*, optional) – File size.
• (**obj** (**kwargs**)) – Arbitrary keyword arguments.
• **bot** (*telegram.Bot*, optional) – The Bot to use for instance methods.

classmethod **de_json**(data, bot)
classmethod **de_list**(data, bot)

**get_file**(timeout=None, **kwargs**)  
Convenience wrapper over *telegram.Bot.get_file*

Parameters
• **timeout** (*int* | *float*, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
• **kwargs** (*dict*) – Arbitrary keyword arguments.

Returns *telegram.File*

Raisess *telegram.TelegramError*

exception **telegram.TelegramError**(message)
Bases: Exception

class **telegram.TelegramObject**
Bases: *object*

Base class for most telegram objects.

classmethod **de_json**(data, bot)
to_dict()
to_json()

Returns *str*

class **telegram.Update**(update_id, message=None, edited_message=None, channel_post=None, edited_channel_post=None, inline_query=None, chosen_inline_result=None, callback_query=None, pre_checkout_query=None, **kwargs**)
Bases: *telegram.base.TelegramObject*

This object represents an incoming update.

**Note:** At most one of the optional parameters can be present in any given update.

**update_id**
*int* – The update’s unique identifier.

**message**

**edited_message**
channel_post

edited_channel_post

inline_query

chosen_inline_result
  telegram.ChosenInlineResult – Optional. The result of an inline query that was chosen by a user.

callback_query

shipping_query

pre_checkout_query

Parameters

- update_id (int) – The update’s unique identifier. Update identifiers start from a certain positive number and increase sequentially. This ID becomes especially handy if you’re using Webhooks, since it allows you to ignore repeated updates or to restore the correct update sequence, should they get out of order.

- message (telegram.Message, optional) – New incoming message of any kind - text, photo, sticker, etc.

- edited_message (telegram.Message, optional) – New version of a message that is known to the bot and was edited.

- channel_post (telegram.Message, optional) – New incoming channel post of any kind - text, photo, sticker, etc.

- edited_channel_post (telegram.Message, optional) – New version of a channel post that is known to the bot and was edited.


- chosen_inline_result (telegram.ChosenInlineResult, optional) – The result of an inline query that was chosen by a user and sent to their chat partner.


- pre_checkout_query (telegram.PreCheckoutQuery, optional) – New incoming pre-checkout query. Contains full information about checkout

- **kwargs (dict) – Arbitrary keyword arguments.

classmethod de_json (data, bot)

effective_chat
  telegram.Chat – The chat that this update was sent in, no matter what kind of update this is. Will be None for inline_query, chosen_inline_result, callback_query from inline messages, shipping_query and pre_checkout_query.

effective_message
  telegram.Message – The message included in this update, no matter what kind of update this is.
Will be None for inline_query, chosen_inline_result, callback_query from inline messages, shipping_query and pre_checkout_query.

**effective_user**

*telegram.User* – The user that sent this update, no matter what kind of update this is. Will be None for channel_post.

**class telegram.User(id, first_name, is_bot, last_name=None, username=None, language_code=None, bot=None, **kwargs)**

This object represents a Telegram user or bot.

- **id**
  - Unique identifier for this user or bot.

- **is_bot**
  - bool – True, if this user is a bot.

- **first_name**
  - str – User’s or bot’s first name.

- **last_name**
  - str – Optional. User’s or bot’s last name.

- **username**
  - str – Optional. User’s or bot’s username.

- **language_code**
  - str – Optional. IETF language tag of the user’s language.

- **bot**

**Parameters**

- **id**(int) – Unique identifier for this user or bot.
- **is_bot**(bool) – True, if this user is a bot.
- **first_name**(str) – User’s or bot’s first name.
- **last_name**(str, optional) – User’s or bot’s last name.
- **username**(str, optional) – User’s or bot’s username.
- **language_code**(str, optional) – IETF language tag of the user’s language.
- **bot**(telegram.Bot, optional) – The Bot to use for instance methods.

**classmethod de_json**(data, bot)

**classmethod de_list**(data, bot)

**full_name**

str – Convenience property. The user’s first_name, followed by (if available) last_name.

**get_profile_photos**(*args, **kwargs*)

Shortcut for:

```
bot.get_user_profile_photos(update.message.from_user.id, *args, **kwargs)
```

**link**

str – Convenience property. If username is available, returns a t.me link of the user.

**mention_html**(name=None)

**Parameters** name**(str) – The name used as a link for the user. Defaults to full_name.

**Returns** The inline mention for the user as HTML.
Return type  str

**mention_markdown** *(name=None)*

**Parameters**  
- **name** *(str)* – The name used as a link for the user. Defaults to *full_name*.

**Returns**  
The inline mention for the user as markdown.

**name**  

str – Convenience property. If available, returns the user’s *username* prefixed with “@”. If *username* is not available, returns *full_name*.

**send_animation** *(*args, **kwargs)*

Shortcut for:

```
bot.send_animation(User.id, *args, **kwargs)
```

Where User is the current instance.

**Returns**  
On success, instance representing the message posted.

**Return type**  
*telegram.Message*

**send_audio** *(*args, **kwargs)*

Shortcut for:

```
bot.send_audio(User.id, *args, **kwargs)
```

Where User is the current instance.

**Returns**  
On success, instance representing the message posted.

**Return type**  
*telegram.Message*

**send_document** *(*args, **kwargs)*

Shortcut for:

```
bot.send_document(User.id, *args, **kwargs)
```

Where User is the current instance.

**Returns**  
On success, instance representing the message posted.

**Return type**  
*telegram.Message*

**send_message** *(*args, **kwargs)*

Shortcut for:

```
bot.send_message(User.id, *args, **kwargs)
```

Where User is the current instance.

**Returns**  
On success, instance representing the message posted.

**Return type**  
*telegram.Message*

**send_photo** *(*args, **kwargs)*

Shortcut for:

```
bot.send_photo(User.id, *args, **kwargs)
```

Where User is the current instance.

**Returns**  
On success, instance representing the message posted.

**Return type**  
*telegram.Message*
send_sticker(*args, **kwargs)
Shortcut for:

```
bot.send_sticker(User.id, *args, **kwargs)
```

Where User is the current instance.

**Returns** On success, instance representing the message posted.

**Return type** telegram.Message

send_video(*args, **kwargs)
Shortcut for:

```
bot.send_video(User.id, *args, **kwargs)
```

Where User is the current instance.

**Returns** On success, instance representing the message posted.

**Return type** telegram.Message

send_video_note(*args, **kwargs)
Shortcut for:

```
bot.send_video_note(User.id, *args, **kwargs)
```

Where User is the current instance.

**Returns** On success, instance representing the message posted.

**Return type** telegram.Message

send_voice(*args, **kwargs)
Shortcut for:

```
bot.send_voice(User.id, *args, **kwargs)
```

Where User is the current instance.

**Returns** On success, instance representing the message posted.

**Return type** telegram.Message

**class** telegram.UserProfilePhotos(total_count, photos, **kwargs)
**Bases:** telegram.base.TelegramObject

This object represent a user’s profile pictures.

**total_count**
int – Total number of profile pictures.

**photos**

**Parameters**

- **total_count** (int) – Total number of profile pictures the target user has.
- **photos** (List[List[telegram.PhotoSize]]) – Requested profile pictures (in up to 4 sizes each).

**classmethod** de_json(data, bot)

to_dict()
This object represents a venue.

**location**

`telegram.Location` – Venue location.

**title**

`str` – Name of the venue.

**address**

`str` – Address of the venue.

**foursquare_id**

`str` – Optional. Foursquare identifier of the venue.

**foursquare_type**

`str` – Optional. Foursquare type of the venue. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”.)

**Parameters**

- `location (telegram.Location)` – Venue location.
- `title (str)` – Name of the venue.
- `address (str)` – Address of the venue.
- `foursquare_id (str, optional)` – Foursquare identifier of the venue.
- `foursquare_type (str, optional)` – Foursquare type of the venue. (For example, “arts_entertainment/default”, “arts_entertainment/aquarium” or “food/icecream”.)
- `**kwargs (dict)` – Arbitrary keyword arguments.

**classmethod de_json (data, bot)**

**class** `telegram.Video (file_id, width, height, duration, thumb=None, mime_type=None, file_size=None, bot=None, **kwargs)`

This object represents a video file.

**file_id**

`str` – Unique identifier for this file.

**width**

`int` – Video width as defined by sender.

**height**

`int` – Video height as defined by sender.

**duration**

`int` – Duration of the video in seconds as defined by sender.

**thumb**


**mime_type**

`str` – Optional. Mime type of a file as defined by sender.

**file_size**

`int` – Optional. File size.

**bot**


**Parameters**

- `file_id (str)` – Unique identifier for this file.
- `width (int)` – Video width as defined by sender.
• **height** (int) – Video height as defined by sender.
• **duration** (int) – Duration of the video in seconds as defined by sender.
• **thumb** (telegram.PhotoSize, optional) – Video thumbnail.
• **mime_type** (str, optional) – Mime type of a file as defined by sender.
• **file_size** (int, optional) – File size.
• **bot** (telegram.Bot, optional) – The Bot to use for instance methods.
• **kwargs** (dict) – Arbitrary keyword arguments.

```python
classmethod de_json(data, bot)
```

```python
def get_file(timeout=None, **kwargs)
```

Convenience wrapper over `telegram.Bot.get_file`

**Parameters**

- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **kwargs** (dict) – Arbitrary keyword arguments.

**Returns**

`telegram.File`

**Raises**

`telegram.TelegramError`

```python
class Voice
```

```python
(file_id, duration, mime_type=None, file_size=None, bot=None, **kwargs)
```

This object represents a voice note.

```python
file_id
```

```python
str – Unique identifier for this file.
```

```python
duration
```

```python
int – Duration of the audio in seconds as defined by sender.
```

```python
mime_type
```

```python
str – Optional. MIME type of the file as defined by sender.
```

```python
file_size
```

```python
int – Optional. File size.
```

```python
bot
```

```python
```

**Parameters**

- **file_id** (str) – Unique identifier for this file.
- **duration** (int, optional) – Duration of the audio in seconds as defined by sender.
- **mime_type** (str, optional) – MIME type of the file as defined by sender.
- **file_size** (int, optional) – File size.
- **bot** (telegram.Bot, optional) – The Bot to use for instance methods.
- **kwargs** (dict) – Arbitrary keyword arguments.

```python
classmethod de_json(data, bot)
```

```python
def get_file(timeout=None, **kwargs)
```

Convenience wrapper over `telegram.Bot.get_file`

**Parameters**
timeout (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

**kwargs (dict) – Arbitrary keyword arguments.

Returns telegram.File

Raises telegram.TelegramError

class telegram.WebhookInfo (url, has_custom_certificate, pending_update_count, last_error_date=None, last_error_message=None, max_connections=None, allowed_updates=None, **kwargs)

This object represents a Telegram WebhookInfo.

Contains information about the current status of a webhook.

url

str – Webhook URL.

has_custom_certificate

bool – If a custom certificate was provided for webhook.

pending_update_count

int – Number of updates awaiting delivery.

last_error_date

int – Optional. Unix time for the most recent error that happened.

last_error_message


max_connections

int – Optional. Maximum allowed number of simultaneous HTTPS connections.

allowed_updates

List[str] – Optional. A list of update types the bot is subscribed to.

Parameters

- **url** (**str**) – Webhook URL, may be empty if webhook is not set up.
- **has_custom_certificate** (**bool**) – True, if a custom certificate was provided for webhook certificate checks.
- **pending_update_count** (**int**) – Number of updates awaiting delivery.
- **last_error_date** (**int**, optional) – Unix time for the most recent error that happened when trying to deliver an update via webhook.
- **last_error_message** (**str**, optional) – Error message in human-readable format for the most recent error that happened when trying to deliver an update via webhook.
- **max_connections** (**int**, optional) – Maximum allowed number of simultaneous HTTPS connections to the webhook for update delivery.
- **allowed_updates** (**List[str]**, optional) – A list of update types the bot is subscribed to. Defaults to all update types.

classmethod de_json (data, bot)

class telegram.Animation (file_id, width, height, duration, thumb=None, file_name=None, mime_type=None, file_size=None, **kwargs)

This object represents an animation file to be displayed in the message containing a game.
file_id
str – Unique file identifier.

width
int – Video width as defined by sender.

height
int – Video height as defined by sender.

duration
int – Duration of the video in seconds as defined by sender.

thumb
telegram.PhotoSize – Optional. Animation thumbnail as defined by sender.

file_name
str – Optional. Original animation filename as defined by sender.

mime_type
str – Optional. MIME type of the file as defined by sender.

file_size
int – Optional. File size.

Parameters

• file_id (str) – Unique file identifier.
• width (int) – Video width as defined by sender.
• height (int) – Video height as defined by sender.
• duration (int) – Duration of the video in seconds as defined by sender.
• thumb (telegram.PhotoSize, optional) – Animation thumbnail as defined by sender.
• file_name (str, optional) – Original animation filename as defined by sender.
• mime_type (str, optional) – MIME type of the file as defined by sender.
• file_size (int, optional) – File size.

classmethod de_json(data, bot)

class telegram.Game (title, description, photo, text=None, text_entities=None, animation=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents a game. Use BotFather to create and edit games, their short names will act as unique identifiers.

title
str – Title of the game.

description
str – Description of the game.

photo
List[telegram.PhotoSize] – Photo that will be displayed in the game message in chats.

text
str – Optional. Brief description of the game or high scores included in the game message. Can be automatically edited to include current high scores for the game when the bot calls set_game_score, or manually edited using edit_message_text.

text_entities
List[telegram.MessageEntity] – Optional. Special entities that appear in text, such as usernames, URLs, bot commands, etc.
animation

telegram.Animation – Optional. Animation that will be displayed in the game message in chats. Upload via BotFather.

Parameters

• **title** (str) – Title of the game.
• **description** (str) – Description of the game.
• **photo** (List[telegram.PhotoSize]) – Photo that will be displayed in the game message in chats.
• **text** (str, optional) – Brief description of the game or high scores included in the game message. Can be automatically edited to include current high scores for the game when the bot calls set_game_score, or manually edited using edit_message_text. 0-4096 characters. Also found as telegram.constants.MAX_MESSAGE_LENGTH.
• **text_entities** (List[telegram.MessageEntity], optional) – Special entities that appear in text, such as usernames, URLs, bot commands, etc.
• **animation** (telegram.Animation, optional) – Animation that will be displayed in the game message in chats. Upload via BotFather.

classmethod de_json(data, bot)

parse_text_entities(types=None)

Returns a dict that maps telegram.MessageEntity to str. It contains entities from this message filtered by their type attribute as the key, and the text that each entity belongs to as the value of the dict.

**Note:** This method should always be used instead of the text_entities attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See parse_text_entity for more info.

Parameters

**types** (List[str], optional) – List of MessageEntity types as strings. If the type attribute of an entity is contained in this list, it will be returned. Defaults to telegram.MessageEntity.ALL_TYPES.

Returns A dictionary of entities mapped to the text that belongs to them, calculated based on UTF-16 codepoints.

Return type Dict[telegram.MessageEntity, str]

parse_text_entity(entity)

Returns the text from a given telegram.MessageEntity.

**Note:** This method is present because Telegram calculates the offset and length in UTF-16 codepoint pairs, which some versions of Python don’t handle automatically. (That is, you can’t just slice Message.text with the offset and length.)

Parameters

**entity** (telegram.MessageEntity) – The entity to extract the text from. It must be an entity that belongs to this message.

Returns The text of the given entity.

Return type str

to_dict()
This object represents one row of the high scores table for a game.

```python
def high_score_row:
    position = int  # Position in high score table for the game.
    user = telegram.User  # User.
    score = int  # Score.
```

### Parameters

- `position` (int) – Position in high score table for the game.
- `user` (telegram.User) – User.
- `score` (int) – Score.

```python
class VideoNote:
    file_id = str  # Unique identifier for this file.
    length = int  # Video width and height as defined by sender.
    duration = int  # Duration of the video in seconds as defined by sender.
    thumb = telegram.PhotoSize  # Optional. Video thumbnail.
    file_size = int  # Optional. File size.
    bot = telegram.Bot  # Optional. The Bot to use for instance methods.
```

### Parameters

- `file_id` (str) – Unique identifier for this file.
- `length` (int) – Video width and height as defined by sender.
- `duration` (int) – Duration of the video in seconds as defined by sender.
- `thumb` (telegram.PhotoSize, optional) – Video thumbnail.
- `file_size` (int, optional) – File size.
- `bot` (telegram.Bot, optional) – The Bot to use for instance methods.
- `**kwargs` (dict) – Arbitrary keyword arguments.

```python
class VideoNote:
    get_file = telegram.Bot.get_file  # Convenience wrapper over telegram.Bot.get_file
```

### Parameters

- `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
**kwargs (dict) – Arbitrary keyword arguments.

Returns telegram.File

Raises telegram.TelegramError

class telegram.LabeledPrice(label, amount, **kwargs)

Bases: telegram.base.TelegramObject

This object represents a portion of the price for goods or services.

label
str – Portion label.

amount
int – Price of the product in the smallest units of the currency.

Parameters

- **label (str)** – Portion label
- **amount (int)** – Price of the product in the smallest units of the currency (integer, not float/double). For example, for a price of US$ 1.45 pass amount = 145. See the exp parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- **kwargs (dict)** – Arbitrary keyword arguments.

class telegram.SuccessfulPayment(currency, total_amount, invoice_payload, telegram_payment_charge_id, provider_payment_charge_id, shipping_option_id=None, order_info=None, **kwargs)

Bases: telegram.base.TelegramObject

This object contains basic information about a successful payment.

currency
str – Three-letter ISO 4217 currency code.

total_amount
int – Total price in the smallest units of the currency.

invoice_payload
str – Bot specified invoice payload.

shipping_option_id
str – Optional. Identifier of the shipping option chosen by the user.

order_info
telegram.OrderInfo – Optional. Order info provided by the user.

telegram_payment_charge_id
str – Telegram payment identifier.

provider_payment_charge_id
str – Provider payment identifier.

Parameters

- **currency (str)** – Three-letter ISO 4217 currency code.
- **total_amount (int)** – Total price in the smallest units of the currency (integer, not float/double). For example, for a price of US$ 1.45 pass amount = 145. See the exp parameter in currencies.json, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- **invoice_payload (str)** – Bot specified invoice payload.
- **shipping_option_id (str, optional)** – Identifier of the shipping option chosen by the user.
**order_info** (*telegram.OrderInfo*, optional) – Order info provided by the user

**telegram_payment_charge_id** (*str*) – Telegram payment identifier.

**provider_payment_charge_id** (*str*) – Provider payment identifier.

**kwargs** (*dict*) – Arbitrary keyword arguments.

classmethod **de_json** (*data*, *bot*)

class **telegram.ShippingOption** (*id*, *title*, *prices*, **kwargs*)

This object represents one shipping option.

**id**

*str* – Shipping option identifier.

**title**

*str* – Option title.

**prices**

[List[*telegram.LabeledPrice*]] – List of price portions.

Parameters

- **id** (*str*) – Shipping option identifier.
- **title** (*str*) – Option title.
- **prices** (List[*telegram.LabeledPrice*]) – List of price portions.
- **kwargs** (*dict*) – Arbitrary keyword arguments.

to_dict ()

class **telegram.ShippingAddress** (*country_code*, *state*, *city*, *street_line1*, *street_line2*, *post_code*, **kwargs*)

This object represents a Telegram ShippingAddress.

**country_code**

*str* – ISO 3166-1 alpha-2 country code.

**state**

*str* – State, if applicable.

**city**

*str* – City.

**street_line1**

*str* – First line for the address.

**street_line2**

*str* – Second line for the address.

**post_code**

*str* – Address post code.

Parameters

- **country_code** (*str*) – ISO 3166-1 alpha-2 country code.
- **state** (*str*) – State, if applicable.
- **city** (*str*) – City.
- **street_line1** (*str*) – First line for the address.
- **street_line2** (*str*) – Second line for the address.
• **post_code** (str) – Address post code.
• **kwargs** (dict) – Arbitrary keyword arguments.

```python
classmethod de_json(data, bot)
```

class `telegram.PreCheckoutQuery`:

```python
class telegram.PreCheckoutQuery(id, from_user, currency, total_amount, invoice_payload, shipping_option_id=None, order_info=None, bot=None, **kwargs)
```

**Bases:** `telegram.base.TelegramObject`

This object contains information about an incoming pre-checkout query.

**Note:**
• In Python `from` is a reserved word, use `from_user` instead.

**Parameters**

• `id` (str) – Unique query identifier.
• `from_user` (`telegram.User`) – User who sent the query.
• `currency` (str) – Three-letter ISO 4217 currency code.
• `total_amount` (int) – Total price in the smallest units of the currency.
• `invoice_payload` (str) – Bot specified invoice payload.
• `shipping_option_id` (str, optional) – Identifier of the shipping option chosen by the user.
• `order_info` (`telegram.OrderInfo`, optional) – Order info provided by the user.
• `bot` (`telegram.Bot`, optional) – The Bot to use for instance methods.
• **kwargs** (dict) – Arbitrary keyword arguments.

```python
answer(*args, **kwargs)
```

Shortcut for:
bot.answer_pre_checkout_query(update.pre_checkout_query.id, *args, **kwargs)

Parameters

- **ok**(bool) – Specify True if everything is alright (goods are available, etc.) and the bot is ready to proceed with the order. Use False if there are any problems.

- **error_message**(str, optional) – Required if ok is False. Error message in human readable form that explains the reason for failure to proceed with the checkout (e.g. “Sorry, somebody just bought the last of our amazing black T-shirts while you were busy filling out your payment details. Please choose a different color or garment!”). Telegram will display this message to the user.

- ****kwargs**(dict) – Arbitrary keyword arguments.

classmethod de_json(data, bot)

class telegram.OrderInfo(name=None, phone_number=None, email=None, shipping_address=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents information about an order.

**name**

str – Optional. User name.

**phone_number**

str – Optional. User’s phone number.

**email**

str – Optional. User email.

**shipping_address**

telegram.ShippingAddress – Optional. User shipping address.

Parameters

- **name**(str, optional) – User name.

- **phone_number**(str, optional) – User’s phone number.

- **email**(str, optional) – User email.

- **shipping_address**(telegram.ShippingAddress, optional) – User shipping address.

- ****kwargs**(dict) – Arbitrary keyword arguments.

classmethod de_json(data, bot)

class telegram.Invoice(title, description, start_parameter, currency, total_amount, **kwargs)

Bases: telegram.base.TelegramObject

This object contains basic information about an invoice.

**title**

str – Product name.

**description**

str – Product description.

**start_parameter**

str – Unique bot deep-linking parameter.

**currency**

str – Three-letter ISO 4217 currency code.
total_amount
int – Total price in the smallest units of the currency.

Parameters

• title (str) – Product name.
• description (str) – Product description.
• start_parameter (str) – Unique bot deep-linking parameter that can be used to generate this invoice.
• currency (str) – Three-letter ISO 4217 currency code.
• total_amount (int) – Total price in the smallest units of the currency (integer, not float/double). For example, for a price of US$ 1.45 pass amount = 145.
• **kwargs (dict) – Arbitrary keyword arguments.

classmethod de_json(data, bot)

This object contains information about an incoming shipping query.

class telegram.ShippingQuery(id, from_user, invoice_payload, shipping_address, bot=None, **kwargs)

Bases: telegram.base.TelegramObject

Note:

• In Python from is a reserved word, use from_user instead.

id
str – Unique query identifier.

from_user
telegram.User – User who sent the query.

invoice_payload
str – Bot specified invoice payload.

shipping_address
telegram.ShippingAddress – User specified shipping address.

bot

Parameters

• id (str) – Unique query identifier.
• from_user (telegram.User) – User who sent the query.
• invoice_payload (str) – Bot specified invoice payload.
• shipping_address (telegram.ShippingAddress) – User specified shipping address.
• bot (telegram.Bot, optional) – The Bot to use for instance methods.
• **kwargs (dict) – Arbitrary keyword arguments.

answer(*args, **kwargs)

Shortcut for:

bot.answer_shipping_query(update.shipping_query.id, *args, **kwargs)
• **ok** (bool) – Specify True if delivery to the specified address is possible and False if there are any problems (for example, if delivery to the specified address is not possible).

• **shipping_options** (List[telegram.ShippingOption], optional) – Required if ok is True. A JSON-serialized array of available shipping options.

• **error_message** (str, optional) – Required if ok is False. Error message in human readable form that explains why it is impossible to complete the order (e.g. "Sorry, delivery to your desired address is unavailable"). Telegram will display this message to the user.

classmethod de_json(data, bot)

class telegram.ChatPhoto(small_file_id, big_file_id, bot=None, **kwargs)
    Bases: telegram.base.TelegramObject

This object represents a chat photo.

**small_file_id**
str – Unique file identifier of small (160x160) chat photo.

**big_file_id**
str – Unique file identifier of big (640x640) chat photo.

Parameters

• **small_file_id** (str) – Unique file identifier of small (160x160) chat photo. This file_id can be used only for photo download.

• **big_file_id** (str) – Unique file identifier of big (640x640) chat photo. This file_id can be used only for photo download.

• **bot** (telegram.Bot, optional) – The Bot to use for instance methods

• **kwargs** (dict) – Arbitrary keyword arguments.

classmethod de_json(data, bot)

class telegram.StickerSet(name, title, contains_masks, stickers, bot=None, **kwargs)
    Bases: telegram.base.TelegramObject

This object represents a sticker set.

**name**
str – Sticker set name.

**title**
str – Sticker set title.

**contains_masks**
bool – True, if the sticker set contains masks.

**stickers**

Parameters

• **name** (str) – Sticker set name.

• **title** (str) – Sticker set title.

• **contains_masks** (bool) – True, if the sticker set contains masks.

• **stickers** (List[telegram.Sticker]) – List of all set stickers.

static de_json(data, bot)

to_dict()
class telegram.MaskPosition(point, x_shift, y_shift, scale, **kwargs)
    Bases: telegram.base.TelegramObject
    This object describes the position on faces where a mask should be placed by default.

    point
        str – The part of the face relative to which the mask should be placed.

    x_shift
        float – Shift by X-axis measured in widths of the mask scaled to the face size, from left to right.

    y_shift
        float – Shift by Y-axis measured in heights of the mask scaled to the face size, from top to bottom.

    scale
        float – Mask scaling coefficient. For example, 2.0 means double size.

Notes

    type should be one of the following: forehead, eyes, mouth or chin. You can use the classconstants for those.

Parameters

    • point (str) – The part of the face relative to which the mask should be placed.
    • x_shift (float) – Shift by X-axis measured in widths of the mask scaled to the face size, from left to right. For example, choosing -1.0 will place mask just to the left of the default mask position.
    • y_shift (float) – Shift by Y-axis measured in heights of the mask scaled to the face size, from top to bottom. For example, 1.0 will place the mask just below the default mask position.
    • scale (float) – Mask scaling coefficient. For example, 2.0 means double size.

CHIN = 'chin'
    str – 'chin'

EYES = 'eyes'
    str – 'eyes'

FOREHEAD = 'forehead'
    str – 'forehead'

MOUTH = 'mouth'
    str – 'mouth'

classmethod de_json(data, bot)
class telegram.CallbackGame
    Bases: telegram.base.TelegramObject
    A placeholder, currently holds no information. Use BotFather to set up your game.

class telegram.InputMedia
    Bases: telegram.base.TelegramObject
    Base class for Telegram InputMedia Objects.

class telegram.InputMediaPhoto(media, caption=None, parse_mode=None)
    Bases: telegram.files.inputmedia.InputMedia
    Represents a photo to be sent.
**InputMedia class**

Represents a media to be sent.

**type**

str – photo.

**media**

str – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.PhotoSize` object to send.

**caption**

str – Optional. Caption of the photo to be sent, 0-1024 characters.

**parse_mode**

str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

**Parameters**

- **media** (str) – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.PhotoSize` object to send.
- **caption** (str, optional) – Caption of the photo to be sent, 0-1024 characters.
- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

**class telegram.InputMediaVideo**

Bases: `telegram.files.inputmedia.InputMedia`

Represents a video to be sent.

**type**

str – video.

**media**

str – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.Video` object to send.

**caption**

str – Optional. Caption of the video to be sent, 0-1024 characters.

**parse_mode**

str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

**width**

int – Optional. Video width.

**height**

int – Optional. Video height.

**duration**

int – Optional. Video duration.

**supports_streaming**

bool – Optional. Pass True, if the uploaded video is suitable for streaming.

**thumb**

filelike object – Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.
Parameters

- **media** *(str)* – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing telegram.Video object to send.

- **caption** *(str, optional)* – Caption of the video to be sent, 0-1024 characters.

- **parse_mode** *(str, optional)* – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

- **width** *(int, optional)* – Video width.

- **height** *(int, optional)* – Video height.

- **duration** *(int, optional)* – Video duration.

- **supports_streaming** *(bool, optional)* – Pass True, if the uploaded video is suitable for streaming.

- **thumb** *(filelike object, optional)* – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

**Note:** When using a telegram.Video for the media attribute. It will take the width, height and duration from that video, unless otherwise specified with the optional arguments.

class telegram.PassportElementError *(source, type, message, **kwargs)*
Baseclass for the PassportElementError* classes.

**source**
*str* – Error source.

**type**
*str* – The section of the user’s Telegram Passport which has the error.

**message**
*str* – Error message

Parameters

- **source** *(str)* – Error source.

- **type** *(str)* – The section of the user’s Telegram Passport which has the error.

- **kwargs** *(dict)* – Arbitrary keyword arguments.

class telegram.PassportElementErrorFile *(type, file_hash, message, **kwargs)*
Baseclass: telegram.passport.passportelementerrors.PassportElementError

Represents an issue with a document scan. The error is considered resolved when the file with the document scan changes.

**type**
*str* – The section of the user’s Telegram Passport which has the issue, one of “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.

**file_hash**
*str* – Base64-encoded file hash.

**message**
*str* – Error message.

Parameters
• **type** *(str)* – The section of the user’s Telegram Passport which has the issue, one of “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.

• **file_hash** *(str)* – Base64-encoded file hash.

• **message** *(str)* – Error message.

• **kwargs** *(dict)* – Arbitrary keyword arguments.

```python
class telegram.PassportElementErrorReverseSide(type, file_hash, message, **kwargs)
Bases: telegram.passport.passpordelementerrors.PassportElementError
```

Represents an issue with the front side of a document. The error is considered resolved when the file with the reverse side of the document changes.

• **type** *(str)* – The section of the user’s Telegram Passport which has the issue, one of “passport”, “driver_license”, “identity_card”.

• **file_hash** *(str)* – Base64-encoded hash of the file with the reverse side of the document.

• **message** *(str)* – Error message.

• **kwargs** *(dict)* – Arbitrary keyword arguments.

```python
class telegram.PassportElementErrorFrontSide(type, file_hash, message, **kwargs)
Bases: telegram.passport.passpordelementerrors.PassportElementError
```

Represents an issue with the front side of a document. The error is considered resolved when the file with the front side of the document changes.

• **type** *(str)* – The section of the user’s Telegram Passport which has the issue, one of “passport”, “driver_license”, “identity_card”, “internal_passport”.

• **file_hash** *(str)* – Base64-encoded hash of the file with the front side of the document.

• **message** *(str)* – Error message.

• **kwargs** *(dict)* – Arbitrary keyword arguments.

```python
class telegram.PassportElementErrorFiles(type, file_hashes, message, **kwargs)
Bases: telegram.passport.passpordelementerrors.PassportElementError
```

296 Chapter 1. telegram package
Represents an issue with a list of scans. The error is considered resolved when the file with the document scan changes.

type
  str – The section of the user’s Telegram Passport which has the issue, one of “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.

file_hash
  str – Base64-encoded file hash.

message
  str – Error message.

Parameters

  • type (str) – The section of the user’s Telegram Passport which has the issue, one of “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.
  • file_hashes (List[str]) – List of base64-encoded file hashes.
  • message (str) – Error message.
  • **kwargs (dict) – Arbitrary keyword arguments.

class telegram.PassportElementErrorDataField(type, field_name, data_hash, message, **kwargs)
Bases: telegram.passport.passportelementerrors.PassportElementError
Represents an issue in one of the data fields that was provided by the user. The error is considered resolved when the field’s value changes.

type
  str – The section of the user’s Telegram Passport which has the error, one of “personal_details”, “passport”, “driver_license”, “identity_card”, “internal_passport”, “address”.

field_name
  str – Name of the data field which has the error.

data_hash
  str – Base64-encoded data hash.

message
  str – Error message.

Parameters

  • type (str) – The section of the user’s Telegram Passport which has the error, one of “personal_details”, “passport”, “driver_license”, “identity_card”, “internal_passport”, “address”.
  • field_name (str) – Name of the data field which has the error.
  • data_hash (str) – Base64-encoded data hash.
  • message (str) – Error message.
  • **kwargs (dict) – Arbitrary keyword arguments.

class telegram.PassportElementErrorFile(type, file_hash, message, **kwargs)
Bases: telegram.passport.passportelementerrors.PassportElementError
Represents an issue with a document scan. The error is considered resolved when the file with the document scan changes.

type
  str – The section of the user’s Telegram Passport which has the issue, one of “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.

1.49. Module contents 297
file_hash
str – Base64-encoded file hash.

message
str – Error message.

Parameters

• **type**(str) – The section of the user’s Telegram Passport which has the issue, one of “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.

• **file_hash**(str) – Base64-encoded file hash.

• **message**(str) – Error message.

• **kwargs**(dict) – Arbitrary keyword arguments.

class telegram.Credentials(secure_data, nonce, bot=None, **kwargs)
Bases: telegram.base.TelegramObject

secure_data
telegram.SecureData – Credentials for encrypted data

nonce
str – Bot-specified nonce

classmethod de_json(data, bot)

class telegram.DataCredentials(data_hash, secret, **kwargs)
Bases: telegrampassport.credentials._CredentialsBase

These credentials can be used to decrypt encrypted data from the data field in EncryptedPassportData.

Parameters

• **data_hash**(str) – Checksum of encrypted data

• **secret**(str) – Secret of encrypted data

hash
str – Checksum of encrypted data

secret
str – Secret of encrypted data

to_dict()
identity_card
telescope.SecureValue, optional – Credentials for encrypted ID card

des
	 telegram.SecureValue, optional – Credentials for encrypted residential address.

utility_bill

bank_statement

rental_agreement
telescope.SecureValue, optional – Credentials for encrypted rental agreement.

passport_registration
telescope.SecureValue, optional – Credentials for encrypted registration from internal passport.

temporary_registration
telescope.SecureValue, optional – Credentials for encrypted temporary registration.

classmethod de_json(data, bot)

class telescope.FileCredentials(file_hash, secret, **kwargs)
Bases: telescope.passport.credentials._CredentialsBase

These credentials can be used to decrypt encrypted files from the front_side, reverse_side, selfie and files fields in EncryptedPassportData.

Parameters

* file_hash (str) – Checksum of encrypted file
* secret (str) – Secret of encrypted file

hash
str – Checksum of encrypted file

secret
str – Secret of encrypted file

to_dict ()

class telescope.IdDocumentData(document_no, expiry_date, bot=None, **kwargs)
Bases: telescope.base.TelegramObject

This object represents the data of an identity document.

document_no
str – Document number.

expiry_date
str – Optional. Date of expiry, in DD.MM.YYYY format.

classmethod de_json(data, bot)

class telescope.PersonalDetails(first_name, last_name, birth_date, gender, country_code, residence_country_code, first_name_native=None, last_name_native=None, middle_name=None, middle_name_native=None, bot=None, **kwargs)
Bases: telescope.base.TelegramObject

This object represents personal details.

first_name
str – First Name.

middle_name
str – Optional. First Name.
last_name
str – Last Name.

birth_date
str – Date of birth in DD.MM.YYYY format.

gender
str – Gender, male or female.

country_code
str – Citizenship (ISO 3166-1 alpha-2 country code).

residence_country_code
str – Country of residence (ISO 3166-1 alpha-2 country code).

first_name
str – First Name in the language of the user’s country of residence.

middle_name
str – Optional. Middle Name in the language of the user’s country of residence.

last_name
str – Last Name in the language of the user’s country of residence.

classmethod de_json(data, bot)

class telegram.ResidentialAddress(street_line1, street_line2, city, state, country_code, post_code, bot=None, **kwargs)
Bases: telegram.base.TelegramObject

This object represents a residential address.

street_line1
str – First line for the address.

street_line2
str – Optional. Second line for the address.

city
str – City.

state
str – Optional. State.

country_code
str – ISO 3166-1 alpha-2 country code.

post_code
str – Address post code.

classmethod de_json(data, bot)

class telegram.InputMediaVideo(media, caption=None, width=None, height=None, duration=None, supports_streaming=None, parse_mode=None, thumb=None)
Bases: telegram.files.inputmedia.InputMedia

Represents a video to be sent.

type
str – video.

media
str – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing telegram.Video object to send.

caption
str – Optional. Caption of the video to be sent, 0-1024 characters.
parse_mode
   str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

width
   int – Optional. Video width.

height
   int – Optional. Video height.

duration
   int – Optional. Video duration.

supports_streaming
   bool – Optional. Pass True, if the uploaded video is suitable for streaming.

thumb
   filelike object – Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

Parameters

- **media** (str) – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing telegram.Video object to send.
- **caption** (str, optional) – Caption of the video to be sent, 0-1024 characters.
- **parse_mode** (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
- **width** (int, optional) – Video width.
- **height** (int, optional) – Video height.
- **duration** (int, optional) – Video duration.
- **supports_streaming** (bool, optional) – Pass True, if the uploaded video is suitable for streaming.
- **thumb** (filelike object, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

Note: When using a telegram.Video for the media attribute. It will take the width, height and duration from that video, unless otherwise specified with the optional arguments.

class telegram.InputMediaAnimation
   (media, thumb=None, caption=None, parse_mode=None, width=None, height=None, duration=None)

Bases: telegram.files.inputmedia.InputMedia

Represents an animation file (GIF or H.264/MPEG-4 AVC video without sound) to be sent.

type
   str – animation.

media
   str – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing telegram.Animation object to send.
thumb

filelike object – Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not passed as a string or file_id.

caption

str – Optional. Caption of the animation to be sent, 0-1024 characters.

parse_mode

str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

width

int – Optional. Animation width.

height

int – Optional. Animation height.

duration

int – Optional. Animation duration.

Parameters

• media (str) – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing telegram.Animation object to send.

• thumb (filelike object, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not passed as a string or file_id.

• caption (str, optional) – Caption of the animation to be sent, 0-1024 characters.

• parse_mode (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

• width (int, optional) – Animation width.

• height (int, optional) – Animation height.

• duration (int, optional) – Animation duration.

Note:  When using a telegram.Animation for the media attribute. It will take the width, height and duration from that video, unless otherwise specified with the optional arguments.

class telegram.InputMediaAudio (media, thumb=None, caption=None, parse_mode=None, duration=None, performer=None, title=None)

Bases: telegram.files.inputmedia.InputMedia

Represents an audio file to be treated as music to be sent.

type

str – audio.

media

str – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing telegram.Audio object to send.

caption

str – Optional. Caption of the audio to be sent, 0-1024 characters.
parse_mode

str – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

duration

int – Duration of the audio in seconds.

performer

str – Optional. Performer of the audio as defined by sender or by audio tags.

title

str – Optional. Title of the audio as defined by sender or by audio tags.

thumb

filelike object – Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

Parameters

- media (str) – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing telegram.Document object to send.
- caption (str, optional) – Caption of the audio to be sent, 0-1024 characters.
- parse_mode (str, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
- duration (int) – Duration of the audio in seconds as defined by sender.
- performer (str, optional) – Performer of the audio as defined by sender or by audio tags.
- title (str, optional) – Title of the audio as defined by sender or by audio tags.
- thumb (filelike object, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

Note: When using a telegram.Audio for the media attribute. It will take the duration, performer and title from that video, unless otherwise specified with the optional arguments.

class telegram.InputMediaDocument (media, thumb=None, caption=None, parse_mode=None)

Bases: telegram.files.inputmedia.InputMedia

Represents a general file to be sent.

type

str – document.

media

str – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing telegram.Document object to send.

caption

str – Optional. Caption of the document to be sent, 0-1024 characters.
**parse_mode**

*str* – Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

**thumb**

*filelike object* – Optional. Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

**Parameters**

- **media** (*str*) – File to send. Pass a file_id to send a file that exists on the Telegram servers (recommended), pass an HTTP URL for Telegram to get a file from the Internet. Lastly you can pass an existing `telegram.Document` object to send.
- **caption** (*str*, optional) – Caption of the document to be sent, 0-1024 characters.
- **parse_mode** (*str*, optional) – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
- **thumb** (*filelike object*, optional) – Thumbnail of the file sent. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 90. Ignored if the file is not is passed as a string or file_id.

**exception** `telegram.TelegramDecryptionError`(message)

Bases: `telegram.error.TelegramError`

Something went wrong with decryption.

**class** `telegram.PassportElementErrorSelfie`(type, file_hash, message, **kwargs)

Bases: `telegram.passport.passportelementerrors.PassportElementError`

Represents an issue with the selfie with a document. The error is considered resolved when the file with the selfie changes.

**type**

*str* – The section of the user’s Telegram Passport which has the issue, one of “passport”, “driver_license”, “identity_card”, “internal_passport”.

**file_hash**

*str* – Base64-encoded hash of the file with the selfie.

**message**

*str* – Error message.

**Parameters**

- **type** (*str*) – The section of the user’s Telegram Passport which has the issue, one of “passport”, “driver_license”, “identity_card”, “internal_passport”.
- **file_hash** (*str*) – Base64-encoded hash of the file with the selfie.
- **message** (*str*) – Error message.
- ****kwargs (dict) – Arbitrary keyword arguments.

**class** `telegram.PassportElementErrorTranslationFile`(type, file_hash, message, **kwargs)

Bases: `telegram.passport.passportelementerrors.PassportElementError`

Represents an issue with one of the files that constitute the translation of a document. The error is considered resolved when the file changes.
type
str – Type of element of the user’s Telegram Passport which has the issue, one of “passport”, “driver_license”, “identity_card”, “internal_passport”, “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.

file_hash
str – Base64-encoded hash of the file.

message
str – Error message.

Parameters

- type (str) – Type of element of the user’s Telegram Passport which has the issue, one of “passport”, “driver_license”, “identity_card”, “internal_passport”, “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.
- file_hash (str) – Base64-encoded hash of the file.
- message (str) – Error message.
- **kwargs (dict) – Arbitrary keyword arguments.

class telegram.PassportElementErrorTranslationFiles(type, file_hashes, message, **kwargs)
Bases: telegram.passport.passportelementerrors.PassportElementError

Represents an issue with the translated version of a document. The error is considered resolved when a file with the document translation change.

type
str – Type of element of the user’s Telegram Passport which has the issue, one of “passport”, “driver_license”, “identity_card”, “internal_passport”, “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.

file_hash
str – Base64-encoded file hash.

message
str – Error message.

Parameters

- type (str) – Type of element of the user’s Telegram Passport which has the issue, one of “passport”, “driver_license”, “identity_card”, “internal_passport”, “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.
- file hashes (List[str]) – List of base64-encoded file hashes.
- message (str) – Error message.
- **kwargs (dict) – Arbitrary keyword arguments.

class telegram.PassportElementErrorUnspecified(type, element_hash, message, **kwargs)
Bases: telegram.passport.passportelementerrors.PassportElementError

Represents an issue in an unspecified place. The error is considered resolved when new data is added.

type
str – Type of element of the user’s Telegram Passport which has the issue.

element_hash
str – Base64-encoded element hash.
**message**
   
   str – Error message.

**Parameters**

- **type** *(str)* – Type of element of the user’s Telegram Passport which has the issue.
- **element_hash** *(str)* – Base64-encoded element hash.
- **message** *(str)* – Error message.
- ****kwargs**(dict)* – Arbitrary keyword arguments.
CHAPTER 2

Changelog

2.1 Changes
Released 2019-02-13
First beta release ever. It has been so long since last release that we would like to test the impact before a final release.

*We do NOT recommend using this beta release in production.*

**Major changes:**

- Context based callbacks
- Persistence
- PrefixHandler added (Handler overhaul)
- Deprecation of RegexHandler and edited_messages, channel_post, etc. arguments (Filter overhaul)
- Various ConversationHandler changes and fixes

See the wiki page at https://git.io/fxJuV for a detailed guide on how to migrate from version 11 to version 12.

### 3.1 Context based callbacks (#1100)

- Use of `pass__` in handlers is deprecated.
- Instead use `use_context=True` on `Updater` or `Dispatcher` and change callback from `(bot, update, others...)` to `(update, context)`.
- This also applies to error handlers `Dispatcher.add_error_handler` and `JobQueue` jobs (change `(bot, job)` to `(context)` here).
- For users with custom handlers subclassing Handler, this is mostly backwards compatible, but to use the new context based callbacks you need to implement the new `collect_additional_context` method.
- Passing `bot` to `JobQueue.__init__` is deprecated. Use `JobQueue.set_dispatcher` with a dispatcher instead.
- Dispatcher makes sure to use a single `CallbackContext` for a entire update. This means that if an update is handled by multiple handlers (by using the group argument), you can add custom arguments to the `CallbackContext` in a lower group handler and use it in higher group handler. NOTE: Never use with @run_async, see docs for more info. (#1283)
• If you have custom handlers they will need to be updated to support the changes in this release.
• Update all examples to use context based callbacks.

3.2 Persistence (#1017)

• Added PicklePersistence and DictPersistence for adding persistence to your bots.
• BasePersistence can be subclassed for all your persistence needs.
• Add a new example that shows a persistent ConversationHandler bot

3.3 Handler overhaul (#1114)

• CommandHandler now only triggers on actual commands as defined by telegram servers (everything that
the clients mark as a tabable link).
• PrefixHandler can be used if you need to trigger on prefixes (like all messages starting with a “/” (old
CommandHandler behaviour) or even custom prefixes like “#” or “!”).

3.4 Filter overhaul (#1221)

• RegexHandler is deprecated and should be replaced with a MessageHandler with a regex filter.
• Use update filters to filter update types instead of arguments (message_updates, channel_post_updates and
edited_updates) on the handlers.
• Completely remove allow_edited argument - it has been deprecated for a while.
• data_filters now exist which allows filters that return data into the callback function. This is how the regex
filter is implemented.
• All this means that it no longer possible to use a list of filters in a handler. Use bitwise operators instead!

3.5 ConversationHandler

• Remove run_async_timeout and timed_out_behavior arguments (#1344)
• Replace with WAITING constant and behavior from states (#1344)
• Only emit one warning for multiple CallbackQueryHandlers in a ConversationHandler (#1319)
• Use warnings.warn for ConversationHandler warnings (#1343)
• Fix unresolvable promises (#1270)

3.6 Bug fixes & improvements

• Handlers should be faster due to deduped logic.
• Avoid compiling compiled regex in regex filter. (#1314)
• Add missing left_chat_member to Message.MESSAGE_TYPES (#1336)
• Make custom timeouts actually work properly (#1330)
• Add convenience classmethods (from_button, from_row and from_column) to InlineKeyboardMarkup
• Small typo fix in setup.py (#1306)
• Add Conflict error (HTTP error code 409) (#1154)
• Change MAX_CAPTION_LENGTH to 1024 (#1262)
• Remove some unnecessary clauses (#1247, #1239)
• Allow filenames without dots in them when sending files (#1228)
• Fix uploading files with unicode filenames (#1214)
• Replace http.server with Tornado (#1191)
• Allow SOCKSConnection to parse username and password from URL (#1211)
• Fix for arguments in passport/data.py (#1213)
• Improve message entity parsing by adding text_mention (#1206)

### 3.7 Internal improvements

• Finally fix our CI builds mostly (too many commits and PRs to list)
• Use multiple bots for CI to improve testing times significantly.
• Allow pypy to fail in CI.
• Remove the last CamelCase CheckUpdate methods from the handlers we missed earlier.
Pre-2019 (up and including to version 11.1.0)

2018-09-01 Released 11.1.0
Fixes and updates for Telegram Passport: (#1198)
• Fix passport decryption failing at random times
• Added support for middle names.
• Added support for translations for documents
• Add errors for translations for documents
• Added support for requesting names in the language of the user’s country of residence
• Replaced the payload parameter with the new parameter nonce
• Add hash to EncryptedPassportElement

2018-08-29 Released 11.0.0
Fully support Bot API version 4.0! (also some bugfixes :))
Telegram Passport (#1174):
• Add full support for telegram passport.
  – New types: PassportData, PassportFile, EncryptedPassportElement, EncryptedCredentials,
  – New bot method: set_passport_data_errors
  – New filter: Filters.passport_data
  – Field passport_data field on Message
  – PassportData can be easily decrypted.
  – PassportFiles are automatically decrypted if originating from decrypted PassportData.
• See new passportbot.py example for details on how to use, or go to our telegram passport wiki page for more info
• NOTE: Passport decryption requires new dependency cryptography.
• Change how Inputfile is handled internally
• This allows support for specifying the thumbnails of photos and videos using the thumb= argument in the different send_ methods.
• Also allows Bot.send_media_group to actually finally send more than one media.
• Add thumb to Audio, Video and Videonote
• Add Bot.edit_message_media together with InputMediaAnimation, InputMediaAudio, and inputMediaDocument.

Other Bot API 4.0 changes:
• Add forusquare_type to Venue, InlineQueryResultVenue, InputVenueMessageContent, and Bot.send_venue. (#1170)
• Add vCard support by adding vcard field to Contact, InlineQueryResultContact, InputContactMessageContent, and Bot.send_contact. (#1166)
• Support new message entities: CASHTAG and PHONE_NUMBER. (#1179)
  – Cashtag seems to be things like $USD and $GBP, but it seems telegram doesn’t currently send them to bots.
  – Phone number also seems to have limited support for now
• Add Bot.send_animation, add width, height, and duration to Animation, and add Filters.animation. (#1172)

Non Bot API 4.0 changes:
• Minor integer comparison fix (#1147)
• Fix Filters.regex failing on non-text message (#1158)
• Fix ProcessLookupError if process finishes before we kill it (#1126)
• Add t.me links for User, Chat and Message if available and update User.mention_* (#1092)
• Fix mention_markdown/html on py2 (#1112)

2018-05-02 Released 10.1.0
Fixes changing previous behaviour:
• Add urllib3 fix for socks5h support (#1085)
• Fix send_sticker() timeout=20 (#1088)

Fixes:
• Add a caption_entity filter for filtering caption entities (#1068)
• Inputfile encode filenames (#1086)
• InputFile: Fix proper naming of file when reading from subprocess.PIPE (#1079)
• Remove pytest-catchlog from requirements (#1099)
• Documentation fixes (#1061, #1078, #1081, #1096)

2018-04-17 Released 10.0.2
Important fix:
• Handle utf8 decoding errors (#1076)

New features:
• Added Filter.regex (#1028)
• Filters for Category and file types (#1046)
• Added video note filter (#1067)

Fixes:
• Fix in telegram.Message (#1042)
• Make chat_id a positional argument inside shortcut methods of Chat and User classes (#1050)
• Make Bot.full_name return a unicode object. (#1063)
• CommandHandler faster check (#1074)
• Correct documentation of Dispatcher.add_handler (#1071)
• Various small fixes to documentation.

2018-03-05 Released 10.0.1

Fixes:
• Fix conversationhandler timeout (PR #1032)
• Add missing docs utils (PR #912)

2018-03-02 Released 10.0.0

Non backward compatible changes and changed defaults
• JobQueue: Remove deprecated prevent_autostart & put() (PR #1012)
• Bot, Updater: Remove deprecated network_delay (PR #1012)
• Remove deprecated Message.new_chat_member (PR #1012)
• Retry bootstrap phase indefinitely (by default) on network errors (PR #1018)

New Features
• Support v3.6 API (PR #1006)
• User.full_name convinience property (PR #949)
• Add send_phone_number_to_provider and send_email_to_provider arguments to send_invoice (PR #986)
• Bot: Add shortcut methods reply_{markdown,html} (PR #827)
• Bot: Add shortcut method reply_media_group (PR #994)
• Added utils.helpers.effective_message_type (PR #826)
• Bot.get_file now allows passing a file in addition to file_id (PR #963)
• Add .get_file() to Audio, Document, PhotoSize, Sticker, Video, VideoNote and Voice (PR #963)
• Add .send_*() methods to User and Chat (PR #963)
• Get jobs by name (PR #1011)
• Add Message caption html/markdown methods (PR #1013)
• File.download_as_bytearray - new method to get a d/led file as bytearray (PR #1019)
• File.download(): Now returns a meaningful return value (PR #1019)
• Added conversation timeout in ConversationHandler (PR #895)

Changes
• Store bot in PreCheckoutQuery (PR #953)
• Updater: Issue INFO log upon received signal (PR #951)
• JobQueue: Thread safety fixes (PR #977)
• WebhookHandler: Fix exception thrown during error handling (PR #985)
• Explicitly check update.effective_chat in ConversationHandler.check_update (PR #959)
• Updater: Better handling of timeouts during get_updates (PR #1007)
• Remove unnecessary to_dict() (PR #834)
• CommandHandler - ignore strings in entities and “/” followed by whitespace (PR #1020)
• Documentation & style fixes (PR #942, PR #956, PR #962, PR #980, PR #983)

2017-12-08 Released 9.0.0

Breaking changes (possibly)

• Drop support for python 3.3 (PR #930)

New Features

• Support Bot API 3.5 (PR #920)

Changes

• Fix race condition in dispatcher start/stop (#887)
• Log error trace if there is no error handler registered (#694)
• Update examples with consistent string formatting (#870)
• Various changes and improvements to the docs.

2017-10-15 Released 8.1.1

• Fix Commandhandler crashing on single character messages (PR #873).

2017-10-14 Released 8.1.0

New features - Support Bot API 3.4 (PR #865).

Changes - MessageHandler & RegexHandler now consider channel_updates. - Fix command not recognized if it is directly followed by a newline (PR #869). - Removed Bot._message_wrapper (PR #822). - Unittests are now also running on AppVeyor (Windows VM). - Various unitest improvements. - Documentation fixes.

2017-09-01 Released 8.0.0

New features

• Fully support Bot Api 3.3 (PR #806).
• DispatcherHandlerStop (see docs).
• Regression fix for text_html & text_markdown (PR #777).
• Added effective_attachment to message (PR #766).

Non backward compatible changes

• Removed Botan support from the library (PR #776).
• Fully support Bot Api 3.3 (PR #806).
• Remove de_json() (PR #789).

Changes

• Sane defaults for tcp socket options on linux (PR #754).
• Add RESTRICTED as constant to ChatMember (PR #761).
• Add rich comparison to CallbackQuery (PR #764).
• Fix get_game_high_scores (PR #771).
• Warn on small con_pool_size during custom initialization of Updater (PR #793).
• Catch exceptions in error handlerfor errors that happen during polling (PR #810).
• For testing we switched to pytest (PR #788).
• Lots of small improvements to our tests and documentation.

2017-07-28 Released 7.0.1

• Fix TypeError exception in RegexHandler (PR #751).
• Small documentation fix (PR #749).

2017-07-25 Released 7.0.0

• Fully support Bot API 3.2.
• New filters for handling messages from specific chat/user id (PR #677).
• Add the possibility to add objects as arguments to send_* methods (PR #742).
• Fixed download of URLs with UTF-8 chars in path (PR #688).
• Fixed URL parsing for Message text properties (PR #689).
• Fixed args dispatching in MessageQueue's decorator (PR #705).
• Fixed regression preventing IPv6 only hosts from connecting to Telegram servers (Issue #720).
• ConversationHandler - check if a user exist before using it (PR #699).
• Removed deprecated telegram.Emoji.
• Removed deprecated Botan import from utils (Botan is still available through contrib).
• Removed deprecated ReplyKeyboardHide.
• Removed deprecated edit_message argument of bot.set_game_score.
• Internal restructure of files.
• Improved documentation.
• Improved unitests.

2017-06-18

Released 6.1.0

• Fully support Bot API 3.0
• Add more fine-grained filters for status updates
• Bug fixes and other improvements

2017-05-29

Released 6.0.3

• Faulty PyPI release

2017-05-29

Released 6.0.2

• Avoid confusion with user's urllib3 by renaming vendored urllib3 to ptb_urlib3

2017-05-19

Released 6.0.1

• Add support for User.language_code
• Fix Message.text_html and Message.text_markdown for messages with emoji

2017-05-19

Released 6.0.0

• Add support for Bot API 2.3.1
• Add support for deleteMessage API method
• New, simpler API for JobQueue - https://github.com/python-telegram-bot/python-telegram-bot/pull/484
• Download files into file-like objects - https://github.com/python-telegram-bot/python-telegram-bot/pull/459
• Use vendor `urllib3` to address issues with timeouts - The default timeout for messages is now 5 seconds. For sending media, the default timeout is now 20 seconds.

• String attributes that are not set are now `None` by default, instead of empty strings.

• Add `text_markdown` and `text_html` properties to `Message` - https://github.com/python-telegram-bot/python-telegram-bot/pull/507

• Add support for Socks5 proxy - https://github.com/python-telegram-bot/python-telegram-bot/pull/518

• Add support for filters in `CommandHandler` - https://github.com/python-telegram-bot/python-telegram-bot/pull/536

• Add the ability to invert (not) filters - https://github.com/python-telegram-bot/python-telegram-bot/pull/552

• Add `Filters.group` and `Filters.private`

• Compatibility with GAE via `urllib3.contrib` package - https://github.com/python-telegram-bot/python-telegram-bot/pull/583

• Add equality rich comparison operators to `telegram` objects - https://github.com/python-telegram-bot/python-telegram-bot/pull/604

• Several bugfixes and other improvements

• Remove some deprecated code

2017-04-17

*Released 5.3.1*

• Hotfix release due to bug introduced by urllib3 version 1.21

2016-12-11

*Released 5.3*

• Implement API changes of November 21st (Bot API 2.3)

• `JobQueue` now supports `datetime.timedelta` in addition to seconds

• `JobQueue` now supports running jobs only on certain days

• New `Filters.reply` filter

• Bugfix for `Message.edit_reply_markup`

• Other bugfixes

2016-10-25

*Released 5.2*

• Implement API changes of October 3rd (games update)

• Add `Message.edit_*` methods

• Filters for the `MessageHandler` can now be combined using bitwise operators (`&` and `|`)

• Add a way to save user- and chat-related data temporarily

• Other bugfixes and improvements

2016-09-24

*Released 5.1*

• Drop Python 2.6 support

• Deprecate `telegram.Emoji`

• Use `ujson` if available

• Add instance methods to `Message`, `Chat`, `User`, `InlineQuery` and `CallbackQuery`

• RegEx filtering for `CallbackQueryHandler` and `InlineQueryHandler`
• New MessageHandler filters: forwarded and entity
• Add Message.get_entity to correctly handle UTF-16 codepoints and MessageEntity offsets
• Fix bug in ConversationHandler when first handler ends the conversation
• Allow multiple Dispatcher instances
• Add ChatMigrated Exception
• Properly split and handle arguments in CommandHandler

2016-07-15
released 5.0
• Rework JobQueue
• Introduce ConversationHandler
• Introduce telegram.constants - https://github.com/python-telegram-bot/python-telegram-bot/pull/342

2016-07-12
released 4.3.4
• Fix proxy support with urllib3 when proxy requires auth

2016-07-08
released 4.3.3
• Fix proxy support with urllib3

2016-07-04
released 4.3.2
• Fix: Use timeout parameter in all API methods

2016-06-29
released 4.3.1
• Update wrong requirement: urllib3>=1.10

2016-06-28
released 4.3
• Use urllib3.PoolManager for connection re-use
• Rewrite run_async decorator to re-use threads
• New requirements: urllib3 and certifi

2016-06-10
released 4.2.1
• Fix CallbackQuery.to_dict() bug (thanks to @jlmadurga)
• Fix editMessageText exception when receiving a CallbackQuery

2016-05-28
released 4.2
• Implement Bot API 2.1
• Move botan module to telegram.contrib
• New exception type: BadRequest
2016-05-22
Released 4.1.2
  • Fix MessageEntity decoding with Bot API 2.1 changes

2016-05-16
Released 4.1.1
  • Fix deprecation warning in Dispatcher

2016-05-15
Released 4.1
  • Implement API changes from May 6, 2016
  • Fix bug when start_polling with clean=True
  • Methods now have snake_case equivalent, for example telegram.Bot.send_message is the same as telegram.Bot.sendMessage

2016-05-01
Released 4.0.3
  • Add missing attribute location to InlineQuery

2016-04-29
Released 4.0.2
  • Bugfixes
  • KeyboardReplyMarkup now accepts str again

2016-04-27
Released 4.0.1
  • Implement Bot API 2.0
  • Almost complete recode of Dispatcher
  • Please read the Transition Guide to 4.0
  • Changes from 4.0rc1
    – The syntax of filters for MessageHandler (upper/lower cases)
    – Handler groups are now identified by int only, and ordered
  • Note: v4.0 has been skipped due to a PyPI accident

2016-04-22
Released 4.0rc1
  • Implement Bot API 2.0
  • Almost complete recode of Dispatcher
  • Please read the Transition Guide to 4.0

2016-03-22
Released 3.4
  • Move Updater, Dispatcher and JobQueue to new telegram.ext submodule (thanks to @rahiel)
  • Add disable_notification parameter (thanks to @aidarbiktimirov)
  • Fix bug where commands sent by Telegram Web would not be recognized (thanks to @shelomentsevd)
  • Add option to skip old updates on bot startup
• Send files from BufferedReader

2016-02-28

Released 3.3

• Inline bots
• Send any file by URL
• Specialized exceptions: Unauthorized, InvalidToken, NetworkError and TimedOut
• Integration for botan.io (thanks to @ollmer)
• HTML Parsemode (thanks to @jlmadurga)
• Bugfixes and under-the-hood improvements

Very special thanks to Noam Meltzer (@tsnoam) for all of his work!

2016-01-09

Released 3.3b1

• Implement inline bots (beta)

2016-01-05

Released 3.2.0

• Introducing JobQueue (original author: @franciscod)
• Streamlining all exceptions to TelegramError (Special thanks to @tsnoam)
• Proper locking of Updater and Dispatcher start and stop methods
• Small bugfixes

2015-12-29

Released 3.1.2

• Fix custom path for file downloads
• Don’t stop the dispatcher thread on uncaught errors in handlers

2015-12-21

Released 3.1.1

• Fix a bug where asynchronous handlers could not have additional arguments
• Add groups and groupdict as additional arguments for regex-based handlers

2015-12-16

Released 3.1.0

• The chat-field in Message is now of type Chat. (API update Oct 8 2015)
• Message now contains the optional fields supergroup_chat_created, migrate_to_chat_id, migrate_from_chat_id and channel_chat_created. (API update Nov 2015)

2015-12-08

Released 3.0.0

• Introducing the Updater and Dispatcher classes

2015-11-11

Released 2.9.2

• Error handling on request timeouts has been improved
2015-11-10
Released 2.9.1

- Add parameter `network_delay` to `Bot.getUpdates` for slow connections

2015-11-10
Released 2.9

- Emoji class now uses `bytes_to_native_str` from future 3rd party lib
- Make `user_from` optional to work with channels
- Raise exception if Telegram times out on long-polling

*Special thanks to @jh0ker for all hard work*

2015-10-08
Released 2.8.7

- Type as optional for `GroupChat` class

2015-10-08
Released 2.8.6

- Adds type to `User` and `GroupChat` classes (pre-release Telegram feature)

2015-09-24
Released 2.8.5

- Handles HTTP Bad Gateway (503) errors on request
- Fixes regression on `Audio` and `Document` for unicode fields

2015-09-20
Released 2.8.4

- `getFile` and `File.download` is now fully supported

2015-09-10
Released 2.8.3

- Moved `Bot._requestURL` to its own class (`telegram.utils.request`)
- Much better, such wow, Telegram Objects tests
- Add consistency for `str` properties on Telegram Objects
- Better design to test if `chat_id` is invalid
- Add ability to set custom filename on `Bot.sendMessage(..., filename='')`
- Fix Sticker as `InputFile`
- Send JSON requests over urlencoded post data
- Markdown support for `Bot.sendMessage(..., parse_mode=ParseMode.MARKDOWN)`
- Refactor of `TelegramError` class (no more handling `IOError` or `URLError`)

2015-09-05
Released 2.8.2

- Fix regression on Telegram ReplyMarkup
- Add certificate to `is_inputfile` method

2015-09-05
Released 2.8.1
• Fix regression on Telegram objects with thumb properties

2015-09-04

Released 2.8
• TelegramError when chat_id is empty for send* methods
• setWebhook now supports sending self-signed certificate
• Huge redesign of existing Telegram classes
• Added support for PyPy
• Added docstring for existing classes

2015-08-19

Released 2.7.1
• Fixed JSON serialization for message

2015-08-17

Released 2.7
• Added support for Voice object and sendVoice method
• Due backward compatibility performer or/and title will be required for sendAudio
• Fixed JSON serialization when forwarded message

2015-08-15

Released 2.6.1
• Fixed parsing image header issue on < Python 2.7.3

2015-08-14

Released 2.6.0
• Depreciation of require_authentication and clearCredentials methods
• Giving AUTHORS the proper credits for their contribution for this project
• Message.date and Message.forward_date are now datetime objects

2015-08-12

Released 2.5.3
• telegram.Bot now supports to be unpickled

2015-08-11

Released 2.5.2
• New changes from Telegram Bot API have been applied
• telegram.Bot now supports to be pickled
• Return empty str instead None when message.text is empty

2015-08-10

Released 2.5.1
• Moved from GPLv2 to LGPLv3

2015-08-09

Released 2.5
• Fixes logging calls in API
2015-08-08

Released 2.4

• Fixes `Emoji` class for Python 3
• PEP8 improvements

2015-08-08

Released 2.3

• Fixes `ForceReply` class
• Remove `logging.basicConfig` from library

2015-07-25

Released 2.2

• Allows `debug=True` when initializing `telegram.Bot`

2015-07-20

Released 2.1

• Fix `to_dict` for `Document` and `Video`

2015-07-19

Released 2.0

• Fixes bugs
• Improves `__str__` over `to_json()`
• Creates abstract class `TelegramObject`

2015-07-15

Released 1.9

• Python 3 officially supported
• PEP8 improvements

2015-07-12

Released 1.8

• Fixes crash when replying an unicode text message (special thanks to JRoot3D)

2015-07-11

Released 1.7

• Fixes crash when `username` is not defined on `chat` (special thanks to JRoot3D)

2015-07-10

Released 1.6

• Improvements for GAE support

2015-07-10

Released 1.5

• Fixes randomly unicode issues when using `InputFile`

2015-07-10

Released 1.4

• `requests` lib is no longer required
• Google App Engine (GAE) is supported
2015-07-10

Released 1.3

- Added support to setWebhook (special thanks to macrojames)

2015-07-09

Released 1.2

- CustomKeyboard classes now available
- Emojis available
- PEP8 improvements

2015-07-08

Released 1.1

- PyPi package now available

2015-07-08

Released 1.0

- Initial checkin of python-telegram-bot
CHAPTER 5

Indices and tables

- genindex
- modindex
- search
Python Module Index

t
telegram.constants, 96
telegram.error, 98
telegram.ext.filters, 8
telegram.utils.helpers, 47
Symbols
__call__() (telegram.ext.DelayQueue method), 18
__call__() (telegram.ext.MessageQueue method), 17
__init__() (telegram.ext.DelayQueue method), 18
__init__() (telegram.ext.MessageQueue method), 17
__weakref__ (telegram.ext.MessageQueue attribute), 17
_queue (telegram.ext.JobQueue attribute), 14

A
add_error_handler() (telegram.ext.Dispatcher method), 6
add_handler() (telegram.ext.Dispatcher method), 6
add_sticker_to_set() (telegram.Bot method), 52
address (telegram.InlineQueryResultVenue attribute), 156, 247
address (telegram.InputVenueMessageContent attribute), 160, 253
address (telegram.SecureData attribute), 173, 299
address (telegram.Venue attribute), 130, 281
addStickerToSet() (telegram.Bot method), 52
ADMINISTRATOR (telegram.ChatMember attribute), 95
all (telegram.ext.filters.Filters attribute), 9
all_members_are_administrators (telegram.Chat attribute), 89, 216
ALL_TYPES (telegram.MessageEntity attribute), 121
allow_edited (telegram.ext.CommandHandler attribute), 27
allow_reentry (telegram.ext.ConversationHandler attribute), 25
allowed_updates (telegram.WebhookInfo attribute), 134, 283
amount (telegram.LabeledPrice attribute), 162, 287
Animation (class in telegram), 50
animation (telegram.ext.filters.Filters attribute), 9
animation (telegram.Game attribute), 168, 284
animation (telegram.Message attribute), 110, 259
answer() (telegram.CallbackQuery method), 88
answer() (telegram.InlineQuery method), 137
answer() (telegram.PreCheckoutQuery method), 167
answer() (telegram.ShippingQuery method), 166
answer_callback_query() (telegram.Bot method), 52
answer_inline_query() (telegram.Bot method), 53
answer_pre_checkout_query() (telegram.Bot method), 54
answer_shipping_query() (telegram.Bot method), 54
answerCallbackQuery() (telegram.Bot method), 52
answerInlineQuery() (telegram.Bot method), 52
answerPreCheckoutQuery() (telegram.Bot method), 52
answerShippingQuery() (telegram.Bot method), 52
args (telegram.ext.CallbackContext attribute), 19
args (telegram.utils.promise.Promise attribute), 48
attach (telegram.InputFile attribute), 102, 251
Audio (class in telegram), 50
audio (telegram.ext.filters.Filters attribute), 9
audio (telegram.Message attribute), 111, 261
audio_duration (telegram.InlineQueryResultAudio attribute), 139, 232
audio_file_id (telegram.InlineQueryResultCachedAudio attribute), 140, 232
audio_url (telegram.InlineQueryResultAudio attribute), 139, 232
author_signature (telegram.Message attribute), 111, 260

B
BadRequest, 98
bank_statement (telegram.SecureData attribute), 173, 299
BaseFilter (class in telegram.ext.filters), 8
BasePersistence (class in telegram.ext), 42
big_file_id (telegram.ChatPhoto attribute), 96, 292
birth_date (telegram.PersonalDetails attribute), 174, 300
BOLD (telegram.MessageEntity attribute), 121
Bot (class in telegram), 51
bot (telegram.Audio attribute), 51, 179
bot (telegram.Document attribute), 97, 225
bot (telegram.EncryptedPassportElement attribute), 177, 256
bot (telegram.ext.CallbackContext attribute), 19
bot (telegram.ext.Dispatcher attribute), 5
bot (telegram.ext.JobQueue attribute), 14
bot (telegram.ext.Updater attribute), 3
bot (telegram.Message attribute), 111, 261
bot (telegram.PassportData attribute), 175, 257
bot (telegram.PassportFile attribute), 176, 255
bot (telegram.PhotoSize attribute), 122, 272
bot (telegram.PreCheckoutQuery attribute), 166, 289
bot (telegram.ShippingQuery attribute), 165, 291
bot (telegram.Sticker attribute), 135, 275
bot (telegram.User attribute), 128, 278
bot (telegram.Video attribute), 131, 281
bot (telegram.VideoNote attribute), 132, 286
bot (telegram.V oice attribute), 133, 282
BOT_COMMAND (telegram.MessageEntity attribute), 121
burst_limit (telegram.ext.DelayQueue attribute), 17
C
callback (telegram.ext.CallbackQueryHandler attribute), 22
callback (telegram.ext.ChosenInlineResultHandler attribute), 23
callback (telegram.ext.CommandHandler attribute), 27
callback (telegram.ext.Handler attribute), 20
callback (telegram.ext.InlineQueryHandler attribute), 29
callback (telegram.ext.Job attribute), 13
callback (telegram.ext.MessageHandler attribute), 31
callback (telegram.ext.PreCheckoutQueryHandler attribute), 33
callback (telegram.ext.PrefixHandler attribute), 34
callback (telegram.ext.ShippingQueryHandler attribute), 36
callback (telegram.ext.StringCommandHandler attribute), 39
callback (telegram.ext.StringRegexHandler attribute), 40
callback (telegram.ext.TypeHandler attribute), 41
callback_data (telegram.InlineKeyboardButton attribute), 101, 228
callback_game (telegram.InlineKeyboardButton attribute), 101, 228
callback_query (telegram.Update attribute), 126, 277
CallbackContext (class in telegram.ext), 19
CallbackGame (class in telegram), 169
CallbackQuery (class in telegram), 87
CallbackQueryHandler (class in telegram.ext), 21
can_add_web_page_previews (telegram.ChatMember attribute), 94, 220
can_be_edited (telegram.ChatMember attribute), 94, 220
can_change_info (telegram.ChatMember attribute), 94, 220
can_delete_messages (telegram.ChatMember attribute), 94, 220
can_edit_messages (telegram.ChatMember attribute), 94, 220
can_invite_users (telegram.ChatMember attribute), 94, 220
can_pin_messages (telegram.ChatMember attribute), 94, 220
can_post_messages (telegram.ChatMember attribute), 94, 220
can_promote_members (telegram.ChatMember attribute), 94, 220
can_restrict_members (telegram.ChatMember attribute), 94, 220
can_send_media_messages (telegram.ChatMember attribute), 94, 220
can_send_messages (telegram.ChatMember attribute), 94, 220
can_send_other_messages (telegram.ChatMember attribute), 94, 220
can_set_sticker_set (telegram.Chat attribute), 90, 216
caption (telegram.InlineQueryResultAudio attribute), 140, 232
caption (telegram.InlineQueryResultCachedAudio attribute), 141, 233
caption (telegram.InlineQueryResultCachedDocument attribute), 141, 234
caption (telegram.InlineQueryResultCachedGif attribute), 142, 235
caption (telegram.InlineQueryResultCachedMpeg4Gif attribute), 143, 236
caption (telegram.InlineQueryResultCachedPhoto attribute), 144, 237
caption (telegram.InlineQueryResultCachedVideo attribute), 146, 238
caption (telegram.InlineQueryResultCachedVoice attribute), 147, 239
caption (telegram.InlineQueryResultDocument attribute), 149, 241
caption (telegram.InlineQueryResultGif attribute), 151, 243
caption (telegram.InlineQueryResultMpeg4Gif attribute), 153, 245
caption (telegram.InlineQueryResultPhoto attribute), 154, 246
caption (telegram.InlineQueryResultVideo attribute), 157, 248
caption (telegram.InlineQueryResultVoice attribute), 158, 250
caption (telegram.InputMediaAnimation attribute), 103, 302
caption (telegram.InputMediaAudio attribute), 104, 302
caption (telegram.InputMediaDocument attribute), 105, 303
caption (telegram.InputMediaPhoto attribute), 106, 294
caption (telegram.InputMediaVideo attribute), 107, 294, 300
caption (telegram.Message attribute), 110, 260
caption_entities (telegram.Message attribute), 110, 259
caption_html (telegram.Message attribute), 113
caption_html(urled (telegram.Message attribute), 113
caption_markdown (telegram.Message attribute), 113
caption_markdown(urled (telegram.Message attribute), 114
CASHTAG (telegram.MessageEntity attribute), 121
CHANNEL (telegram.Chat attribute), 90
Index
E
edit_caption() (telegram.Message method), 114
edit_date (telegram.Message attribute), 109, 259
edit_media() (telegram.Message method), 114
edit_message_caption() (telegram.Bot method), 58
edit_message_caption() (telegram.CallbackQuery method), 88
edit_message_live_location() (telegram.Bot method), 58
edit_message_media() (telegram.Bot method), 59
date (telegram.Message attribute), 109, 259
date (telegram.Update attribute), 126, 277
date_post (telegram.Update attribute), 126, 276
date_updates (telegram.MessageHandler attribute), 31
deMessageCaption() (telegram.Bot method), 57
delMessageLiveLocation() (telegram.Bot method), 57
delMessageReplyMarkup() (telegram.Bot method), 58
delMessageText() (telegram.Bot method), 58
effective_attachment (telegram.Message attribute), 115
effective_chat (telegram.Update attribute), 127
effective_message (telegram.Update attribute), 127
effective_message_type() (in module telegram.utils.helpers), 47
effective_user (telegram.Update attribute), 127
effectors (telegram.Message attribute), 109, 259
email (telegram.EncryptedPassportElement attribute), 177, 256
EMAIL (telegram.MessageEntity attribute), 121
email (telegram.OrderInfo attribute), 163, 290
emoji (telegram.Sticker attribute), 135, 275
enabled (telegram.ext.Job attribute), 14
EncryptedCredentials (class in telegram), 178
EncryptedPassportElement (class in telegram), 176
END (telegram.ext.ConversationHandler attribute), 26
eocode_conversations_to_json() (in module telegram.utils.helpers), 47
eties (telegram.Message attribute), 109, 259
entry_points (telegram.ext.ConversationHandler attribute), 25
eror (telegram.ext.CallbackContext attribute), 19
eror_handlers (telegram.ext.Dispatcher attribute), 7
escape_markdown() (in module telegram.utils.helpers), 47
exc_route (telegram.ext.DelayQueue attribute), 17
exception (telegram.utils.promise.Promise attribute), 48
expiry_date (telegram.IdDocumentData attribute), 174, 299
export_chat_invite_link() (telegram.Bot method), 60
exportChatInviteLink() (telegram.Bot method), 60
EYES (telegram.MaskPosition attribute), 136
F
fallbacks (telegram.ext.ConversationHandler attribute), 25
field_name (telegram.PassportElementErrorDataField attribute), 172, 297
File (class in telegram), 99
file_date (telegram.PassportFile attribute), 176, 255
file_hash (telegram.PassportElementErrorFile attribute), 170, 295, 297
file_hash (telegram.PassportElementErrorFiles attribute), 171, 297
file_hash (telegram.PassportElementErrorFrontSide attribute), 171, 296
file_hash (telegram.PassportElementErrorReverseSide attribute), 170, 296
file_hash (telegram.PassportElementErrorSelfie attribute), 304
file_hash (telegram.PassportElementErrorTranslationFile attribute), 305
file_hash (telegram.PassportElementErrorTranslationFiles attribute), 305
file_id (telegram.Animation attribute), 50, 283
file_id (telegram.Audio attribute), 50, 179
file_id (telegram.Document attribute), 97, 225
file_id (telegram.File attribute), 99, 226
file_id (telegram.PassportFile attribute), 176, 255
file_id (telegram.PhotoSize attribute), 122, 271
file_id (telegram.Sticker attribute), 134, 275
file_id (telegram.Video attribute), 131, 281
file_id (telegram.VideoNote attribute), 132, 286
file_id (telegram.Voice attribute), 133, 282
file_name (telegram.Animation attribute), 50, 284
file_name (telegram.Document attribute), 97, 225
file_path (telegram.File attribute), 99, 226
file_size (telegram.Animation attribute), 50, 284
file_size (telegram.Audio attribute), 51, 179
file_size (telegram.Document attribute), 97, 225
file_size (telegram.File attribute), 99, 226
file_size (telegram.PassportFile attribute), 176, 255
file_size (telegram.PhotoSize attribute), 122, 271
file_size (telegram.Sticker attribute), 135, 275
file_size (telegram.Video attribute), 131, 281
file_size (telegram.VideoNote attribute), 132, 286
FileCredentials (class in telegram), 173
filename (telegram.ext.PicklePersistence attribute), 44
filename (telegram.InputFile attribute), 102, 251
files (telegram.EncryptedPassportElement attribute), 177, 256
filter() (telegram.ext.filters.BaseFilter method), 9
filter() (telegram.ext.filters.InvertedFilter method), 12
filter() (telegram.ext.filters.MergedFilter method), 13
Filters (class in telegram.ext.filters), 9
filters (telegram.ext.CommandHandler attribute), 27
filters (telegram.ext.MessageHandler attribute), 31
filters (telegram.ext.PredicateHandler attribute), 34
Filters.captions_entity (class in telegram.ext.filters), 9
Filters.chat (class in telegram.ext.filters), 9
Filters.entity (class in telegram.ext.filters), 10
Filters.language (class in telegram.ext.filters), 10
Filters.regex (class in telegram.ext.filters), 11
Filters.user (class in telegram.ext.filters), 12
FIND_LOCATION (telegram.ChatAction attribute), 93
first_name (telegram.Bot attribute), 61
first_name (telegram.Chat attribute), 89, 216
first_name (telegram.Contact attribute), 97, 225
first_name (telegram.InlineQueryResultContact attribute), 148, 240
first_name (telegram.InputContactMessageContent attribute), 161, 251
first_name (telegram.PersonalDetails attribute), 174, 299, 300
first_name (telegram.User attribute), 128, 278
flush() (telegram.ext.BasePersistence method), 43
flush() (telegram.ext.PicklePersistence method), 44
force_reply (telegram.ForceReply attribute), 100, 227
ForceReply (class in telegram), 100
FOREHEAD (telegram.MaskPosition attribute), 137
forward() (telegram.Message method), 115
forward_date (telegram.Message attribute), 109, 259
forward_from (telegram.Message attribute), 109, 259
forward_from_chat (telegram.Message attribute), 109, 259
forward_from_message_id (telegram.Message attribute), 109, 259
forward_from_message_id (telegram.Message attribute), 109, 259
forward_message() (telegram.Bot method), 61
forward_signature (telegram.Message attribute), 111, 260
forwarded (telegram.ext.filters.Filters attribute), 10
forwardMessage() (telegram.Bot method), 61
foursquare_id (telegram.InlineQueryResultVenue attribute), 156, 247
foursquare_id (telegram.InlineQueryResultVenue attribute), 156, 247
foursquare_id (telegram.InlineQueryResultVenue attribute), 156, 247
foursquare_id (telegram.InlineQueryResultVenue attribute), 156, 247
foursquare_type (telegram.Venue attribute), 131, 281
foursquare_type (telegram.Venue attribute), 131, 281
from_button() (telegram.InlineKeyboardMarkup class method), 102
from_button() (telegram.ReplyKeyboardMarkup class method), 124
from_call() (telegram.InlineKeyboardMarkup class method), 102
from_call() (telegram.ReplyKeyboardMarkup class method), 125
from_row() (telegram.InlineKeyboardMarkup class method), 102
from_row() (telegram.ReplyKeyboardMarkup class method), 125
from_timestamp() (in module telegram.utils.helpers), 47
from_user (telegram.CallbackQuery attribute), 87, 223
from_user (telegram.ChosenInlineResult attribute), 161, 222
from_user (telegram.InlineQuery attribute), 137, 230
from_user (telegram.Message attribute), 109, 259
from_user (telegram.PreCheckoutQuery attribute), 166, 289
from_user (telegram.ShippingQuery attribute), 165, 291
full_name (telegram.User attribute), 128
G
Game (class in telegram), 167
game (telegram.ext.filters.Filters attribute), 10
game (telegram.Message attribute), 110, 259
game_short_name (telegram.CallbackQuery attribute), 87, 223
game_short_name (telegram.InlineQueryResultGame attribute), 150, 250
GameHighScore (class in telegram), 169
gender (telegram.PersonalDetails attribute), 174, 300
get() (telegram.utils.request.Request method), 49
get_administrators() (telegram.Chat method), 90
get_chat() (telegram.Bot method), 62
get_chat_administrators() (telegram.Bot method), 62
get_chat_data() (telegram.ext.BasePersistence method), 43
get_chat_data() (telegram.ext.DictPersistence method), 46
get_chat_data() (telegram.ext.PicklePersistence method), 44
get_chat_member() (telegram.Bot method), 62
get_chat_members_count() (telegram.Bot method), 63
get_conversations() (telegram.ext.BasePersistence method), 43
get_conversations() (telegram.ext.DictPersistence method), 46
get_conversations() (telegram.ext.PicklePersistence method), 44
get_file() (telegram.Audio method), 51
get_file() (telegram.Bot method), 63
get_file() (telegram.Document method), 98
get_file() (telegram.PassportFile method), 176
get_file() (telegram.PhotoSize method), 122
get_file() (telegram.Sticker method), 135
get_file() (telegram.Video method), 132
<table>
<thead>
<tr>
<th>Method/Attribute</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_file()</td>
<td>132</td>
</tr>
<tr>
<td>get_file()</td>
<td>133</td>
</tr>
<tr>
<td>get_game_high_scores()</td>
<td>63</td>
</tr>
<tr>
<td>get_instance()</td>
<td>7</td>
</tr>
<tr>
<td>get_jobs_by_name()</td>
<td>14</td>
</tr>
<tr>
<td>get_me()</td>
<td>64</td>
</tr>
<tr>
<td>get_member()</td>
<td>91</td>
</tr>
<tr>
<td>get_members_count()</td>
<td>91</td>
</tr>
<tr>
<td>get_profile_photos()</td>
<td>128</td>
</tr>
<tr>
<td>get_signal_name()</td>
<td>47</td>
</tr>
<tr>
<td>get_sticker_set()</td>
<td>64</td>
</tr>
<tr>
<td>get_updates()</td>
<td>66</td>
</tr>
<tr>
<td>get_user_data()</td>
<td>43</td>
</tr>
<tr>
<td>get_user_data()</td>
<td>46</td>
</tr>
<tr>
<td>get_user_data()</td>
<td>45</td>
</tr>
<tr>
<td>get_webhook_info()</td>
<td>65</td>
</tr>
<tr>
<td>get_chat()</td>
<td>61</td>
</tr>
<tr>
<td>get_chatAdministrators()</td>
<td>61</td>
</tr>
<tr>
<td>get_chatMember()</td>
<td>61</td>
</tr>
<tr>
<td>get_chatMembersCount()</td>
<td>61</td>
</tr>
<tr>
<td>get_file()</td>
<td>66</td>
</tr>
<tr>
<td>get_gameHighScores()</td>
<td>62</td>
</tr>
<tr>
<td>getMe()</td>
<td>62</td>
</tr>
<tr>
<td>getStickerSet()</td>
<td>62</td>
</tr>
<tr>
<td>getUpdates()</td>
<td>62</td>
</tr>
<tr>
<td>getUserProfilePhotos()</td>
<td>62</td>
</tr>
<tr>
<td>gif_duration</td>
<td>151, 243</td>
</tr>
<tr>
<td>gif_file_id</td>
<td>142, 235</td>
</tr>
<tr>
<td>gif_height</td>
<td>151, 242</td>
</tr>
<tr>
<td>gif_url</td>
<td>151, 242</td>
</tr>
<tr>
<td>gif_width</td>
<td>151, 242</td>
</tr>
<tr>
<td>group</td>
<td>10</td>
</tr>
<tr>
<td>group_chat_created</td>
<td>110</td>
</tr>
<tr>
<td>groups</td>
<td>7</td>
</tr>
<tr>
<td>handle_update()</td>
<td>26</td>
</tr>
<tr>
<td>has_custom_certificate</td>
<td>134, 283</td>
</tr>
</tbody>
</table>

**Index**

- id (telegram.Bot attribute), 65
- id (telegram.CallbackQuery attribute), 87, 223
- id (telegram.Chat attribute), 89, 215
- id (telegram.InlineQuery attribute), 137, 230
- id (telegram.InlineQueryResult attribute), 138, 231
- id (telegram.InlineQueryResultArticle attribute), 138, 231
- id (telegram.InlineQueryResultAudio attribute), 139, 232
- id (telegram.InlineQueryResultCachedAudio attribute), 140, 233
- id (telegram.InlineQueryResultCachedDocument attribute), 141, 234
- id (telegram.InlineQueryResultCachedGif attribute), 142, 235
- id (telegram.InlineQueryResultCachedMpeg4Gif attribute), 143, 236
- id (telegram.InlineQueryResultCachedPhoto attribute), 144, 237
- id (telegram.InlineQueryResultCachedSticker attribute), 145, 238
- id (telegram.InlineQueryResultCachedVideo attribute), 146, 238
- id (telegram.InlineQueryResultCachedVoice attribute), 147, 239
- id (telegram.InlineQueryResultDocument attribute), 149, 241
- id (telegram.InlineQueryResultGame attribute), 150, 250
- id (telegram.InlineQueryResultGif attribute), 151, 242
- id (telegram.InlineQueryResultLocation attribute), 152, 243
- id (telegram.InlineQueryResultMpeg4Gif attribute), 153, 245
- id (telegram.InlineQueryResultPhoto attribute), 154, 246
InputMediaPhoto (class in telegram), 106
InputMediaVideo (class in telegram), 106
InputMessageContent (class in telegram), 159
InputTextMessageContent (class in telegram), 159
InputVenueMessageContent (class in telegram), 160
internal_passport (telegram.SecureData attribute), 173, 298
interval (telegram.ext.Job attribute), 14
interval_seconds (telegram.ext.Job attribute), 14
InvalidToken, 98
InvertedFilter (class in telegram.ext.filters), 12
invite_link (telegram.Chat attribute), 89, 216
Invoice (class in telegram), 162
invoice (telegram.ext.filters.Filters attribute), 10
invoice (telegram.Message attribute), 111, 260
invoice_payload (telegram.PreCheckoutQuery attribute), 166, 289
invoice_payload (telegram.ShippingQuery attribute), 165, 291
invoice_payload (telegram.SuccessfulPayment attribute), 164, 287
is_bot (telegram.User attribute), 128, 278
is_image() (telegram.InputFile static method), 103
ITALIC (telegram.MessageEntity attribute), 121
J
Job (class in telegram.ext), 13
job (telegram.ext.CallbackContext attribute), 19
job_queue (telegram.ext.CallbackContext attribute), 19
job_queue (telegram.ext.Dispatcher attribute), 6
job_queue (telegram.ext.Job attribute), 14
job_queue (telegram.ext.Updater attribute), 3
JobQueue (class in telegram.ext), 14
jobs() (telegram.ext.JobQueue method), 14
K
keyboard (telegram.ReplyKeyboardMarkup attribute), 123, 273
KeyboardButton (class in telegram), 108
kick_chat_member() (telegram.Bot method), 65
kick_member() (telegram.Chat method), 91
kickChatMember() (telegram.Bot method), 65
KICKED (telegram.ChatMember attribute), 95
kwargs (telegram.utils.promise.Promise attribute), 48
L
label (telegram.LabeledPrice attribute), 162, 287
LabeledPrice (class in telegram), 162
language_code (telegram.User attribute), 128, 278
last_error_date (telegram.WebhookInfo attribute), 134, 283
last_error_message (telegram.WebhookInfo attribute), 134, 283
last_name (telegram.Bot attribute), 66
last_name (telegram.Chat attribute), 89, 216
last_name (telegram.Contact attribute), 97, 225
last_name (telegram.InlineQueryResultContact attribute), 148, 240
last_name (telegram.InputContactMessageContent attribute), 161, 251
last_name (telegram.PersonalDetails attribute), 174, 299, 300
last_name (telegram.User attribute), 128, 278
latitude (telegram.InlineQueryResultLocation attribute), 152, 244
latitude (telegram.InlineQueryResultVenue attribute), 155, 247
latitude (telegram.InputLocationMessageContent attribute), 160, 252
latitude (telegram.InputVenueMessageContent attribute), 160, 253
latitude (telegram.Location attribute), 108, 254
leave() (telegram.Chat method), 91
leave_chat() (telegram.Bot method), 66
leaveChat() (telegram.Bot method), 66
LEFT (telegram.ChatMember attribute), 95
left_chat_member (telegram.ext.filters.Filters attribute), 11
left_chat_member (telegram.Message attribute), 110, 260
length (telegram.MessageEntity attribute), 121, 270
length (telegram.Videonote attribute), 132, 286
link (telegram.Chat attribute), 91
link (telegram.Message attribute), 115
link (telegram.User attribute), 128
live_period (telegram.InlineQueryResultLocation attribute), 152, 244
Location (class in telegram), 108
location (telegram.ChosenInlineResult attribute), 161, 222
location (telegram.ext.filters.Filters attribute), 10
location (telegram.InlineQuery attribute), 137, 230
location (telegram.Message attribute), 110, 260
location (telegram.Venue attribute), 130, 281
longitude (telegram.InlineQueryResultLocation attribute), 152, 244
longitude (telegram.InlineQueryResultVenue attribute), 155, 247
longitude (telegram.InputLocationMessageContent attribute), 160, 252
longitude (telegram.InputVenueMessageContent attribute), 160, 253
longitude (telegram.Location attribute), 108, 254
M
MARKDOWN (telegram.ParseMode attribute), 122
mask_position (telegram.Sticker attribute), 135, 275
MaskPosition (class in telegram), 136
match (telegram.ext.CallbackContext attribute), 19
matches (telegram.ext.CallbackContext attribute), 19
MAX_CAPTION_LENGTH (in module telegram.constants), 96
max_connections (telegram.WebhookInfo attribute), 134, 283
MAX_FILESIZE_DOWNLOAD (in module telegram.constants), 96
MAX_FILESIZE_UPLOAD (in module telegram.constants), 96
MAX_INLINE_QUERY_RESULTS (in module telegram.constants), 96
MAX_MESSAGE_ENTITIES (in module telegram.constants), 97
MAX_MESSAGE_LENGTH (in module telegram.constants), 96
MAX_MESSAGES_PER_MINUTE_PER_GROUP (in module telegram.constants), 96
MAX_MESSAGES_PER_SECOND (in module telegram.constants), 96
MAX_MESSAGES_PER_SECOND_PER_CHAT (in module telegram.constants), 96
media (telegram.InputMediaAnimation attribute), 103, 301
media (telegram.InputMediaAudio attribute), 104, 302
media (telegram.InputMediaDocument attribute), 105, 303
media (telegram.InputMediaPhoto attribute), 106, 294
media (telegram.InputMediaVideo attribute), 106, 294, 300
media_group_id (telegram.Message attribute), 109, 259
MEMBER (telegram.ChatMember attribute), 95
MENTION (telegram.MessageEntity attribute), 121
mention_html() (in module telegram.utils.helpers), 47
mention_html() (telegram.User method), 128
mention_markdown() (in module telegram.utils.helpers), 47
mention_markdown() (telegram.User method), 128
MergedFilter (class in telegram.ext.filters), 13
Message (class in telegram), 109
message (telegram.CallbackQuery attribute), 87, 223
message (telegram.ext.filters.Filters attribute), 12
message (telegram.PassportElementError attribute), 169, 295
message (telegram.PassportElementErrorDataField attribute), 172, 297
message (telegram.PassportElementErrorFile attribute), 170, 295, 298
message (telegram.PassportElementErrorFiles attribute), 171, 297
message (telegram.PassportElementErrorFrontSide attribute), 171, 296
message (telegram.PassportElementErrorReverseSide attribute), 170, 296
message (telegram.PassportElementErrorSelfie attribute), 304
message (telegram.PassportElementErrorTranslationFile attribute), 305
message (telegram.PassportElementErrorTranslationFiles attribute), 305
message (telegram.PassportElementErrorUnspecified attribute), 305
message (telegram.Update attribute), 126, 276
message_id (telegram.Message attribute), 109, 258
message_text (telegram.InputTextMessageContent attribute), 159, 252
message_updates (telegram.ext.MessageHandler attribute), 31
MessageEntity (class in telegram), 121
MessageHandler (class in telegram.ext), 31
MessageQueue (class in telegram.ext), 16
messages (telegram.ext.filters.Filters attribute), 12
middle_name (telegram.PersonalDetails attribute), 12
migrate (telegram.ext.filters.Filters attribute), 11
migrate_from_chat_id (telegram.Message attribute), 111, 260
migrate_to_chat_id (telegram.Message attribute), 111, 260
mime_type (telegram.Animation attribute), 50, 284
mime_type (telegram.Audio attribute), 51, 179
mime_type (telegram.Document attribute), 97, 225
mime_type (telegram.InlineQueryResultDocument attribute), 149, 241
mime_type (telegram.InlineQueryResultDocument attribute), 157, 248
mime_type (telegram.Video attribute), 131, 281
mime_type (telegram.Voice attribute), 133, 282
MOUTH (telegram.MaskPosition attribute), 137
mpeg4_duration (telegram.InlineQueryResultMpeg4Gif attribute), 153, 245
mpeg4_file_id (telegram.InlineQueryResultCachedMpeg4Gif attribute), 143, 236
mpeg4_height (telegram.InlineQueryResultMpeg4Gif attribute), 153, 245
mpeg4_url (telegram.InlineQueryResultMpeg4Gif attribute), 153, 245
mpeg4_width (telegram.InlineQueryResultMpeg4Gif attribute), 153, 245

N
name (telegram.Bot attribute), 66
name (telegram.ext.ConversationHandler attribute), 25
name (telegram.ext.DelayQueue attribute), 18
name (telegram.ext.filters.BaseFilter attribute), 8
name (telegram.ext.Job attribute), 13
name (telegram.OrderInfo attribute), 163, 290
name (telegram.StickerSet attribute), 136, 292
name (telegram.User attribute), 128
NetworkError, 98
new_chat_members (telegram.ext.filters.Filters attribute), 11
new_chat_members (telegram.Message attribute), 110, 260
new_chat_photo (telegram.ext.filters.Filters attribute), 11
new_chat_photo (telegram.Message attribute), 110, 260
new_chat_title (telegram.ext.filters.Filters attribute), 11
new_chat_title (telegram.Message attribute), 110, 260
nonce (telegram.Credentials attribute), 172, 298

O
offset (telegram.InlineQuery attribute), 137, 230
Python Telegram Bot Documentation, Release 12.0.0b1

Index 341

offset (telegram.MessageEntity attribute), 121, 270
on_flush (telegram.ext.PicklePersistence attribute), 44
one_time_keyboard (telegram.ReplyKeyboardMarkup attribute), 123, 273
order_info (telegram.PreCheckoutQuery attribute), 166, 289
order_info (telegram.SuccessfulPayment attribute), 164, 287
OrderInfo (class in telegram), 163

P
parse_caption_entities() (telegram.Message method), 115
parse_caption_entity() (telegram.Message method), 116
parse_entities() (telegram.Message method), 116
parse_mode (telegram.InlineQueryResultAudio attribute), 140, 232
parse_mode (telegram.InlineQueryResultCachedAudio attribute), 141, 233
parse_mode (telegram.InlineQueryResultCachedDocument attribute), 142, 234
parse_mode (telegram.InlineQueryResultCachedGif attribute), 142, 235
parse_mode (telegram.InlineQueryResultCachedMpeg4Gif attribute), 143, 236
parse_mode (telegram.InlineQueryResultCachedPhoto attribute), 144, 237
parse_mode (telegram.InlineQueryResultCachedVideo attribute), 146, 238
parse_mode (telegram.InlineQueryResultCachedVoice attribute), 147, 239
parse_mode (telegram.InlineQueryResultDocument attribute), 149, 241
parse_mode (telegram.InlineQueryResultGif attribute), 151, 243
parse_mode (telegram.InlineQueryResultMpeg4Gif attribute), 153, 245
parse_mode (telegram.InlineQueryResultPhoto attribute), 154, 246
parse_mode (telegram.InlineQueryResultVideo attribute), 157, 248
parse_mode (telegram.InlineQueryResultVoice attribute), 158, 250
parse_mode (telegram.InputMediaAnimation attribute), 103, 302
parse_mode (telegram.InputMediaAudio attribute), 104, 302
parse_mode (telegram.InputMediaDocument attribute), 105, 303
parse_mode (telegram.InputMediaPhoto attribute), 106, 294
parse_mode (telegram.InputMediaVideo attribute), 107, 294, 300
parse_mode (telegram.InputTextMessageContent attribute), 159, 252
parse_text_entities() (telegram.Game method), 168
parse_text_entity() (telegram.Game method), 168
ParseMode (class in telegram), 122
pass_args (telegram.ext.CommandHandler attribute), 27
pass_args (telegram.ext.PrefixHandler attribute), 34
pass_args (telegram.ext.StringCommandHandler attribute), 39
pass_chat_data (telegram.ext.CallbackQueryHandler attribute), 22
pass_chat_data (telegram.ext.Chi1nedInlineResultHandler attribute), 24
pass_chat_data (telegram.ext.CommandHandler attribute), 27
pass_chat_data (telegram.ext.Handler attribute), 20
pass_chat_data (telegram.ext.InlineQueryHandler attribute), 29
pass_chat_data (telegram.ext.MessageHandler attribute), 31
pass_chat_data (telegram.ext.PreCheckoutQueryHandler attribute), 33
pass_chat_data (telegram.ext.ReplaceCommandHandler attribute), 36
pass_chat_data (telegram.ext.RegexHandler attribute), 36
pass_groups (telegram.ext.CallbackQueryHandler attribute), 22
pass_groups (telegram.ext.InlineQueryHandler attribute), 29
pass_groups (telegram.ext.RegexHandler attribute), 36
pass_groups (telegram.ext.StringRegexHandler attribute), 40
pass_job_queue (telegram.ext.CallbackQueryHandler attribute), 22
pass_job_queue (telegram.ext.InlineQueryHandler attribute), 29
pass_job_queue (telegram.ext.MessageHandler attribute), 31
pass_job_queue (telegram.ext.PrefixHandler attribute), 34
pass_job_queue (telegram.ext.PreCheckoutQueryHandler attribute), 33
pass_job_queue (telegram.ext.RegexHandler attribute), 36
pass_job_queue (telegram.ext.ShippingQueryHandler attribute), 37
pass_job_queue (telegram.ext.StringCommandHandler attribute), 39
pass_job_queue (telegram.ext.StringRegexHandler attribute), 40
pass_job_queue (telegram.ext.TypeHandler attribute), 42
pass_update_queue (telegram.ext.CallbackQueryHandler attribute), 22
pass_update_queue (telegram.ext.ChosenInlineResultHandler attribute), 23
pass_update_queue (telegram.ext.CommandHandler attribute), 27
pass_update_queue (telegram.ext.Handler attribute), 20
pass_update_queue (telegram.ext.InlineQueryHandler attribute), 29
pass_update_queue (telegram.ext.MessageHandler attribute), 31
pass_update_queue (telegram.ext.PreCheckoutQueryHandler attribute), 33
pass_update_queue (telegram.extPREFIXHandler attribute), 34
pass_update_queue (telegram.ext.RegexHandler attribute), 36
pass_update_queue (telegram.ext.ShippingQueryHandler attribute), 37
pass_update_queue (telegram.ext.StringCommandHandler attribute), 39
pass_update_queue (telegram.ext.StringRegexHandler attribute), 40
pass_user_data (telegram.ext.CallbackQueryHandler attribute), 22
pass_user_data (telegram.ext.ChosenInlineResultHandler attribute), 24
pass_user_data (telegram.ext.CommandHandler attribute), 27
pass_user_data (telegram.ext.Handler attribute), 20
pass_user_data (telegram.ext.InlineQueryHandler attribute), 29
pass_user_data (telegram.ext.MessageHandler attribute), 31
pass_user_data (telegram.ext.PreCheckoutQueryHandler attribute), 33
pass_user_data (telegram.ext.PREFIXHandler attribute), 34
pass_user_data (telegram.ext.RegexHandler attribute), 36
pass_user_data (telegram.ext.ShippingQueryHandler attribute), 38
passport (telegram.SecureData attribute), 173, 298
passport_data (telegram.ext.Filters.Filters attribute), 10
passport_data (telegram.Message attribute), 111, 260
passport_registration (telegram.SecureData attribute), 173, 299
PassportData (class in telegram), 175
PassportElementError (class in telegram), 169
PassportElementErrorDataField (class in telegram), 171
PassportElementErrorFile (class in telegram), 170
PassportElementErrorFiles (class in telegram), 171
PassportElementErrorFrontSide (class in telegram), 170
PassportElementErrorReverseSide (class in telegram), 170
PassportFile (class in telegram), 176
pattern (telegram.ext.CallbackQueryHandler attribute), 22
pattern (telegram.ext.InlineQueryHandler attribute), 29
pattern (telegram.ext.RegexHandler attribute), 36
pattern (telegram.ext.StringRegexHandler attribute), 40
pay (telegram.InlineKeyboardButton attribute), 101, 228
pending_update_count (telegram.WebhookInfo attribute), 134, 283
per_chat (telegram.ext.ConversationHandler attribute), 25
per_message (telegram.ext.ConversationHandler attribute), 25
per_user (telegram.ext.ConversationHandler attribute), 25
performer (telegram.Audio attribute), 50, 179
performer (telegram.InlineQueryResultAudio attribute), 140, 232
performer (telegram.InputMediaAudio attribute), 104, 303
persistence (telegram.ext.ConversationHandler attribute), 27
persistence (telegram.ext.Dispatcher attribute), 6
persistence (telegram.ext.Updater attribute), 3
persistent (telegram.ext.ConversationHandler attribute), 25
personal_details (telegram.SecureData attribute), 173, 298
PersonalDetails (class in telegram), 174
phone_number (telegram.Contact attribute), 97, 225
phone_number (telegram.EncryptedPassportElement attribute), 177, 256
phone_number (telegram.InlineQueryResultContact attribute), 148, 240
phone_number (telegram.InputContactMessageContent attribute), 161, 251
PHONE_NUMBER (telegram.MessageEntity attribute), 121
phone_number (telegram.OrderInfo attribute), 163, 290
photo (telegram.Chat attribute), 89, 216
photo (telegram.ext.Filters.Filters attribute), 10
photo (telegram.Game attribute), 167, 284
photo (telegram.Message attribute), 110, 259
photo_file_id (telegram.InlineQueryResultCachedPhoto attribute), 144, 237
photo_height (telegram.InlineQueryResultPhoto attribute), 154, 246
photo_url (telegram.InlineQueryResultPhoto attribute), 154, 246
photo_width (telegram.InlineQueryResultPhoto attribute), 154, 246
photos (telegram.UserProfilePhotos attribute), 130, 280
PhotoSize (class in telegram), 122
PicklePersistence (class in telegram.ext), 44
pin_chat_message() (telegram.Bot method), 66
pinChatMessage() (telegram.Bot method), 66
pinned_message (telegram.Chat attribute), 89, 216
pinned_message (telegram.ext.filters.Filters attribute), 11
pinned_message (telegram.Message attribute), 111, 260
point (telegram.MaskPosition attribute), 136, 293
pooled_function (telegram.util.promise.Promise attribute), 48
position (telegram.GameHighScore attribute), 169, 286
post() (telegram.util.request.Request method), 49
post_code (telegram.ResidentialAddress attribute), 175, 300
post_code (telegram.ShippingAddress attribute), 163, 288
PRE (telegram.MessageEntity attribute), 121
pre_checkout_query (telegram.Update attribute), 127, 277
PreCheckoutQuery (class in telegram), 166
PreCheckoutQueryHandler (class in telegram.ext), 32
prefix (telegram.ext.PrefixHandler attribute), 34
PrefixHandler (class in telegram.ext), 34
prices (telegram.ShippingOption attribute), 164, 288
PRIVATE (telegram.Chat attribute), 90
private (telegram.ext.filters.Filters attribute), 10
process_update() (telegram.ext.Dispatcher method), 7
Promise (class in telegram.util.promise), 48
promote_chat_member() (telegram.Bot method), 67
promoteChatMember() (telegram.Bot method), 67
provider_payment_charge_id (telegram.SuccessfulPayment attribute), 165, 287
Q
query (telegram.ChosenInlineResult attribute), 161, 222
query (telegram.InlineQuery attribute), 137, 230
R
RECORD_AUDIO (telegram.ChatAction attribute), 93
RECORD_VIDEO (telegram.ChatAction attribute), 93
RECORD_VIDEO_NOTE (telegram.ChatAction attribute), 93
RegexHandler (class in telegram.ext), 36
remove_error_handler() (telegram.ext.Dispatcher method), 7
remove_handler() (telegram.ext.Dispatcher method), 7
remove_keyboard (telegram.ReplyKeyboardRemove attribute), 123, 272
removed (telegram.ext.Job attribute), 14
rental_agreement (telegram.SecureData attribute), 173, 299
repeat (telegram.ext.Job attribute), 14
reply (telegram.ext.filters.Filters attribute), 11
reply_animation() (telegram.Message method), 117
reply_audio() (telegram.Message method), 117
reply_contact() (telegram.Message method), 117
reply_document() (telegram.Message method), 117
reply_html() (telegram.Message method), 118
reply_location() (telegram.Message method), 118
reply_markdown() (telegram.Message method), 118
reply_markup (telegram.InlineQueryResultArticle attribute), 139, 231
reply_markup (telegram.InlineQueryResultAudio attribute), 140, 233
reply_markup (telegram.InlineQueryResultCachedAudio attribute), 141, 233
reply_markup (telegram.InlineQueryResultCachedDocument attribute), 142, 234
reply_markup (telegram.InlineQueryResultCachedGif attribute), 143, 235
reply_markup (telegram.InlineQueryResultCachedMpeg4Gif attribute), 143, 236
reply_markup (telegram.InlineQueryResultCachedPhoto attribute), 144, 237
reply_markup (telegram.InlineQueryResultCachedSticker attribute), 145, 238
reply_markup (telegram.InlineQueryResultCachedVideo attribute), 146, 239
reply_markup (telegram.InlineQueryResultContact attribute), 148, 240
reply_markup (telegram.InlineQueryResultDocument attribute), 149, 241
reply_markup (telegram.InlineQueryResultGame attribute), 150, 250
reply_markup (telegram.InlineQueryResultGif attribute), 151, 243
reply_markup (telegram.InlineQueryResultLocation attribute), 152, 244
reply_markup (telegram.InlineQueryResultMpeg4Gif attribute), 153, 245
reply_markup (telegram.InlineQueryResultPhoto attribute), 155, 246
reply_markup (telegram.InlineQueryResultVenue attribute), 156, 247
reply_markup (telegram.InlineQueryResultVideo attribute), 157, 249
reply_markup (telegram.InlineQueryResultVoice attribute), 158, 250
reply_media_group() (telegram.Message method), 118
reply_photo() (telegram.Message method), 119
reply_sticker() (telegram.Message method), 119
reply_text() (telegram.Message method), 119
reply_to_message (telegram.Message attribute), 109, 259
reply_venue() (telegram.Message method), 119
reply_video() (telegram.Message method), 119
reply_video_note() (telegram.Message method), 119
reply_voice() (telegram.Message method), 120
ReplyKeyboardMarkup (class in telegram), 123
ReplyKeyboardRemove (class in telegram), 123
ReplyMarkup (class in telegram), 126
Request (class in telegram.utils.request), 48
request_contact (telegram.KeyboardButton attribute), 108, 253
request_location (telegram.KeyboardButton attribute), 108, 253
residence_country_code (telegram.PersonalDetails attribute), 174, 300
ResidentialAddress (class in telegram), 174
resize_keyboard (telegram.ReplyKeyboardMarkup attribute), 123, 273
restrict_chat_member() (telegram.Bot method), 68
restrictChatMember() (telegram.Bot method), 68
RESTRICTED (telegram.ChatMember attribute), 96
result() (telegram.utils.promise.Promise method), 48
result_id (telegram.ChosenInlineResult attribute), 161, 222
retrieve() (telegram.utils.request.Request method), 49
RetryAfter, 98
reverse_side (telegram.EncryptedPassportElement attribute), 177, 256
run() (telegram.ext.DelayQueue method), 18
run() (telegram.ext.Job method), 14
run() (telegram.utils.promise.Promise method), 48
run_async() (telegram.ext.Dispatcher method), 7
run_daily()(telegram.ext.JobQueue method), 14
run_once() (telegram.ext.JobQueue method), 15
run_repeating() (telegram.ext.JobQueue method), 15
running (telegram.ext.Dispatcher attribute), 8
running (telegram.ext.Updater attribute), 3
S
scale (telegram.MaskPosition attribute), 136, 293
schedule_removal() (telegram.ext.Job method), 14
score (telegram.GameHighScore attribute), 169, 286
secret (telegram.DataCredentials attribute), 172, 298
secret (telegram.EncryptedCredentials attribute), 178, 254
secret (telegram.FileCredentials attribute), 173, 299
secure_data (telegram.Credentials attribute), 172, 298
SecureData (class in telegram), 173
selective (telegram.ForcedReply attribute), 100, 227
selective (telegram.ReplyKeyboardMarkup attribute), 123, 273
selective (telegram.ReplyKeyboardRemove attribute), 123, 272
selfie (telegram.EncryptedPassportElement attribute), 177, 256
send_action() (telegram.Chat method), 91
send_animation() (telegram.Bot method), 69
send_animation() (telegram.Chat method), 91
send_animation() (telegram.User method), 128
send_audio() (telegram.Bot method), 70
send_audio() (telegram.Chat method), 92
send_audio() (telegram.User method), 129
send_chat_action() (telegram.Bot method), 71
send_contact() (telegram.Bot method), 72
send_document() (telegram.Bot method), 72
send_document() (telegram.Chat method), 92
send_document() (telegram.User method), 129
send_game() (telegram.Bot method), 73
send_invoice() (telegram.Bot method), 73
send_location() (telegram.Bot method), 75
send_media_group() (telegram.Bot method), 75
send_message() (telegram.Bot method), 76
send_message() (telegram.Chat method), 92
send_message() (telegram.User method), 129
send_photo() (telegram.Bot method), 76
send_photo() (telegram.Chat method), 92
send_photo() (telegram.User method), 129
send_sticker() (telegram.Bot method), 77
send_sticker() (telegram.Chat method), 92
send_sticker() (telegram.User method), 129
send_venue() (telegram.Bot method), 78
send_video() (telegram.Bot method), 79
send_video() (telegram.Chat method), 92
send_video() (telegram.User method), 129
send_video_note() (telegram.Bot method), 79
send_video_note() (telegram.Chat method), 92
send_video_note() (telegram.User method), 130
send_voice() (telegram.Bot method), 80
send_voice() (telegram.Chat method), 93
send_voice() (telegram.User method), 130
sendAnimation() (telegram.Bot method), 68
sendAudio() (telegram.Bot method), 68
sendChatAction() (telegram.Bot method), 68
sendContact() (telegram.Bot method), 68
sendDocument() (telegram.Bot method), 69
sendGame() (telegram.Bot method), 69
sendInvoice() (telegram.Bot method), 69
sendLocation() (telegram.Bot method), 69
sendMediaGroup() (telegram.Bot method), 69
sendMessage() (telegram.Bot method), 69
sendPhoto() (telegram.Bot method), 69
sendSticker() (telegram.Bot method), 69
sendVenue() (telegram.Bot method), 69
sendVideo() (telegram.Bot method), 69
sendVideoNote() (telegram.Bot method), 69
sendVoice() (telegram.Bot method), 69
set_chat_description() (telegram.Bot method), 81
set_chat_photo() (telegram.Bot method), 82
set_chat_sticker_set() (telegram.Bot method), 82
set_chat_title() (telegram.Bot method), 82
set_game_score() (telegram.Bot method), 83
text (telegram.Message attribute), 109, 259
text_entities (telegram.Game attribute), 168, 284
text_html (telegram.Message attribute), 120
text_html_urlencoded (telegram.Message attribute), 120
TEXT_LINK (telegram.MessageEntity attribute), 122
text_markdown (telegram.Message attribute), 120
text_markdown_urlencoded (telegram.Message attribute), 120
TEXT_MENTION (telegram.MessageEntity attribute), 122
thumb (telegram.Animation attribute), 50, 284
thumb (telegram.Audio attribute), 51, 179
thumb (telegram.Document attribute), 97, 225
thumb (telegram.InputMediaAnimation attribute), 103, 301
thumb (telegram.InputMediaAudio attribute), 105, 303
thumb (telegram.InputMediaDocument attribute), 105, 304
thumb (telegram.InputMediaVideo attribute), 107, 294, 301
thumb (telegram.Sticker attribute), 135, 275
thumb (telegram.Video attribute), 131, 281
thumb (telegram.VideoNote attribute), 132, 286
thumb_height (telegram.InlineQueryResultArticle attribute), 139, 232
thumb_height (telegram.InlineQueryResultContact attribute), 148, 240
thumb_height (telegram.InlineQueryResultDocument attribute), 149, 242
thumb_height (telegram.InlineQueryResultLocation attribute), 152, 244
thumb_height (telegram.InlineQueryResultVenue attribute), 156, 247
thumb_url (telegram.InlineQueryResultArticle attribute), 139, 231
thumb_url (telegram.InlineQueryResultContact attribute), 148, 240
thumb_url (telegram.InlineQueryResultDocument attribute), 149, 242
thumb_url (telegram.InlineQueryResultLocation attribute), 152, 244
thumb_url (telegram.InlineQueryResultVenue attribute), 156, 247
title (telegram.Audio attribute), 51, 179
title (telegram.Chat attribute), 89, 215
title (telegram.Game attribute), 167, 284
title (telegram.InlineQueryResultArticle attribute), 138, 231
title (telegram.InlineQueryResultAudio attribute), 139, 232
title (telegram.InlineQueryResultCachedDocument attribute), 141, 234
title (telegram.InlineQueryResultCachedGif attribute), 142, 235
title (telegram.InlineQueryResultCachedMpeg4Gif attribute), 143, 236
title (telegram.InlineQueryResultCachedPhoto attribute), 144, 237
title (telegram.InlineQueryResultCachedVideo attribute), 146, 238
title (telegram.InlineQueryResultCachedVoice attribute), 147, 239
title (telegram.InlineQueryResultDocument attribute), 149, 241
title (telegram.InlineQueryResultGif attribute), 151, 243
title (telegram.InlineQueryResultLocation attribute), 152, 244
title (telegram.InlineQueryResultMpeg4Gif attribute), 153, 245
title (telegram.InlineQueryResultPhoto attribute), 154, 246
title (telegram.InlineQueryResultVenue attribute), 155, 247
title (telegram.InlineQueryResultVideo attribute), 157, 248
title (telegram.InlineQueryResultVoice attribute), 158, 249
title (telegram.InputMediaAudio attribute), 104, 303
title (telegram.InlineQueryResultDocument attribute), 152, 244
title (telegram.InlineQueryResultMpeg4Gif attribute), 153, 245
title (telegram.InlineQueryResultPhoto attribute), 154, 246
title (telegram.InlineQueryResultVenue attribute), 156, 247
title (telegram.InlineQueryResultVideo attribute), 157, 248
tick() (telegram.ext.JobQueue method), 16
time_limit (telegram.ext.DelayQueue attribute), 17
TimedOut, 98
TIMEOUT (telegram.ext.ConversationHandler attribute), 26
title (telegram.Audio attribute), 51, 179
thumb_width (telegram.InlineQueryResultVenue attribute), 156, 247
translation (telegram.EncryptedPassportElement attribute), 177, 256
type (telegram.Chat attribute), 89, 215
type (telegram.EncryptedPassportElement attribute), 176, 256
type (telegram.ext.TypeHandler attribute), 41
type (telegram.InlineQueryResult attribute), 138, 231
type (telegram.InlineQueryResultArticle attribute), 138, 231
type (telegram.InlineQueryResultAudio attribute), 139, 232
type (telegram.InlineQueryResultCachedAudio attribute), 140, 233
type (telegram.InlineQueryResultCachedDocument attribute), 141, 234
type (telegram.InlineQueryResultCachedGif attribute), 142, 235
type (telegram.InlineQueryResultCachedMpeg4Gif attribute), 143, 236
type (telegram.InlineQueryResultCachedPhoto attribute), 144, 237
type (telegram.InlineQueryResultCachedSticker attribute), 145, 238
type (telegram.InlineQueryResultCachedVideo attribute), 146, 238
type (telegram.InlineQueryResultCachedVoice attribute), 147, 239
type (telegram.InlineQueryResultContact attribute), 148, 240
type (telegram.InlineQueryResultDocument attribute), 149, 241
type (telegram.InlineQueryResultGame attribute), 150, 250
type (telegram.InlineQueryResultGif attribute), 151, 242
type (telegram.InlineQueryResultLocation attribute), 152, 243
type (telegram.InlineQueryResultMpeg4Gif attribute), 153, 245
type (telegram.InlineQueryResultPhoto attribute), 154, 246
type (telegram.InlineQueryResultVenue attribute), 155, 247
type (telegram.InlineQueryResultVideo attribute), 156, 248
type (telegram.InlineQueryResultVoice attribute), 158, 249
type (telegram.InputMediaAnimation attribute), 103, 301
type (telegram.InputMediaAudio attribute), 104, 302
type (telegram.InputMediaDocument attribute), 105, 303
type (telegram.InputMediaPhoto attribute), 106, 293
type (telegram.InputMediaVideo attribute), 106, 294, 300
type (telegram.MessageEntity attribute), 121, 270
type (telegram.PassportElementError attribute), 169, 295
type (telegram.PassportElementErrorDataField attribute), 171, 297
type (telegram.PassportElementErrorFile attribute), 170, 295, 297
type (telegram.PassportElementErrorFiles attribute), 171, 297
type (telegram.PassportElementErrorFrontSide attribute), 171, 296
type (telegram.PassportElementErrorReverseSide attribute), 170, 296
type (telegram.PassportElementErrorSelfie attribute), 304
type (telegram.PassportElementErrorTranslationFile attribute), 304
type (telegram.PassportElementErrorTranslationFiles attribute), 305
type (telegram.PassportElementErrorUnspecified attribute), 305
TypeHandler (class in telegram.ext), 41
TYPING (telegram.ChatAction attribute), 93

U
Unauthorized, 99
unban_chat_member() (telegram.Bot method), 85
unban_member() (telegram.Chat method), 93
unbanChatMember() (telegram.Bot method), 85
unpin_chat_message() (telegram.Bot method), 86
unpinChatMessage() (telegram.Bot method), 86
until_date (telegram.ChatMember attribute), 94, 220
Update (class in telegram), 126
update (telegram.ext.filters.Filters attribute), 12
update_chat_data() (telegram.ext.BasePersistence method), 43
update_chat_data() (telegram.ext.DictPersistence method), 46
update_chat_data() (telegram.ext.PicklePersistence method), 45
update_conversation() (telegram.ext.BasePersistence method), 43
update_conversation() (telegram.ext.DictPersistence method), 46
update_conversation() (telegram.ext.PicklePersistence method), 45
update_filter (telegram.ext.filters.BaseFilter attribute), 8
update_id (telegram.Update attribute), 126, 276
update_persistence() (telegram.ext.Dispatcher method), 8
update_queue (telegram.ext.CallbackContext attribute), 20
update_queue (telegram.ext.Dispatcher attribute), 6
update_queue (telegram.ext.Updater attribute), 3
update_user_data() (telegram.ext.BasePersistence method), 43
update_user_data() (telegram.ext.DictPersistence method), 46