<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.23</td>
<td>telegram.ChatMemberRestricted</td>
</tr>
<tr>
<td>3.2.24</td>
<td>telegram.ChatMemberLeft</td>
</tr>
<tr>
<td>3.2.25</td>
<td>telegram.ChatMemberBanned</td>
</tr>
<tr>
<td>3.2.26</td>
<td>telegram.ChatMemberUpdated</td>
</tr>
<tr>
<td>3.2.27</td>
<td>telegram.ChatPermissions</td>
</tr>
<tr>
<td>3.2.28</td>
<td>telegram.ChatPhoto</td>
</tr>
<tr>
<td>3.2.29</td>
<td>telegram.constants Module</td>
</tr>
<tr>
<td>3.2.30</td>
<td>telegram.Contact</td>
</tr>
<tr>
<td>3.2.31</td>
<td>telegram.Dice</td>
</tr>
<tr>
<td>3.2.32</td>
<td>telegram.Document</td>
</tr>
<tr>
<td>3.2.33</td>
<td>telegram.error module</td>
</tr>
<tr>
<td>3.2.34</td>
<td>telegram.File</td>
</tr>
<tr>
<td>3.2.35</td>
<td>telegram.ForceReply</td>
</tr>
<tr>
<td>3.2.36</td>
<td>telegram.InlineKeyboardButton</td>
</tr>
<tr>
<td>3.2.37</td>
<td>telegram.InlineKeyboardMarkup</td>
</tr>
<tr>
<td>3.2.38</td>
<td>telegram.InputFile</td>
</tr>
<tr>
<td>3.2.39</td>
<td>telegram.InputMedia</td>
</tr>
<tr>
<td>3.2.40</td>
<td>telegram.InputMediaAnimation</td>
</tr>
<tr>
<td>3.2.41</td>
<td>telegram.InputMediaAudio</td>
</tr>
<tr>
<td>3.2.42</td>
<td>telegram.InputMediaDocument</td>
</tr>
<tr>
<td>3.2.43</td>
<td>telegram.InputMediaPhoto</td>
</tr>
<tr>
<td>3.2.44</td>
<td>telegram.InputMediaVideo</td>
</tr>
<tr>
<td>3.2.45</td>
<td>telegram.KeyboardButton</td>
</tr>
<tr>
<td>3.2.46</td>
<td>telegram.KeyboardButtonPollType</td>
</tr>
<tr>
<td>3.2.47</td>
<td>telegram.Location</td>
</tr>
<tr>
<td>3.2.48</td>
<td>telegram.LoginUrl</td>
</tr>
<tr>
<td>3.2.49</td>
<td>telegram.Message</td>
</tr>
<tr>
<td>3.2.50</td>
<td>telegram.MessageAutoDeleteTimerChanged</td>
</tr>
<tr>
<td>3.2.51</td>
<td>telegram.MessageId</td>
</tr>
<tr>
<td>3.2.52</td>
<td>telegram.MessageEntity</td>
</tr>
<tr>
<td>3.2.53</td>
<td>telegram.ParseMode</td>
</tr>
<tr>
<td>3.2.54</td>
<td>telegram.PhotoSize</td>
</tr>
<tr>
<td>3.2.55</td>
<td>telegram.Poll</td>
</tr>
<tr>
<td>3.2.56</td>
<td>telegram.PollAnswer</td>
</tr>
<tr>
<td>3.2.57</td>
<td>telegram.PollOption</td>
</tr>
<tr>
<td>3.2.58</td>
<td>telegram.ProximityAlertTriggered</td>
</tr>
<tr>
<td>3.2.59</td>
<td>telegram.ReplyKeyboardRemove</td>
</tr>
<tr>
<td>3.2.60</td>
<td>telegram.ReplyKeyboardMarkup</td>
</tr>
<tr>
<td>3.2.61</td>
<td>telegram.ReplyMarkup</td>
</tr>
<tr>
<td>3.2.62</td>
<td>telegram.TelegramObject</td>
</tr>
<tr>
<td>3.2.63</td>
<td>telegram.Update</td>
</tr>
<tr>
<td>3.2.64</td>
<td>telegram.User</td>
</tr>
<tr>
<td>3.2.65</td>
<td>telegram.UserProfilePhotos</td>
</tr>
<tr>
<td>3.2.66</td>
<td>telegram.Venue</td>
</tr>
<tr>
<td>3.2.67</td>
<td>telegram.Video</td>
</tr>
<tr>
<td>3.2.68</td>
<td>telegram.VideoNote</td>
</tr>
<tr>
<td>3.2.69</td>
<td>telegram.Voice</td>
</tr>
<tr>
<td>3.2.70</td>
<td>telegram.VoiceChatStarted</td>
</tr>
<tr>
<td>3.2.71</td>
<td>telegram.VoiceChatEnded</td>
</tr>
<tr>
<td>3.2.72</td>
<td>telegram.VoiceChatScheduled</td>
</tr>
<tr>
<td>3.2.73</td>
<td>telegram.VoiceChatParticipantsInvited</td>
</tr>
<tr>
<td>3.2.74</td>
<td>telegram.WebhookInfo</td>
</tr>
<tr>
<td>3.2.75</td>
<td>Stickers</td>
</tr>
<tr>
<td>3.2.76</td>
<td>Inline Mode</td>
</tr>
<tr>
<td>3.2.77</td>
<td>Payments</td>
</tr>
<tr>
<td>3.2.78</td>
<td>Games</td>
</tr>
<tr>
<td>3.2.79</td>
<td>Passport</td>
</tr>
<tr>
<td>3.2.80</td>
<td>utils</td>
</tr>
</tbody>
</table>
GUIDES AND TUTORIALS

If you’re just starting out with the library, we recommend following our “Your first Bot” tutorial that you can find on our wiki. On our wiki you will also find guides like how to use handlers, webhooks, emoji, proxies and much more.
A great way to learn is by looking at examples. Ours can be found in our examples folder on Github.
CHAPTER THREE
REFERENCE

Below you can find a reference of all the classes and methods in python-telegram-bot. Apart from the telegram.ext package the objects should reflect the types defined in the official Telegram Bot API documentation.

3.1 telegram.ext package

3.1.1 telegram.ext.ExtBot

class telegram.ext.ExtBot(token, base_url=None, base_file_url=None, request=None, private_key=None, private_key_password=None, defaults=None, arbitrary_callback_data=False)

Bases: telegram.bot.Bot

This object represents a Telegram Bot with convenience extensions.

[Warning: Not to be confused with telegram.Bot.]

For the documentation of the arguments, methods and attributes, please see telegram.Bot.

New in version 13.6.

Parameters

• defaults (telegram.ext.Defaults, optional) – An object containing default values to be used if not set explicitly in the bot methods.

• arbitrary_callback_data (bool|int, optional) – Whether to allow arbitrary objects as callback data for telegram.InlineKeyboardButton. Pass an integer to specify the maximum number of objects cached in memory. For more details, please see our wiki. Defaults to False.

arbitrary_callback_data

Whether this bot instance allows to use arbitrary objects as callback data for telegram.InlineKeyboardButton.

Type bool|int

callback_data_cache

The cache for objects passed as callback data for telegram.InlineKeyboardButton.

Type telegram.ext.CallbackDataCache

insert_callback_data (self, update)

If this bot allows for arbitrary callback data, this inserts the cached data into all corresponding buttons within this update.
Note: Checks `telegram.Message.via_bot` and `telegram.Message.from_user` to check if the reply markup (if any) was actually sent by this bot. If it was not, the message will be returned unchanged.

Note that this will fail for channel posts, as `telegram.Message.from_user` is `None` for those! In the corresponding reply markups the callback data will be replaced by `telegram.ext.InvalidCallbackData`.

**Warning:** *In place*, i.e. the passed `telegram.Message` will be changed!

**Parameters**
```
```

### 3.1.2 telegram.ext.Updater

**class telegram.ext.Updater**
```
(token=None, base_url=None, workers=4, bot=None, 
private_key=None, private_key_password=None, 
user_sig_handler=None, request_kwargs=None, pers- 
sistence=None, defaults=None, use_context=True, 
dispatcher=None, base_file_url=None, arbitrary 
callback_data=False, context_types=None)
```

**Bases:** Generic
```
[telegram.ext.utils.types.CCT, telegram.ext.utils.types.UD, 
telegram.ext.utils.types.CD, telegram.ext.utils.types.BD]
```

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.

**Note:**
- You must supply either a `bot` or a `token` argument.
- If you supply a `bot`, you will need to pass `arbitrary_callback_data` and `defaults` to the bot instead of the `telegram.ext.Updater`. In this case, you’ll have to use the class `telegram.ext.ExtBot`.

Changed in version 13.6.

**Parameters**

- `token` *(str, optional)* – The bot’s token given by the @BotFather.
- `base_url` *(str, optional)* – Base_url for the bot.
- `base_file_url` *(str, optional)* – Base_file_url for the bot.
- `workers` *(int, optional)* – Amount of threads in the thread pool for functions decorated with @run_async (ignored if `dispatcher` argument is used).
- `bot` *(telegram.Bot, optional)* – A pre-initialized bot instance (ignored if `dispatcher` argument is used). 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.
• **dispatcher** (*telegram.ext.Dispatcher, optional*) – A pre-initialized dispatcher instance. If a pre-initialized dispatcher is used, it is the user’s responsibility to create it with proper arguments.

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

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

• **user_sig_handler** (*function, optional*) – Takes signum, frame as positional arguments. This will be called when a signal is received, defaults are (SIGINT, SIGTERM, SIGABRT) settable with `idle`.

• **request_kwargs** (*dict, optional*) – Keyword args to control the creation of a `telegram.utils.request.Request` object (ignored if `bot` or `dispatcher` 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 uses the context based callback API (ignored if `dispatcher` argument is used). Defaults to True. **New users:** set this to True.

• **persistence** (*telegram.ext.BasePersistence, optional*) – The persistence class to store data that should be persistent over restarts (ignored if `dispatcher` argument is used).

• **defaults** (*telegram.ext.Defaults, optional*) – An object containing default values to be used if not set explicitly in the bot methods.

• **arbitrary_callback_data** (*bool | int | None, optional*) – Whether to allow arbitrary objects as callback data for `telegram.InlineKeyboardButton`. Pass an integer to specify the maximum number of cached objects. For more details, please see our wiki. Defaults to False.
  
  New in version 13.6.

• **context_types** (*telegram.ext.ContextTypes, optional*) – Pass an instance of `telegram.ext.ContextTypes` to customize the types used in the context interface. If not passed, the defaults documented in `telegram.ext.ContextTypes` will be used.

  New in version 13.6.

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

**bot**

The bot used with this Updater.

**Type** `telegram.Bot`

**user_sig_handler**

Optional. Function to be called when a signal is received.

**Type** `function`

**update_queue**

Queue for the updates.

**Type** `Queue`

**job_queue**

Jobqueue for the updater.

**Type** `telegram.ext.JobQueue`

**dispatcher**

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

**Type** `telegram.ext.Dispatcher`
running
  Indicates if the updater is running.

  Type  bool

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

  Type  telegram.ext.BasePersistence

use_context
  Optional. True if using context based callbacks.

  Type  bool

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 (list | tuple) – List containing signals from the signal module that should be subscribed to. Updater.stop() will be called on receiving one of those signals. Defaults to (SIGINT, SIGTERM, SIGABRT).

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

  • drop_pending_updates (bool, optional) – Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is False.

  New in version 13.4.

  • clean (bool, optional) – Alias for drop_pending_updates.

  Deprecated since version 13.4: Use drop_pending_updates instead.

  • bootstrap_retries (int, optional) – Whether the bootstrapping phase of the telegram.ext.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=None, bootstrap_retries=0, webhook_url=None, allowed_updates=None, force_event_loop=None, drop_pending_updates=None, ip_address=None, max_connections=40)
  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. Also calls telegram.Bot.set_webhook() as required.
Changed in version 13.4: `start_webhook()` now always calls `telegram.Bot.set_webhook()`, so pass `webhook_url` instead of calling `updater.bot.set_webhook(webhook_url)` manually.

**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.
- **drop_pending_updates** (bool, optional) – Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is `False`. New in version 13.4.
- **clean** (bool, optional) – Alias for `drop_pending_updates`. Deprecated since version 13.4: Use `drop_pending_updates` instead.
- **bootstrap_retries** (int, optional) – Whether the bootstrapping phase of the `telegram.ext.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`.
- **ip_address** (str, optional) – Passed to `telegram.Bot.set_webhook()`.
  New in version 13.4.
- **allowed_updates** (List[str], optional) – Passed to `telegram.Bot.set_webhook()`.
- **force_event_loop** (bool, optional) – Legacy parameter formerly used for a workaround on Windows + Python 3.8+. No longer has any effect.
  Deprecated since version 13.6: Since version 13.6, `tornado>=6.1` is required, which resolves the former issue.
- **max_connections** (int, optional) – Passed to `telegram.Bot.set_webhook()`.
  New in version 13.6.

**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.
3.1.3 `telegram.ext.Dispatcher`

```python
class telegram.ext.Dispatcher(bot, update_queue, workers=4, exception_event=None, job_queue=None, persistence=None, use_context=True, context_types=None)
```

Bases: `Generic[telegram.ext.utils.types.CCT, telegram.ext.utils.types.UD, telegram.ext.utils.types.CD, telegram.ext.utils.types.BD]`

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

**Parameters**

- `bot`(telegram.Bot) – The bot object that should be passed to the handlers.
- `update_queue`(Queue) – The synchronized queue that will contain the updates.
- `workers`(int, optional) – Number of maximum concurrent worker threads for the `@run_async` decorator and `run_async()`. 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` uses the context based callback API (ignored if `dispatcher` argument is used). Defaults to `True`. **New users**: set this to `True`.
- `context_types`(telegram.ext.ContextTypes, optional) – Pass an instance of `telegram.ext.ContextTypes` to customize the types used in the context interface. If not passed, the defaults documented in `telegram.ext.ContextTypes` will be used.

New in version 13.6.

**bot**

The bot object that should be passed to the handlers.

Type `telegram.Bot`

**update_queue**

The synchronized queue that will contain the updates.

Type `Queue`

**job_queue**

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

Type `telegram.ext.JobQueue`

**workers**

Number of maximum concurrent worker threads for the `@run_async` decorator and `run_async()`.

Type `int`, optional

**user_data**

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

Type `defaultdict`

**chat_data**

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

Type `defaultdict`

**bot_data**

A dictionary handlers can use to store data for the bot.
Type `dict`

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

```python
Type telegram.ext.BasePersistence
```

```python
context_types
```
Container for the types used in the `context` interface.

New in version 13.6.

```python
Type telegram.ext.ContextTypes
```

```python
add_error_handler(callback, run_async=False)
```
Registers an error handler in the Dispatcher. This handler will receive every error which happens in your bot.

**Note:** Attempts to add the same callback multiple times will be ignored.

**Warning:** The errors handled within these handlers won’t show up in the logger, so you need to make sure that you reraise the error.

**Parameters**

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

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

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

• `run_async` *(bool, optional)* – Whether this handlers callback should be run asynchronously using `run_async()`. Defaults to `False`.

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

```python
add_handler(handler, group=0)
```
Register a handler.

**TL;DR:** Order and priority counts. 0 or 1 handlers per group will be used. End handling of update with `telegram.ext.DispatcherHandlerStop`.

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, promise=None)*

Dispatches an error.

**Parameters**

- **update** *(object | telegram.Update)* – The update that caused the error.
- **error** *(Exception)* – The error that was raised.
- **promise** *(telegram.utils.Promise, optional)* – The promise whose pooled function raised the error.

**error_handlers**: Dict[Callable, Union[bool, telegram.utils.helpers.DefaultValue]]

A dict, where the keys are error handlers and the values indicate whether they are to be run asynchronously.

**Type** Dict[callable, bool]

**classmethod get_instance()**

Get the singleton instance of this class.

**Returns** telegram.ext.Dispatcher

**Raises** RuntimeError –

**groups**: List[int]

A list with all groups.

**Type** List[int]

**handlers**: Dict[int, List[telegram.ext.handler.Handler]]

Holds the handlers per group.

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

**process_update** *(update)*

Processes a single update and updates the persistence.

**Note**: If the update is handled by least one synchronously running handlers (i.e. run_async=False), **update_persistence()** is called *once* after all handlers synchronous handlers are done. Each asynchronously running handler will trigger **update_persistence()** on its own.

**Parameters**


**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, update=None, **kwargs)*

Queue a function (with given args/kwargs) to be run asynchronously. Exceptions raised by the function will be handled by the error handlers registered with **add_error_handler()**.
Warning:

• If you’re using @run_async/`run_async()` you cannot rely on adding custom attributes to `telegram.ext.CallbackContext`. See its docs for more info.

• Calling a function through `run_async()` from within an error handler can lead to an infinite error handling loop.

Parameters

- `func` (callable) – The function to run in the thread.
- `*args` (tuple, optional) – Arguments to `func`.
- `update` (telegram.Update | object, optional) – The update associated with the functions call. If passed, it will be available in the error handlers, in case an exception is raised by `func`.
- `**kwargs` (dict, optional) – Keyword arguments to `func`.

Returns

Promise

`running`

Indicates if this dispatcher is running.

Type bool

`start` (ready=None)

Thread target of thread ‘dispatcher’.

Runs in background and processes the update queue.

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

`stop`()

Stops the thread.

`update_persistence` (update=None)

Update user_data, chat_data and bot_data in persistence.

Parameters update (telegram.Update, optional) – The update to process. If passed, only the corresponding user_data and chat_data will be updated.

3.1.4 telegram.ext.DispatcherHandlerStop

class telegram.ext.DispatcherHandlerStop (state=None)

Bases: Exception

Raise this in handler to prevent execution of any other handler (even in different group).

In order to use this exception in a `telegram.ext.ConversationHandler`, pass the optional state parameter instead of returning the next state:

```python
def callback(update, context):
    ...
    raise DispatcherHandlerStop(next_state)
```

`state`

Optional. The next state of the conversation.

Type object

Parameters state (object, optional) – The next state of the conversation.
3.1.5 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.

Parameters dispatcher (telegram.ext.Dispatcher) – The dispatcher associated with this context.

matches

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. Note that filters short circuit, so combined regex filters will not always be evaluated.

Type List[re match object]

args

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.

Type List[str]

error

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

Type Exception

async_args

Optional. Positional arguments of the function that raised the error. Only present when the raising function was run asynchronously using telegram.ext.Dispatcher.run_async().

Type List[object]

async_kargs

Optional. Keyword arguments of the function that raised the error. Only present when the raising function was run asynchronously using telegram.ext.Dispatcher.run_async().

Type Dict[str, object]

job

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

Type telegram.ext.Job

Warning: Do not combine custom attributes and @run_async/telegram.ext.Dispatcher.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.
**property bot**
The bot associated with this context.

Type `telegram.Bot`

**property bot_data**
Optional. A dict that can be used to keep any data in. For each update it will be the same dict.

Type `dict`

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

**Warning:** When a group chat migrates to a supergroup, its chat id will change and the `chat_data` needs to be transferred. For details see our wiki page.

Type `dict`

**property dispatcher**
The dispatcher associated with this context.

Type `telegram.ext.Dispatcher`

**drop_callback_data**(callback_query)
Deletes the cached data for the specified callback query.

New in version 13.6.

**Note:** Will not raise exceptions in case the data is not found in the cache. Will raise `KeyError` in case the callback query can not be found in the cache.

Parameters `callback_query`(telegram.CallbackQuery) – The callback query.

Raises `KeyError` | `RuntimeError` – `KeyError`, if the callback query can not be found in the cache and `RuntimeError`, if the bot doesn’t allow for arbitrary callback data.

**classmethod from_error**(update, error, dispatcher, async_args=None, async_kwargs=None)
Constructs an instance of `telegram.ext.CallbackContext` to be passed to the error handlers.

See also:

`telegram.ext.Dispatcher.add_error_handler()`

Parameters

- `update`(object | telegram.Update) – The update associated with the error. May be `None`, e.g. for errors in job callbacks.
- `error`(Exception) – The error.
- `dispatcher`(telegram.ext.Dispatcher) – The dispatcher associated with this context.
- `async_args`(List[object]) – Optional. Positional arguments of the function that raised the error. Pass only when the raising function was run asynchronously using `Dispatcher.run_async()`.
- `async_kwargs`(Dict[str, object]) – Optional. Keyword arguments of the function that raised the error. Pass only when the raising function was run asynchronously using `Dispatcher.run_async()`.
Returns `telegram.ext.CallbackContext`

classmethod `from_job(job, dispatcher)`
Constructs an instance of `telegram.ext.CallbackContext` to be passed to a job callback.

See also:
`telegram.ext.JobQueue()`

Parameters
- `job (telegram.ext.Job)` – The job.
- `dispatcher (telegram.ext.Dispatcher)` – The dispatcher associated with this context.

Returns `telegram.ext.CallbackContext`

classmethod `from_update(update, dispatcher)`
Constructs an instance of `telegram.ext.CallbackContext` to be passed to the handlers.

See also:
`telegram.ext.Dispatcher.add_handler()`

Parameters
- `dispatcher (telegram.ext.Dispatcher)` – The dispatcher associated with this context.

Returns `telegram.ext.CallbackContext`

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

Type `telegram.ext.JobQueue`

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

Type `Regex match type`

refresh_data() If `dispatcher` uses persistence, calls `telegram.ext.BasePersistence.refresh_bot_data()` on `bot_data`, `telegram.ext.BasePersistence.refresh_chat_data()` on `chat_data` and `telegram.ext.BasePersistence.refresh_user_data()` on `user_data`, if appropriate.

New in version 13.6.

update(data)
Updates `self.__slots__` with the passed data.

Parameters `data (Dict[str, object])` – The data.

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

Type `queue.Queue`

property `user_data`
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.
**Type** dict

### 3.1.6 telegram.ext.Job

class telegram.ext.Job(callback=None, context=None, name=None, job_queue=None, job=None)

Bases: object

This class is a convenience wrapper for the jobs held in a *telegram.ext.JobQueue*. With the current backend APScheduler, *job* holds a *apscheduler.job.Job* instance.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their id is equal.

**Note:**

- All attributes and instance methods of *job* are also directly available as attributes/methods of the corresponding *telegram.ext.Job* object.
- Two instances of *telegram.ext.Job* are considered equal, if their corresponding *job* attributes have the same id.
- If *job* isn’t passed on initialization, it must be set manually afterwards for this *telegram.ext.Job* to be useful.

**Parameters**

- **callback** (callable) – The callback function that should be executed by the new job. Callback signature for context based API:

  ```python
def callback(CallbackContext)
  a context.job is the *telegram.ext.Job* instance. It can be used to access its job.context or change it to a repeating job.
  ```

- **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__*.

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

- **job** (apscheduler.job.Job, optional) – The APS Job this job is a wrapper for.

**callback**

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

**Type** callable

**context**

Optional. Additional data needed for the callback function.

**Type** object

**name**

Optional. The name of the new job.

**Type** str

**job_queue**

Optional. The *JobQueue* this job belongs to.

**Type** telegram.ext.JobQueue

**job**

Optional. The APS Job this job is a wrapper for.
Type `apscheduler.job.Job`  

**property enabled**  
Whether this job is enabled.  

Type `bool`  

**property next_t**  
Datetime for the next job execution. Datetime is localized according to `tzinfo`. If job is removed or already ran it equals to `None`.  

Type `datetime.datetime`  

**property removed**  
Whether this job is due to be removed.  

Type `bool`  

`run(dispatcher)`  
Executes the callback function independently of the jobs schedule.  

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

### 3.1.7 `telegram.ext.JobQueue`  

**class** `telegram.ext.JobQueue`  
Bases: `object`  

This class allows you to periodically perform tasks with the bot. It is a convenience wrapper for the APScheduler library.  

**scheduler**  
The APScheduler  

Type `apscheduler.schedulers.background.BackgroundScheduler`  

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

Type `telegram.Bot`  

`get_jobs_by_name(name)`  
Returns a tuple of all pending/scheduled jobs with the given name that are currently in the `JobQueue`.  

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

`run_custom(callback, job_kwargs, context=None, name=None)`  
Creates a new customly defined Job.  

**Parameters**  

- `callback (callable)`: The callback function that should be executed by the new job. Callback signature for context based API:  
  
  ```python  
  def callback(Context)  
  context.job is the `telegram.ext.Job` instance. It can be used to access its job.context or change it to a repeating job.  
  
  - `job_kwargs (dict)`: Arbitrary keyword arguments. Used as arguments for `scheduler.add_job`.  
  
  - `context (object, optional)`: Additional data needed for the callback function. Can be accessed through `job.context` in the callback. Defaults to `None`.  

18 Chapter 3. Reference
• **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_daily *(callback, time, days=(0, 1, 2, 3, 4, 5, 6), context=None, name=None, job_kwargs=None)*  
Creates a new `Job` that runs on a daily basis and adds it to the queue.

---

**Note:** For a note about DST, please see the documentation of APScheduler.

**Parameters**

• **callback** *(callable)* – The callback function that should be executed by the new job. Callback signature for context based API:

```python
def callback(CallbackContext)
```

`context.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. If the timezone (time.tzinfo) is None, the default timezone of the bot will be used.

• **days** *(Tuple[int], optional)* – Defines on which days of the week the job should run (where 0-6 correspond to monday - sunday). 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__.

• **job_kwargs** *(dict, optional)* – Arbitrary keyword arguments to pass to the scheduler.add_job().

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

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

run_monthly *(callback, when, day, context=None, name=None, day_is_strict=True, job_kwargs=None)*  
Creates a new `Job` that runs on a monthly basis and adds it to the queue.

**Parameters**

• **callback** *(callable)* – The callback function that should be executed by the new job. Callback signature for context based API:

```python
def callback(CallbackContext)
```

`context.job` is the `telegram.ext.Job` instance. It can be used to access its job.context or change it to a repeating job.

• **when** *(datetime.time)* – Time of day at which the job should run. If the timezone (when.tzinfo) is None, the default timezone of the bot will be used.

• **day** *(int)* – Defines the day of the month whereby the job would run. It should be within the range of 1 and 31, inclusive.

• **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__.
• `day_is_strict` (bool, optional) – If `False` and `day > month.days`, will pick the last day in the month. Defaults to `True`.

• `job_kwargs` (dict, optional) – Arbitrary keyword arguments to pass to the `scheduler.add_job()`.

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, job_kwargs=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. Callback signature for context based API:

     def callback(CallbackContext)

     `context.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. If the timezone (`datetime.tzinfo`) is `None`, the default timezone of the bot will be used.
  - `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. If the timezone (`time.tzinfo`) is `None`, the default timezone of the bot will be used.

• `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__`.

• `job_kwargs` (dict, optional) – Arbitrary keyword arguments to pass to the `scheduler.add_job()`.

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

Return type `telegram.ext.Job`

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

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

Note: For a note about DST, please see the documentation of `APScheduler`.

Parameters

• `callback` (callable) – The callback function that should be executed by the new job. Callback signature for context based API:

     def callback(CallbackContext)
context.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. If the timezone (datetime.tzinfo) is None, the default timezone of the bot will be used.
  - 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. If the timezone (time.tzinfo) is None, the default timezone of the bot will be used.

  Defaults to interval

- **last** (int | float | datetime.timedelta | datetime.datetime | datetime.time, optional) – Latest possible time for the job to run. This parameter will be interpreted depending on its type. See first for details.

  If last is datetime.datetime or datetime.time type and last.tzinfo is None, the default timezone of the bot will be assumed.

  Defaults to None.

- **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__.

- **job_kwargs** (dict, optional) – Arbitrary keyword arguments to pass to the scheduler.add_job().

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

  Return type telegram.ext.Job

**set_dispatcher** (dispatcher)
Set the dispatcher to be used by this JobQueue. Use this instead of passing a telegram.Bot to the JobQueue, which is deprecated.

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

**start**
Starts the job_queue thread.

**stop**
Stops the thread.
3.1.8 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.

Deprecated since version 13.3: telegram.ext.MessageQueue in its current form is deprecated and will be reinvented in a future release. See this thread for a list of known bugs.

Parameters

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

__call__ (promise, is_group_msg=False)

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

Parameters

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

Note: 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.

3.1.9 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 call-
backs with delays.

Deprecated since version 13.3: telegram.ext.DelayQueue in its current form is deprecated and will
be reinvented in a future release. See this thread for a list of known bugs.

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 ob-
                              ject’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.

burst_limit
    Number of maximum callbacks to process per time-window.

    Type int

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

    Type int

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

    Type callable

name
    Thread’s name.
Type str

__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.

3.1.10 telegram.ext.ContextTypes

class telegram.ext.ContextTypes (context=<class 'telegram.ext.callbackcontext.CallbackContext'>, bot_data=<class 'dict'>, chat_data=<class 'dict'>, user_data=<class 'dict'>)

Bases: Generic[telegram.ext.utils.types.CCT, telegram.ext.utils.types.UD, telegram.ext.utils.types.CD, telegram.ext.utils.types.BD]

Convenience class to gather customizable types of the telegram.ext.CallbackContext interface.

New in version 13.6.

Parameters

- **context** (type, optional) – Determines the type of the context argument of all (error-)handler callbacks and job callbacks. Must be a subclass of telegram.ext.CallbackContext. Defaults to telegram.ext.CallbackContext.

- **bot_data** (type, optional) – Determines the type of context.bot_data of all (error-)handler callbacks and job callbacks. Defaults to dict. Must support instantiating without arguments.
- **chat_data** *(type, optional)* – Determines the type of `context.chat_data` of all (error-)handler callbacks and job callbacks. Defaults to `dict`. Must support instantiating without arguments.

- **user_data** *(type, optional)* – Determines the type of `context.user_data` of all (error-)handler callbacks and job callbacks. Defaults to `dict`. Must support instantiating without arguments.

### 3.1.11 `telegram.ext.Defaults`

```python
class telegram.ext.Defaults(parse_mode=None, disable_notification=None, disable_web_page_preview=None, timeout=None, quote=None, tzinfo=<UTC>, run_async=False, allow_sending_without_reply=None)
```

Bases: `object`

Convenience Class to gather all parameters with a (user defined) default value

**Parameters**

- **parse_mode** *(str, optional)* – Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or URLs in your bot’s message.

- **disable_notification** *(bool, optional)* – Sends the message silently. Users will receive a notification with no sound.

- **disable_web_page_preview** *(bool, optional)* – Disables link previews for links in this message.

- **allow_sending_without_reply** *(bool, optional)* – Pass `True`, if the message should be sent even if the specified replied-to message is not found. Default: `True` in group chats and `False` in private chats.

- **timeout** *(int | float, optional)* – If 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:** Will **not** be used for `telegram.Bot.get_updates()`!

- **quote** *(bool, optional)* – If set to `True`, the reply is sent as an actual reply to the 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.

- **tzinfo** *(tzinfo, optional)* – A timezone to be used for all date(time) inputs appearing throughout PTB, i.e. if a timezone naive date(time) object is passed somewhere, it will be assumed to be in `tzinfo`. Must be a timezone provided by the `pytz` module. Defaults to UTC.

- **run_async** *(bool, optional)* – Default setting for the `run_async` parameter of handlers and error handlers registered through `Dispatcher.add_handler()` and `Dispatcher.add_error_handler()`. Defaults to `False`.

**property allow_sending_without_reply**

Optional. Pass `True`, if the message should be sent even if the specified replied-to message is not found.

Type `bool`

**property disable_notification**

Optional. Sends the message silently. Users will receive a notification with no sound.

Type `bool`

**property disable_web_page_preview**

Optional. Disables link previews for links in this message.
Type bool

property explanation_parse_mode
    Optional. Alias for parse_mode, used for the corresponding parameter of telegram.Bot.send_poll().
    Type str

property parse_mode
    Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or URLs in your bot’s message.
    Type str

property quote
    Optional. If set to True, the reply is sent as an actual reply to the 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.
    Type bool

property run_async
    Optional. Default setting for the run_async parameter of handlers and error handlers registered through Dispatcher.add_handler() and Dispatcher.add_error_handler().
    Type bool

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

property tzinfo
    A timezone to be used for all date(time) objects appearing throughout PTB.
    Type tzinfo

3.1.12 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, run_async=False)
    Bases: Generic[telegram.ext.handler.UT, telegram.ext.utils.types.CCT], abc.ABC

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

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.

Warning: When setting run_async to True, you cannot rely on adding custom attributes to telegram.ext.CallbackContext. See its docs 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. DEPRE-
  CATED: 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. DEPRE-
  CATED: 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. DEPRE-
  CATED: Please switch to context based callbacks.

  • **run_async (bool)** – Determines whether the callback will run asynchronously. De-
  faults to False.

  `callback`
  
  The callback function for this handler.

  **Type** callable

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

  **Type** bool

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

  **Type** bool

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

  **Type** bool

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

  **Type** bool

  `run_async`
  
  Determines whether the callback will run asynchronously.

  **Type** bool

  **abstract check_update (update)**
  
  This method is called to determine if an update should be handled by this handler instance. It should
  always be overridden.
Note: Custom updates types can be handled by the dispatcher. Therefore, an implementation of this method should always check the type of `update`.

**Parameters**

`update (str | telegram.Update)`: The update to be tested.

**Returns**

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

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

Prepares additional arguments for the context. Override if needed.

**Parameters**

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

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

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

DEPRECATED: This method is being replaced by new context based callbacks. Please see [https://git.io/fxJuV](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 `callback` along with its respectful arguments. To work with the `telegram.ext.ConversationHandler`, this method returns the value returned from `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 (obj)`: The result from `check_update`.
- `context (telegram.ext.CallbackContext, optional)`: The context as provided by the dispatcher.
```
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, run_async=False)
```

**Bases:** `telegram.ext.handler.Handler[telegram.update.Update, telegram.ext.utils.types.CCT]`

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

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

**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.

- If your bot allows arbitrary objects as `callback_data`, it may happen that the original `callback_data` for the incoming `telegram.CallbackQuery` cannot be found. This is the case when either a malicious client tempered with the `callback_data` or the data was simply dropped from cache or not persisted. In these cases, an instance of `telegram.ext.InvalidCallbackData` will be set as `callback_data`.

  New in version 13.6.

**Warning:** When setting `run_async` to `True`, you cannot rely on adding custom attributes to `telegram.ext.CallbackContext`. See its docs 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 | callable | type, optional)` – Pattern to test `telegram.CallbackQuery.data` against. If a string or a regex pattern is passed, `re.match()` is used on `telegram.CallbackQuery.data` to determine if an up-
date should be handled by this handler. If your bot allows arbitrary objects as `callback_data`, non-strings will be accepted. To filter arbitrary objects you may pass

- a callable, accepting exactly one argument, namely the `telegram.CallbackQuery.data`. It must return True or False/None to indicate whether the update should be handled.

- a type. If `telegram.CallbackQuery.data` is an instance of that type (or a subclass), the update will be handled.

If `telegram.CallbackQuery.data` is None, the `telegram.CallbackQuery` update will not be handled.

Changed in version 13.6: Added support for arbitrary callback data.

- `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.

- `run_async` (bool) – Determines whether the callback will run asynchronously. Defaults to False.

callback

The callback function for this handler.

Type callable

pass_update_queue

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

Type bool

pass_job_queue

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

Type bool

pattern

Optional. Regex pattern, callback or type to test `telegram.CallbackQuery.data` against. Changed in version 13.6: Added support for arbitrary callback data.

Type `Pattern | callable | type`

pass_groups

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

Type bool

pass_groupdict

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

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

Type `bool`

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

Type `bool`

**run_async**
Determines whether the callback will run asynchronously.

Type `bool`

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


Returns `bool`

**collect_additional_context** *(context, update, dispatcher, check_result)*
Add the result of `re.match(pattern, update.callback_query.data)` to `CallbackContext.matches` as list with one element.

**collect_optional_args** *(dispatcher, update=None, check_result=None)*
Pass the results of `re.match(pattern, data).{groups(), groupdict()}` to the callback as a keyword arguments called `groups` and `groupdict`, respectively, if needed.

---

**telegram.ext.ChatJoinRequestHandler**

**class telegram.ext.ChatJoinRequestHandler**(callback, pass_update_queue=False, pass_job_queue=False, pass_user_data=False, pass_chat_data=False, run_async=False)


Handler class to handle Telegram updates that contain a chat join request.

**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.

---

**Warning:** When setting `run_async` to True, you cannot rely on adding custom attributes to `telegram.ext.CallbackContext`. See its docs for more info.

New in version 13.8.

**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.

• **run_async** (bool) – Determines whether the callback will run asynchronously. Defaults to False.

`callback`

The callback function for this handler.

  Type callable

**pass_update_queue**

Determine whether update_queue will be passed to the callback function.

  Type bool

**pass_job_queue**

Determine whether job_queue will be passed to the callback function.

  Type bool

**pass_user_data**

Determine whether user_data will be passed to the callback function.

  Type bool

**pass_chat_data**

Determine whether chat_data will be passed to the callback function.

  Type bool

**run_async**

Determine whether the callback will run asynchronously.

  Type bool

**check_update**(update)

Determines whether an update should be passed to this handler's `callback`.


  Returns bool
telegram.ext.ChatMemberHandler

class telegram.ext.ChatMemberHandler(callback, chat_member_types=-1, pass_update_queue=False, pass_job_queue=False, pass_user_data=False, pass_chat_data=False, run_async=False)


Handler class to handle Telegram updates that contain a chat member update.

New in version 13.4.

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.

Warning: When setting run_async to True, you cannot rely on adding custom attributes to telegram.ext.CallbackContext. See its docs 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.

• chat_member_types (int, optional) – Pass one of MY_CHAT_MEMBER, CHAT_MEMBER or ANY_CHAT_MEMBER to specify if this handler should handle only updates with telegram.Update.my_chat_member, telegram.Update.chat_member or both. Defaults to MY_CHAT_MEMBER.

• 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.

• run_async (bool) – Determines whether the callback will run asynchronously. Defaults to False.
**callback**

The callback function for this handler.

Type `callable`

**chat_member_types**

Specifies if this handler should handle only updates with `teleogram.Update.my_chat_member`, `teleogram.Update.chat_member` or both.

Type `int`, optional

**pass_update_queue**

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

Type `bool`

**pass_job_queue**

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

Type `bool`

**pass_user_data**

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

Type `bool`

**pass_chat_data**

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

Type `bool`

**run_async**

Determines whether the callback will run asynchronously.

Type `bool`

**ANY_CHAT_MEMBER**: ClassVar[int] = 1

Used as a constant to handle bot `teleogram.Update.my_chat_member` and `teleogram.Update.chat_member`.

Type `int`

**CHAT_MEMBER**: ClassVar[int] = 0

Used as a constant to handle only `teleogram.Update.chat_member`.

Type `int`

**MY_CHAT_MEMBER**: ClassVar[int] = -1

Used as a constant to handle only `teleogram.Update.my_chat_member`.

Type `int`

**check_update**(update)

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


Returns `bool`
**telegram.ext.ChosenInlineResultHandler**

```python
class telegram.ext.ChosenInlineResultHandler(callback, pass_update_queue=False,
                                           pass_job_queue=False,
                                           pass_user_data=False,
                                           pass_chat_data=False,
                                           run_async=False, pattern=None)
```

                     utils.types.CCT]`

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

**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](https://git.io/fxJuV) for more info.

**Warning:** When setting `run_async` to True, you cannot rely on adding custom attributes to `telegram.ext.CallbackContext`. See its docs 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.

- `run_async` *(bool)* – Determines whether the callback will run asynchronously. Defaults to False.

- `pattern` *(str | Pattern, optional)* – Regex pattern. If not None, `re.match` is used on `telegram.ChosenInlineResult.result_id` to determine if an update should be handled by this handler. This is accessible in the callback as `telegram.ext.CallbackContext.matches`.

---

3.1. `telegram.ext` package 35
New in version 13.6.

**callback**
The callback function for this handler.

Type: callable

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

Type: bool

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

Type: bool

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

Type: bool

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

Type: bool

**run_async**
Determines whether the callback will run asynchronously.

Type: bool

**pattern**
Optional. Regex pattern to test `telegram.ChosenInlineResult.result_id` against.

New in version 13.6.

Type: Pattern

**check_update** *(update)*
Determines whether an update should be passed to this handler's callback.

Parameters:

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

Returns:

Type: bool

**collect_additional_context** *(context, update, dispatcher, check_result)*
This function adds the matched regex pattern result to `telegram.ext.CallbackContext.matches`.

### telegram.ext.CommandHandler

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


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.
Note:

- `CommandHandler` does *not* handle (edited) channel posts.
- `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.

Warning: When setting `run_async` to `True`, you cannot rely on adding custom attributes to `telegram.ext.CallbackContext`. See its docs for more info.

Parameters

- `command` ([`telegram.utils.types.SLT[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.

• **run_async** (bool) – Determines whether the callback will run asynchronously. Defaults to False.

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

**command**

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

**Type** `telegram.utils.types.SLT[str]`

**callback**

The callback function for this handler.

**Type** `callable`

**filters**

Optional. Only allow updates with these Filters.

**Type** `telegram.ext.BaseFilter`

**allow_edited**

Determines whether the handler should also accept edited messages.

**Type** `bool`

**pass_args**

Determines whether the handler should be passed `args`.

**Type** `bool`

**pass_update_queue**

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

**Type** `bool`

**pass_job_queue**

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

**Type** `bool`

**pass_user_data**

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

**Type** `bool`

**pass_chat_data**

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

**Type** `bool`

**run_async**

Determines whether the callback will run asynchronously.

**Type** `bool`

**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)

Add text after the command to CallbackContext.args as list, split on single whitespaces and add output of data filters to CallbackContext as well.

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

Provide text after the command to the callback the args argument as list, split on single whitespaces.

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, map_to_parent=None, run_async=False)


A handler to hold a conversation with a single or multiple users through Telegram updates by managing four collections of other handlers.

Note: ConversationHandler will only accept updates that are (subclass-)instances of telegram.Update. This is, because depending on the per_user and per_chat ConversationHandler relies on telegram.Update.effective_user and/or telegram.Update.effective_chat in order to determine which conversation an update should belong to. For per_message=True, ConversationHandler uses update.callback_query.message.message_id when per_chat=True and update.callback_query.inline_message_id when per_chat=False. For a more detailed explanation, please see our FAQ.

Finally, ConversationHandler, does not handle (edited) channel posts.

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

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.

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. Finally, telegram.ext.DispatcherHandlerStop can be used in conversations as described in the corresponding documentation.

Note: In each of the described collections of handlers, a handler may in turn be a ConversationHandler. In that case, the nested ConversationHandler should have the attribute map_to_parent which allows to return to the parent conversation at specified states within the nested conversation.

Note that the keys in map_to_parent must not appear as keys in states attribute or else the latter will be ignored. You may map END to one of the parents states to continue the parent conversation after this
has ended or even map a state to `END` to end the `parent` conversation from within the nested one. For an example on nested `ConversationHandler`s, see our examples.

**Parameters**

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

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

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

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

- `per_chat` (bool, optional) – If the 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 and the corresponding context will be handled by ALL the handler’s who’s `check_update` method returns `True` that are in the state `ConversationHandler.TIMEOUT`.

**Note:** Using `conversation_timeout` with nested conversations is currently not supported. You can still try to use it, but it will likely behave differently from what you expect.

- `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`.

- `map_to_parent` (Dict[`object`, `object`], optional) – A dict that can be used to instruct a nested conversation handler to transition into a mapped state on its parent conversation handler in place of a specified nested state.

- `run_async` (bool, optional) – Pass `True` to override the `Handler.run_async` setting of all handlers (in `entry_points`, `states` and `fallbacks`).

**Note:** If set to `True`, you should not pass a handler instance, that needs to be run synchronously in another context.

New in version 13.2.

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

*Type* bool

**run_async**
If True, will override the `Handler.run_async` setting of all internal handlers on initialization.

New in version 13.2.

*Type* bool

**END: ClassVar[int] = -1**
Used as a constant to return when a conversation is ended.

*Type* int

**TIMEOUT: ClassVar[int] = -2**
Used as a constant to handle state when a conversation is timed out.

*Type* int

**WAITING: ClassVar[int] = -3**
Used as a constant to handle state when a conversation is still waiting on the previous `@run_sync` decorated running handler to finish.

*Type* int

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

*Type* bool

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


*Returns* bool

**property conversation_timeout**
Optional. When this handler is inactive more than this timeout (in seconds), it will be automatically ended.

*Type* `float | datetime.timedelta`

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

*Type* List[`telegram.ext.Handler`]

**property fallbacks**
A list of handlers that might be used if the user is in a conversation, but every handler for their current state returned False on `check_update`.

*Type* List[`telegram.ext.Handler`]

**handle_update** *(update, dispatcher, check_result, context=None)*
Send the update to the callback for the current state and Handler

*Parameters*

- `check_result` – The result from `check_update`. For this handler it’s a tuple of key, handler, and the handler’s check result.
- `dispatcher` *(telegram.ext.Dispatcher)* – Dispatcher that originated the Update.
• **context** (*telegram.ext.CallbackContext*, optional) – The context as provided by the dispatcher.

**property map_to_parent**
Optional. A dict that can be used to instruct a nested *ConversationHandler* to transition into a mapped state on its parent *ConversationHandler* in place of a specified nested state.

Type: Dict[object, object]

**property name**
Optional. The name for this *ConversationHandler*.

Type: str

**property per_chat**
If the conversation key should contain the Chat’s ID.

Type: bool

**property per_message**
If the conversation key should contain the message’s ID.

Type: bool

**property per_user**
If the conversation key should contain the User’s ID.

Type: bool

**property persistence**
The persistence class as provided by the *Dispatcher*.

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

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

### 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, run_async=False, chat_types=None)*


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

**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](https://git.io/fxJuV) for more info.

**Warning:**
• When setting *run_async* to True, you cannot rely on adding custom attributes to *telegram.ext.CallbackContext*. See its docs 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.

- **chat_types** (List[str], optional) – List of allowed chat types. If passed, will only handle inline queries with the appropriate `telegram.InlineQuery.chat_type`. New in version 13.5.

- **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.

- **run_async** (bool) – Determines whether the callback will run asynchronously. Defaults to False.

**callback**

The callback function for this handler.

| Type  | callable |

**pass_update_queue**

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

| Type  | bool |

# 3.1. `telegram.ext` package

43
pass_job_queue
Determines whether job_queue will be passed to the callback function.
Type bool

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

chat_types
List of allowed chat types.
New in version 13.5.
Type List[str], optional

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

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

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

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

run_async
Determines whether the callback will run asynchronously.
Type bool

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

collect_additional_context(context, update, dispatcher, check_result)
Add the result of re.match(pattern, update.inline_query.query) to
CallbackContext.matches as list with one element.

collect_optional_args(dispatcher, update=None, check_result=None)
Pass the results of re.match(pattern, query).{groups(), groupdict()} to the call-
back as a keyword arguments called groups and groupdict, respectively, if needed.

telegram.ext.MessageHandler
class telegram.ext.MessageHandler(filters, callback, pass_update_queue=False,
pass_job_queue=False, pass_user_data=False,
pass_chat_data=False, message_updates=None,
channel_post_updates=None, edited_updates=None,
run_async=False)
utils.types.CCT]
Handler class to handle telegram messages. They might contain text, media or status updates.
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.

Warning: When setting **run_async** to True, you cannot rely on adding custom attributes to **telegram.ext.CallbackContext**. See its docs 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 -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.

- **run_async** (bool) – Determines whether the callback will run asynchronously. Defaults to False.

Raises ValueError –
filters
   Only allow updates with these Filters. See `telegram.ext.filters` for a full list of all available filters.
      Type Filter
callback
   The callback function for this handler.
      Type callable
pass_update_queue
   Determines whether `update_queue` will be passed to the callback function.
      Type bool
pass_job_queue
   Determines whether `job_queue` will be passed to the callback function.
      Type bool
pass_user_data
   Determines whether `user_data` will be passed to the callback function.
      Type bool
pass_chat_data
   Determines whether `chat_data` will be passed to the callback function.
      Type bool
message_updates
   Should “normal” message updates be handled? Default is `None`.
      Type bool
channel_post_updates
   Should channel posts updates be handled? Default is `None`.
      Type bool
edited_updates
   Should “edited” message updates be handled? Default is `None`.
      Type bool
run_async
   Determines whether the callback will run asynchronously.
      Type bool
check_update (update)
   Determines whether an update should be passed to this handler’s `callback`.
      Returns bool
collect_additional_context (context, update, dispatcher, check_result)
   Adds possible output of data filters to the `CallbackContext`.
telegram.ext.filters Module

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

class telegram.ext.filters.BaseFilter(*args, **kwargs)

Base class for all Filters.

Filters subclassing from this class can combined using bitwise operators:

And:

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

Or:

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

Exclusive Or:

```python
>>> (Filters.regex('To Be') ^ Filters.regex('Not 2B'))
```

Not:

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

Also works with more than two filters:

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

Note: Filters use the same short circuiting logic as python’s and, or and not. This means that for example:

```python
>>> Filters.regex(r'(a?x)') | Filters.regex(r'(b?x)')
```

With `message.text == x`, will only ever return the matches for the first filter, since the second one is never evaluated.

If you want to create your own filters create a class inheriting from either MessageFilter or UpdateFilter and implement a `filter()` method that returns a boolean: True if the message should be handled, False otherwise. Note that the filters work only as class instances, not actual class objects (so remember to initialize your filter classes).

By default the filters name (what will get printed when converted to a string for display) will be the class name. If you want to overwrite this assign a better name to the `name` class variable.

**name**

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

*Type:* str

**data_filter**

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

*Type:* bool

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

3.1. telegram.ext package 47
Examples

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

```python
all = Filters.all
    All Messages.
animation = Filters.animation
    Messages that contain `telegram.Animation`.
attachment = Filters.attachment
    Messages that contain `telegram.Message.effective_attachment()`.
    New in version 13.6.
audio = Filters.audio
    Messages that contain `telegram.Audio`.
caption = Filters.caption
    Messages with a caption. If a list of strings is passed, it filters messages to only allow those whose caption is appearing in the given list.
```

Examples

```python
MessageHandler(Filters.caption, callback_method)
```

Parameters `update` ([List[str] | Tuple[str], optional] – Which captions to allow. Only exact matches are allowed. If not specified, will allow any message with a caption.

```python
caption_entity(*args, **kwargs)
    Bases: telegram.ext.filters.MessageFilter
    Filters media messages to only allow those which have a `telegram.MessageEntity` where their type matches `entity_type`.
```

Examples

```python
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`.

```python
caption_regex(*args, **kwargs)
    Bases: telegram.ext.filters.MessageFilter
    Filters updates by searching for an occurrence of `pattern` in the message caption.
    This filter works similarly to `Filters.regex`, with the only exception being that it applies to the message caption instead of the text.
```

Examples

```python
Use `MessageHandler(Filters.photo & Filters.caption_regex(r'help'),
callback)` to capture all photos with caption containing the word ‘help’.
```
Note: This filter will not work on simple text messages, but only on media with caption.

Parameters

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

```python
class chat(*args, **kwargs)
Bases: telegram.ext.filters.Filters._ChatUserBaseFilter
Filters messages to allow only those which are from a specified chat ID or username.
```

Examples

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

Warning: `chat_ids` will give a copy of the saved chat ids as frozenset. This is to ensure thread safety. To add/remove a chat, you should use `add_usernames()`, `add_chat_ids()`, `remove_usernames()` and `remove_chat_ids()`. Only update the entire set by `filter`. `chat_ids/usernames` = `new_set`, if you are entirely sure that it is not causing race conditions, as this will complete replace the current set of allowed chats.

```python
Parameters

• `chat_id` *(telegram.utils.types.SLT[int], optional)* – Which chat ID(s) to allow through.

• `username` *(telegram.utils.types.SLT[str], optional)* – Which username(s) to allow through. Leading '@'s in usernames will be discarded.

• `allow_empty` *(bool, optional)* – Whether updates should be processed, if no chat is specified in `chat_ids` and `usernames`. Defaults to False

Raises `RuntimeError` – If `chat_id` and `username` are both present.

```python
chat_ids
Which chat ID(s) to allow through.
Type set(int), optional

usernames
Which username(s) (without leading '@') to allow through.
Type set(str), optional

allow_empty
Whether updates should be processed, if no chat is specified in `chat_ids` and `usernames`. Defaults to False
Type bool, optional

add_chat_ids(chat_id)
Add one or more chats to the allowed chat ids.
Parameters

chat_id *(telegram.utils.types.SLT[int], optional)* – Which chat ID(s) to allow through.

add_usernames(username)
Add one or more chats to the allowed usernames.
Parameters

username *(telegram.utils.types.SLT[str], optional)* – Which username(s) to allow through. Leading '@'s in usernames will be discarded.

get_chat_or_user(message)
```

```python
remove_chat_ids(chat_id)
Remove one or more chats from allowed chat ids.
```
Parameters `chat_id` *(telegram.utils.types.SLT[int], optional)* – Which chat ID(s) to disallow through.

remove_usernames *(username)*
Remove one or more chats from allowed usernames.

Parameters `username` *(telegram.utils.types.SLT[str], optional)* – Which username(s) to disallow through. Leading '@'s in usernames will be discarded.

`chat_type` = `Filters.chat_type`
Subset for filtering the type of chat.

Examples
Use these filters like: `Filters.chat_type.channel` or `Filters.chat_type.supergroup` etc. Or use just `Filters.chat_type` for all chat types.

`channel`
Updates from channel

`group`
Updates from group

`supergroup`
Updates from supergroup

`groups`
Updates from group or supergroup

`private`
Updates sent in private chat

`command` = `Filters.command`
Messages with a `telegram.MessageEntity.BOT_COMMAND`. By default only allows messages starting with a bot command. Pass `False` to also allow messages that contain a bot command anywhere in the text.

Examples:

```
MessageHandler(Filters.command, command_at_start_callback)
MessageHandler(Filters.command(False), command_anywhere_callback)
```

Note: `Filters.text` also accepts messages containing a command.

Parameters `update` *(bool, optional)* – Whether to only allow messages that start with a bot command. Defaults to `True`.

`contact` = `Filters.contact`
Messages that contain `telegram.Contact`.

`dice` = `Filters.dice`
Dice Messages. If an integer or a list of integers is passed, it filters messages to only allow those whose dice value is appearing in the given list.

Examples
To allow any dice message, simply use `MessageHandler(Filters.dice, callback_method)`.

To allow only dice messages with the emoji, but any value, use `MessageHandler(Filters.dice.dice, callback_method)`.
To allow only dice messages with the emoji and with value 6, use `MessageHandler(Filters.dice.darts(6), callback_method).

To allow only dice messages with the emoji and with value 5 or 6, use `MessageHandler(Filters.dice.football([5, 6]), callback_method).

**Note:** Dice messages don’t have text. If you want to filter either text or dice messages, use `Filters.text | Filters.dice`.

**Parameters**

- **update** *(telegram.utils.types.SLT[int], optional)* – Which values to allow. If not specified, will allow any dice message.

**dice**

Dice messages with the emoji . Passing a list of integers is supported just as for `Filters.dice`.

**darts**

Dice messages with the emoji . Passing a list of integers is supported just as for `Filters.dice`.

**basketball**

Dice messages with the emoji . Passing a list of integers is supported just as for `Filters.dice`.

**football**

Dice messages with the emoji . Passing a list of integers is supported just as for `Filters.dice`.

**slot_machine**

Dice messages with the emoji . Passing a list of integers is supported just as for `Filters.dice`.

**bowling**

Dice messages with the emoji . Passing a list of integers is supported just as for `Filters.dice`.

New in version 13.4.

**document = Filters.document**

Subset for messages containing a document/file.

**Examples**

Use these filters like: `Filters.document.mp3`, `Filters.document.mime_type("text/plain")` etc. Or use just `Filters.document` for all document messages.

**category**

Filters documents by their category in the mime-type attribute

**Note:** This Filter only filters by the mime_type of the document, it doesn’t check the validity of the document. The user can manipulate the mime-type of a message and send media with wrong types that don’t fit to this handler.

**Example**

`Filters.document.category('audio/')` filters all types of audio sent as file, for example ‘audio/mpeg’ or ‘audio/x-wav’.

**application**

Same as `Filters.document.category("application")`.

**audio**

Same as `Filters.document.category("audio")`.

---

3.1. telegram.ext package
image
Same as Filters.document.category("image").

video
Same as Filters.document.category("video").

text
Same as Filters.document.category("text").

mime_type
Filters documents by their mime-type attribute

Note: This Filter only filters by the mime_type of the document, it doesn’t check the validity of document.

The user can manipulate the mime-type of a message and send media with wrong types that don’t fit to this handler.

Example
Filters.document.mime_type('audio/mpeg') filters all audio in mp3 format.

apk
Same as Filters.document.mime_type("application/vnd.android.package-archive").

doc
Same as Filters.document.mime_type("application/msword").

docx
Same as Filters.document.mime_type("application/vnd.openxmlformats-officedocument.wordprocessingml.document").

exe
Same as Filters.document.mime_type("application/x-ms-dos-executable").

gif
Same as Filters.document.mime_type("image/mp4").

jpg
Same as Filters.document.mime_type("image/jpeg").

mp3
Same as Filters.document.mime_type("audio/mpeg").

pdf
Same as Filters.document.mime_type("application/pdf").

py
Same as Filters.document.mime_type("text/x-python").

svg
Same as Filters.document.mime_type("image/svg+xml").

txt
Same as Filters.document.mime_type("text/plain").
targz
Same as Filters.document.mime_type("application/x-compressed-tar").

wav
Same as Filters.document.mime_type("audio/x-wav").
xml
Same as Filters.document.mime_type("application/xml").

zip
Same as Filters.document.mime_type("application/zip").

file_extension
This filter filters documents by their file ending/extension.

**Note:**
- This Filter only filters by the file ending/extension of the document, it doesn’t check the validity of document.
- The user can manipulate the file extension of a document and send media with wrong types that don’t fit to this handler.
- Case insensitive by default, you may change this with the flag case_sensitive=True.
- Extension should be passed without leading dot unless it’s a part of the extension.
- Pass None to filter files with no extension, i.e. without a dot in the filename.

**Example**
- Filters.document.file_extension(".jpg") filters files with extension ".jpg".
- Filters.document.file_extension(".jpg") filters files with extension ".jpg".
- Filters.document.file_extension("Dockerfile", case_sensitive=True) filters files with extension ".Dockerfile" minding the case.
- Filters.document.file_extension(None) filters files without a dot in the filename.

class entity(*args, **kwargs)
Bases: telegram.ext.filters.MessageFilter
Filters messages to only allow those which have a telegram.MessageEntity where their type matches entity_type.

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

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

forwarded = Filters.forwarded
Messages that are forwarded.

class forwarded_from(*args, **kwargs)
Bases: telegram.ext.filters.Filters._ChatUserBaseFilter
Filters messages to allow only those which are forwarded from the specified chat ID(s) or username(s) based on telegram.Message.forward_from and telegram.Message.forward_from_chat.

New in version 13.5.

Examples
MessageHandler(Filters.forwarded_from(chat_id=1234),
callback_method)

Note: When a user has disallowed adding a link to their account while forwarding their messages, this filter will not work since both telegram.Message.forwarded_from and telegram.Message.forwarded_from_chat are None. However, this behaviour is undocumented and might be changed by Telegram.

Warning: chat_ids will give a copy of the saved chat ids as frozenset. This is to ensure thread safety. To add/remove a chat, you should use add_usernames(), add_chat_ids(), remove_usernames() and remove_chat_ids(). Only update the entire set by filter. chat_ids/usernames = new_set, if you are entirely sure that it is not causing race conditions, as this will complete replace the current set of allowed chats.

Parameters

• chat_id (telegram.utils.types.SLT[int], optional) – Which chat/user ID(s) to allow through.
• username (telegram.utils.types.SLT[str], optional) – Which username(s) to allow through. Leading '@'s in usernames will be discarded.
• allow_empty (bool, optional) – Whether updates should be processed, if no chat is specified in chat_ids and usernames. Defaults to False.

Raises RuntimeError – If both chat_id and username are present.

chat_ids
Which chat/user ID(s) to allow through.
Type set(int), optional

usernames
Which username(s) (without leading '@') to allow through.
Type set(str), optional

allow_empty
Whether updates should be processed, if no chat is specified in chat_ids and usernames.
Type bool, optional

add_chat_ids(chat_id)
Add one or more chats to the allowed chat ids.
Parameters chat_id (telegram.utils.types.SLT[int], optional) – Which chat/user ID(s) to allow through.

add_usernames(username)
Add one or more chats to the allowed usernames.
Parameters username (telegram.utils.types.SLT[str], optional) – Which username(s) to allow through. Leading '@'s in usernames will be discarded.

get_chat_or_user(message)

remove_chat_ids(chat_id)
Remove one or more chats from allowed chat ids.
Parameters chat_id (telegram.utils.types.SLT[int], optional) – Which chat/user ID(s) to disallow through.

remove_usernames(username)
Remove one or more usernames from allowed usernames.
Parameters `username` *(telegram.utils.types.SLT[str], optional)* — Which username(s) to disallow through. Leading '@'s in usernames will be discarded.

```python
game = Filters.game
    Messages that contain `telegram.Game`.
```

```python
group = Filters.group
    Messages sent in a group or a supergroup chat.
```

**Note:** DEPRECATED. Use `telegram.ext.Filters.chat_type.groups` instead.

```python
has_protected_content = Filters.has_protected_content
    Messages that contain `telegram.Message.has_protected_content`.
```

New in version 13.9.

```python
invoice = Filters.invoice
    Messages that contain `telegram.Invoice`.
```

```python
is_automatic_forward = Filters.is_automatic_forward
    Messages that contain `telegram.Message.is_automatic_forward`.
```

New in version 13.9.

```python
class language(*args, **kwargs)
    Bases: `telegram.ext.filters.MessageFilter`
```

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

**Note:** According to official Telegram API documentation, not every single user has the `language_code` attribute. Do not count on this filter working on all users.

**Examples**

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

Parameters `lang` *(telegram.utils.types.SLT[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'.

```python
location = Filters.location
    Messages that contain `telegram.Location`.
```

```python
passport_data = Filterspassport_data
    Messages that contain a `telegram.PassportData`.
```

```python
photo = Filters.photo
    Messages that contain `telegram.PhotoSize`.
```

```python
poll = Filters.poll
    Messages that contain a `telegram.Poll`.
```

```python
private = Filters.private
    Messages sent in a private chat.
```

**Note:** DEPRECATED. Use `telegram.ext.Filters.chat_type.private` instead.
class regex(*args, **kwargs)
Bases: telegram.ext.filters.MessageFilter
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.

Note: Filters use the same short circuiting logic as python’s and, or and not. This means that for example:

```python
>>> Filters.regex(r'(a?x)') | Filters.regex(r'(b?x)')
```

With a message text of x, will only ever return the matches for the first filter, since the second one is never evaluated.

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

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

class sender_chat(*args, **kwargs)
Bases: telegram.ext.filters.Filters._ChatUserBaseFilter
Filters messages to allow only those which are from a specified sender chat’s chat ID or username.

Examples
- To filter for messages sent to a group by a channel with ID -1234, use MessageHandler(Filters.sender_chat(-1234), callback_method).
- To filter for messages of anonymous admins in a super group with username @anonymous, use MessageHandler(Filters.sender_chat(usernames=['anonymous']), callback_method).
- To filter for messages sent to a group by any channel, use MessageHandler(Filters.sender_chat.channel, callback_method).
- To filter for messages of anonymous admins in any super group, use MessageHandler(Filters.sender_chat.super_group, callback_method).

Note: Remember, sender_chat is also set for messages in a channel as the channel itself, so when your bot is an admin in a channel and the linked discussion group, you would receive the message twice (once from inside the channel, once inside the discussion group). Since v13.9, the field telegram.Message.is_automatic_forward will be True for the discussion group message.

See also: Filters.is_automatic_forward
Warning: `chat_ids` will return a copy of the saved chat ids as frozenset. This is to ensure thread safety. To add/remove a chat, you should use `add_usernames()`, `add_chat_ids()`, `remove_usernames()` and `remove_chat_ids()`. Only update the entire set by `filter`. `chat_ids/usernames = new_set`, if you are entirely sure that it is not causing race conditions, as this will complete replace the current set of allowed chats.

Parameters

- **chat_id** *(telegram.utils.types.SLT[int], optional)* – Which sender chat ID(s) to allow through.
- **username** *(telegram.utils.types.SLT[str], optional)* – Which sender chat username(s) to allow through. Leading '@' s in usernames will be discarded.
- **allow_empty** *(bool, optional)* – Whether updates should be processed, if no sender chat is specified in `chat_ids` and `usernames`. Defaults to False

Raises `RuntimeError` – If both `chat_id` and `username` are present.

**chat_ids**
Which sender chat chat ID(s) to allow through.

**Type** set(int), optional

**usernames**
Which sender chat username(s) (without leading '@') to allow through.

**Type** set(str), optional

**allow_empty**
Whether updates should be processed, if no sender chat is specified in `chat_ids` and `usernames`.

**Type** bool, optional

**super_group**
Messages whose sender chat is a super group.

**channel**
Messages whose sender chat is a channel.

**Examples**

```
Filters.sender_chat.supergroup
```

```
Filters.sender_chat.channel
```

**add_chat_ids**(chat_id)
Add one or more sender chats to the allowed chat ids.

**Parameters**

`chat_id` *(telegram.utils.types.SLT[int], optional)* – Which sender chat ID(s) to allow through.

**add_usernames**(username)
Add one or more sender chats to the allowed usernames.

**Parameters**

`username` *(telegram.utils.types.SLT[str], optional)* – Which sender chat username(s) to allow through. Leading '@' s in usernames will be discarded.
**remove_chat_ids** *(chat_id)*
Remove one or more sender chats from allowed chat ids.

**Parameters**
- `chat_id` *(telegram.utils.types.SLT[int], optional)* – Which sender chat ID(s) to disallow through.

**remove_usernames** *(username)*
Remove one or more sender chats from allowed usernames.

**Parameters**
- `username` *(telegram.utils.types.SLT[str], optional)* – Which sender chat username(s) to disallow through. Leading '@'s in usernames will be discarded.

**super_group** = `_SuperGroup`

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

---

**Examples**

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

---

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

**connected_website**
Messages that contain `telegram.Message.connected_website`.

**delete_chat_photo**
Messages that contain `telegram.Message.delete_chat_photo`.

**left_chat_member**
Messages that contain `telegram.Message.left_chat_member`.

**migrate**
Messages that contain `telegram.Message.migrate_to_chat_id` or `telegram.Message.migrate_from_chat_id`.

**new_chat_members**
Messages that contain `telegram.Message.new_chat_members`.

**new_chat_photo**
Messages that contain `telegram.Message.new_chat_photo`.

**new_chat_title**
Messages that contain `telegram.Message.new_chat_title`.

**message_auto_delete_timer_changed**
Messages that contain `message_auto_delete_timer_changed`.

New in version 13.4.

**pinned_message**
Messages that contain `telegram.Message.pinned_message`.

**proximity_alert_triggered**
Messages that contain `telegram.Message.proximity_alert_triggered`.

**voice_chat_scheduled**
Messages that contain `telegram.Message.voice_chat_scheduled`.

New in version 13.5.

**voice_chat_started**
Messages that contain `telegram.Message.voice_chat_started`.
New in version 13.4.

**voice_chat_ended**
Messages that contain `telegram.Message.voice_chat_ended`.

New in version 13.4.

**voice_chat_participants_invited**
Messages that contain `telegram.Message.voice_chat_participants_invited`.

New in version 13.4.

**sticker** = `Filters.sticker`
Messages that contain `telegram.Sticker`.

**successful_payment** = `Filters.successful_payment`
Messages that confirm a `telegram.SuccessfulPayment`.

**text** = `Filters.text`
Text Messages. If a list of strings is passed, it filters messages to only allow those whose text is appearing in the given list.

### Examples

To allow any text message, simply use `MessageHandler(Filters.text, callback_method)`.

A simple use case for passing a list is to allow only messages that were sent by a custom `telegram.ReplyKeyboardMarkup`:

```python
buttons = ['Start', 'Settings', 'Back']
markup = ReplyKeyboardMarkup.from_column(buttons)
MessageHandler(Filters.text(buttons), callback_method)
```

**Note:**

- Dice messages don’t have text. If you want to filter either text or dice messages, use `Filters.text | Filters.dice`.
- Messages containing a command are accepted by this filter. Use `Filters.text & (~Filters.command)`, if you want to filter only text messages without commands.

### Parameters

**update** (List[str] | Tuple[str], optional) – Which messages to allow. Only exact matches are allowed. If not specified, will allow any text message.

**message**
Updates with `telegram.Update.message`

**edited_message**
Updates with `telegram.Update.edited_message`

---

3.1. `telegram.ext` package
messages
Updates with either `telegram.Update.message` or `telegram.Update.edited_message`

channel_post
Updates with `telegram.Update.channel_post`

edited_channel_post
Updates with `telegram.Update.edited_channel_post`

channel_posts
Updates with either `telegram.Update.channel_post` or `telegram.Update.edited_channel_post`

class user(*args, **kwargs)
Bases: `telegram.ext.filters.Filters._ChatUserBaseFilter`
Filters messages to allow only those which are from specified user ID(s) or username(s).

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

Warning: `user_ids` will give a copy of the saved user ids as frozenset. This is to ensure thread safety. To add/remove a user, you should use `add_usernames()`, `add_user_ids()`, `remove_usernames()` and `remove_user_ids()`. Only update the entire set by `filter`. `user_ids/usernames = new_set`, if you are entirely sure that it is not causing race conditions, as this will complete replace the current set of allowed users.

Parameters

- **user_id** (`telegram.utils.types.SLT[int]`, optional) – Which user ID(s) to allow through.
- **username** (`telegram.utils.types.SLT[str]`, optional) – Which username(s) to allow through. Leading '@' s in usernames will be discarded.
- **allow_empty** (`bool`, optional) – Whether updates should be processed, if no user is specified in `user_ids` and `usernames`. Defaults to False

Raises **RuntimeError** – If `user_id` and `username` are both present.

user_ids
Which user ID(s) to allow through.
Type `set(int)`, optional

usernames
Which username(s) (without leading '@') to allow through.
Type `set(str)`, optional

allow_empty
Whether updates should be processed, if no user is specified in `user_ids` and `usernames`.
Type `bool`, optional

add_user_ids(user_id)
Add one or more users to the allowed user ids.
Parameters **user_id** (`telegram.utils.types.SLT[int]`, optional) – Which user ID(s) to allow through.

add_usernames(username)
Add one or more users to the allowed usernames.
Parameters **username** *(telegram.utils.types.SLT[str], optional)* – Which username(s) to allow through. Leading '@' s in usernames will be discarded.

```python
get_chat_or_user(message)
```

**remove_user_ids**(user_id)

Remove one or more users from allowed user ids.

Parameters **user_id** *(telegram.utils.types.SLT[int], optional)* – Which user ID(s) to disallow through.

```python
remove_usernames(username)
```

Remove one or more users from allowed usernames.

Parameters **username** *(telegram.utils.types.SLT[str], optional)* – Which username(s) to disallow through. Leading '@' s in usernames will be discarded.

```python
property user_ids
```

```python
venue = Filters.venue
```

Messages that contain `telegram.Venue`

```python
class via_bot(*args, **kwargs)
```

Bases: `telegram.ext.filters.Filters._ChatUserBaseFilter`

Filters messages to allow only those which are from specified via_bot ID(s) or username(s).

Examples

```python
MessageHandler(Filters.via_bot(1234), callback_method)
```

**Warning:** *bot_ids* will give a copy of the saved bot ids as frozenset. This is to ensure thread safety. To add/remove a bot, you should use `add_usernames()`, `add_bot_ids()`, `remove_usernames()` and `remove_bot_ids()`. Only update the entire set by filter. `bot_ids/usernames = new_set`, if you are entirely sure that it is not causing race conditions, as this will complete replace the current set of allowed bots.

Parameters

- **bot_id** *(telegram.utils.types.SLT[int], optional)* – Which bot ID(s) to allow through.

- **username** *(telegram.utils.types.SLT[str], optional)* – Which username(s) to allow through. Leading '@' s in usernames will be discarded.

- **allow_empty** *(bool, optional)* – Whether updates should be processed, if no user is specified in `bot_ids` and `usernames`. Defaults to False

Raises **RuntimeError** – If bot_id and username are both present.

```python
bot_ids
```

Which bot ID(s) to allow through.

Type set(int), optional

```python
usernames
```

Which username(s) (without leading '@') to allow through.

Type set(str), optional

```python
allow_empty
```

Whether updates should be processed, if no bot is specified in `bot_ids` and `usernames`.

Type bool, optional

```python
add_bot_ids(bot_id)
```

Add one or more users to the allowed user ids.
Parameters `bot_id` (telegram.utils.types.SLT[int], optional) – Which bot ID(s) to allow through.

`add_usernames` (username)
Add one or more users to the allowed usernames.

Parameters `username` (telegram.utils.types.SLT[str], optional) – Which username(s) to allow through. Leading '@'s in usernames will be discarded.

property `bot_ids`
`get_chat_or_user` (message)

`remove_bot_ids` (bot_id)
Remove one or more users from allowed user ids.

Parameters `bot_id` (telegram.utils.types.SLT[int], optional) – Which bot ID(s) to disallow through.

`remove_usernames` (username)
Remove one or more users from allowed usernames.

Parameters `username` (telegram.utils.types.SLT[str], optional) – Which username(s) to disallow through. Leading '@'s in usernames will be discarded.

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

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

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

class `telegram.ext.filters.InvertedFilter`(*args, **kwargs)
Bases: `telegram.ext.filters.UpdateFilter`
Represents a filter that has been inverted.

Parameters `f` – The filter to invert.

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

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

Returns dict or bool.

class `telegram.ext.filters.MergedFilter`(*args, **kwargs)
Bases: `telegram.ext.filters.UpdateFilter`
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.

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

Returns dict or bool.
class telegram.ext.filters.MessageFilter(*args, **kwargs)
Bases: telegram.ext.filters.BaseFilter

Base class for all Message Filters. In contrast to UpdateFilter, the object passed to filter() is update.effective_message.

Please see telegram.ext.filters.BaseFilter for details on how to create custom filters.

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

Type str

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

Type bool

abstract filter(message)
This method must be overwritten.

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

Returns dict or bool

class telegram.ext.filters.UpdateFilter(*args, **kwargs)
Bases: telegram.ext.filters.BaseFilter

Base class for all Update Filters. In contrast to MessageFilter, the object passed to filter() is update, which allows to create filters like Filters.update.edited_message.

Please see telegram.ext.filters.BaseFilter for details on how to create custom filters.

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

Type str

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

Type bool

abstract filter(update)
This method must be overwritten.

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

Returns dict or bool.

class telegram.ext.filters.XORFilter(*args, **kwargs)
Bases: telegram.ext.filters.UpdateFilter

Convenience filter acting as wrapper for MergedFilter representing the an XOR gate for two filters.

Parameters

• base_filter – Filter 1 of the merged filter.

• xor_filter – Filter 2 of the merged filter.

filter(update)
This method must be overwritten.

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

Returns dict or bool.
**telegram.ext.PollAnswerHandler**

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


Handler class to handle Telegram updates that contain a poll answer.

**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](https://git.io/fxJuV) for more info.

**Warning:** When setting `run_async` to `True`, you cannot rely on adding custom attributes to `telegram.ext.CallbackContext`. See its docs 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.

- `run_async (bool)` – Determines whether the callback will run asynchronously. Defaults to `False`.

**callback**

The callback function for this handler.

**Type** `callable`

**pass_update_queue**

Determine whether `update_queue` will be passed to the callback function.
**Type** bool

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

**Type** bool

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

**Type** bool

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

**Type** bool

**run_async**
Determines whether the callback will run asynchronously.

**Type** bool

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

**Parameters**
- **update** *(telegram.Update|object)* – Incoming update.

**Returns** bool

**telegram.ext.PollHandler**

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

**Bases:** telegram.ext.handler.Handler[telegram.update.Update, telegram.ext.utils.types.CCT]

Handler class to handle Telegram updates that contain a poll.

**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.

**Warning:** When setting run_async to True, you cannot rely on adding custom attributes to telegram.ext.CallbackContext. See its docs 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.

  ```python
  • 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.

- **run_async** (bool) – Determines whether the callback will run asynchronously. Defaults to False.

**callback**

The callback function for this handler.

Type: callable

**pass_update_queue**

Determines whether update_queue will be passed to the callback function.

Type: bool

**pass_job_queue**

Determines whether job_queue will be passed to the callback function.

Type: bool

**pass_user_data**

Determines whether user_data will be passed to the callback function.

Type: bool

**pass_chat_data**

Determines whether chat_data will be passed to the callback function.

Type: bool

**run_async**

Determines whether the callback will run asynchronously.

Type: bool

**check_update** *(update)*

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


Returns: bool
telegram.ext.PreCheckoutQueryHandler

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


Handler class to handle Telegram PreCheckout callback queries.

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.

Warning: When setting run_async to True, you cannot rely on adding custom attributes to telegram.ext.CallbackContext. See its docs 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.

- **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.

- **run_async** (bool) – Determines whether the callback will run asynchronously. Defaults to False.

  `callback`

  The callback function for this handler.

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

*Type* bool

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

*Type* bool

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

*Type* bool

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

*Type* bool

**run_async**
Determines whether the callback will run asynchronously.

*Type* bool

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

*Parameters*

**update** *(telegram.Update | object)* – Incoming 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, run_async=False)

**Bases:** telegram.ext.handler.Handler[telegram.update.Update, telegram.ext.utils.types.CCT]

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.

Note:

• PrefixHandler does not handle (edited) channel posts.

• 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.

Warning: When setting run_async to True, you cannot rely on adding custom attributes to telegram.ext.CallbackContext. See its docs for more info.

Parameters

• prefix (telegram.utils.types.SLT[str]) – The prefix(es) that will precede command.

• command (telegram.utils.types.SLT[str]) – The command or list of commands this handler should listen for.

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

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

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

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

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

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

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

• **run_async** (bool) – Determines whether the callback will run asynchronously. Defaults to False.

callback
The callback function for this handler.

*Type* callable

**filters**
Optional. Only allow updates with these Filters.

*Type* telegram.ext.BaseFilter

**pass_args**
Determines whether the handler should be passed args.

*Type* bool

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

*Type* bool

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

*Type* bool

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

*Type* bool

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

*Type* bool

**run_async**
Determines whether the callback will run asynchronously.

*Type* bool

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


*Returns* The list of args for the handler.

*Return type* list

**property command**
The list of commands this handler should listen for.

*Returns* List[str]

**property prefix**
The prefixes that will precede command.

*Returns* List[str]
**telegram.ext.RegexHandler**

```python
class telegram.ext.RegexHandler(pattern, callback, pass_groups=False, pass_groupdict=False, pass_update_queue=False, pass_job_queue=False, pass_user_data=False, pass_chat_data=False, message_updates=True, channel_post_updates=False, allow_edited=False, message_updates=True, channel_post_updates=False, allow_edited=False, run_async=False)
```

**Bases:** telegram.ext.handler.Handler[telegram.update.Update, telegram.ext.utils.types.CCT]

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.

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

**Warning:** When setting `run_async` to True, you cannot rely on adding custom attributes to `telegram.ext.CallbackContext`. See its docs for more info.

**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`.

---

3.1. `telegram.ext` package
• **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**.

• **run_async** (bool) – Determines whether the callback will run asynchronously. Defaults to **False**.

Raises **ValueError** –

**pattern**
The regex pattern.

*Type* str | Pattern

**callback**
The callback function for this handler.

*Type* callable

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

*Type* bool

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

*Type* bool

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

*Type* bool

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

*Type* bool

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

*Type* bool

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

*Type* bool

**run_async**
Determines whether the callback will run asynchronously.

*Type* bool

**collect_optional_args** (*dispatcher, update=None, check_result=None*)
Pass the results of `re.match(pattern, text).{groups(), groupdict()}` to the callback as a keyword arguments called **groups** and **groupdict**, respectively, if needed.
telegram.ext.ShippingQueryHandler

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


Handler class to handle Telegram shipping callback queries.

**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](https://git.io/fxJuV) for more info.

**Warning:** When setting `run_async` to True, you cannot rely on adding custom attributes to `telegram.ext.CallbackContext`. See its docs 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.

- **run_async** (bool) – Determines whether the callback will run asynchronously. Defaults to False.
pass_update_queue
Determines whether update_queue will be passed to the callback function.
Type bool

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

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

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

run_async
Determines whether the callback will run asynchronously.
Type bool

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, run_async=False)
    Bases: telegram.ext.handler.Handler[str, telegram.ext.utils.types.CCT]

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

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.

Warning: When setting run_async to True, you cannot rely on adding custom attributes to telegram.ext.CallbackContext. See its docs for more info.

Parameters

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

def callback(update: Update, context: CallbackContext)

The return value of the callback is usually ignored except for the special case of telegram.ext.ConversationHandler.
- **pass_args** (bool, optional) – Determines whether the handler should be passed the arguments passed to the command as a keyword argument called \texttt{args}. It will contain a list of strings, which is the text following the command split on single or consecutive whitespace characters. Default is \texttt{False}. DEPRECATED: Please switch to context based callbacks.

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

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

- **run_async** (bool) – Determines whether the callback will run asynchronously. Defaults to \texttt{False}.

---

**command**

The command this handler should listen for.

Type \texttt{str}

**callback**

The callback function for this handler.

Type \texttt{callable}

**pass_args**

Determines whether the handler should be passed \texttt{args}.

Type \texttt{bool}

**pass_update_queue**

Determines whether \texttt{update\_queue} will be passed to the callback function.

Type \texttt{bool}

**pass_job_queue**

Determines whether \texttt{job\_queue} will be passed to the callback function.

Type \texttt{bool}

**run_async**

Determines whether the callback will run asynchronously.

Type \texttt{bool}

**check_update** (update)

Determines whether an update should be passed to this handlers \texttt{callback}.

Parameters

update (object) – The incoming update.

Returns \texttt{bool}

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

Add text after the command to \texttt{CallbackContext.args} as list, split on single whitespaces.

**collect_optional_args** (dispatcher, update=\texttt{None}, check_result=\texttt{None})

Provide text after the command to the callback the \texttt{args} argument as list, split on single whitespaces.
**telegram.ext.StringRegexHandler**

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

Bases: `telegram.ext.handler.Handler`[str, telegram.ext.utils.types.CCT]

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.

**Warning:** When setting `run_async` to `True`, you cannot rely on adding custom attributes to `telegram.ext.CallbackContext`. See its docs for more info.

**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.

- `run_async` (**bool**) – Determines whether the callback will run asynchronously. Defaults to `False`.

**pattern**

The regex pattern.

*Type str | Pattern*
callback
   The callback function for this handler.
   
   Type callable

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

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

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

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

run_async
   Determines whether the callback will run asynchronously.
   
   Type bool

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

   Parameters
   update (object) – The incoming update.

   Returns
   bool

collect_additional_context(context, update, dispatcher, check_result)
   Add the result of re.match(pattern, update) to CallbackContext.matches as list with one element.

collect_optional_args(dispatcher, update=None, check_result=None)
   Pass the results of re.match(pattern, update).{groups(), groupdict()} to the callback as a keyword arguments called groups and groupdict, respectively, if needed.

telegram.ext.TypeHandler
class telegram.ext.TypeHandler(type, callback, strict=False, pass_update_queue=False, pass_job_queue=False, run_async=False)
Bases: telegram.ext.handler.Handler[telegram.ext.typehandler.UT, telegram.ext.utils.types.CCT]

Handler class to handle updates of custom types.

Warning: When setting run_async to True, you cannot rely on adding custom attributes to telegram.ext.CallbackContext. See its docs for more info.

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.
    • **run_async** (bool) – Determines whether the callback will run asynchronously. De- 
     faults to `False`.

    **type**  
    The type of updates this handler should process.
    
    Type **type**

    **callback**  
    The callback function for this handler.
    
    Type **callable**

    **strict**  
    Use `type` instead of `isinstance`. Default is `False`.
    
    Type **bool**

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

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

    **run_async**  
    Determines whether the callback will run asynchronously.
    
    Type **bool**

    **check_update** *(update)*  
    Determines whether an update should be passed to this handlers `callback`.
    
    Parameters **update** *(object)* – Incoming update.
    
    Returns **bool**
3.1.13 Persistence

**telegram.ext.BasePersistence**

class telegram.ext.BasePersistence(*args, **kwargs)

Bases: Generic[telegram.ext.utils.types.UD, telegram.ext.utils.types.CD, telegram.ext.utils.types.BD], abc.ABC

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 includes:

- get_bot_data()
- update_bot_data()
- refresh_bot_data()
- get_chat_data()
- update_chat_data()
- refresh_chat_data()
- get_user_data()
- update_user_data()
- refresh_user_data()
- get_callback_data()
- update_callback_data()
- get_conversations()
- update_conversation()
- flush()

If you don’t actually need one of those methods, a simple pass is enough. For example, if store_bot_data=False, you don’t need get_bot_data(), update_bot_data() or refresh_bot_data().

**Warning:** Persistence will try to replace telegram.Bot instances by REPLACED_BOT and insert the bot set with set_bot() upon loading of the data. This is to ensure that changes to the bot apply to the saved objects, too. If you change the bots token, this may lead to e.g. Chat not found errors. For the limitations on replacing bots see replace_bot() and insert_bot().

**Note:** replace_bot() and insert_bot() are used independently of the implementation of the update/get_*() methods, i.e. you don’t need to worry about it while implementing a custom persistence subclass.

**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.
- store_bot_data (bool, optional) – Whether bot_data should be saved by this persistence class. Default is True.
• **store_callback_data** *(bool, optional)* – Whether callback_data should be saved by this persistence class. Default is False.

New in version 13.6.

**store_user_data**

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

Type bool

**store_chat_data**

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

Type bool

**store_bot_data**

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

Type bool

**store_callback_data**

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

New in version 13.6.

Type bool

**REPLACED_BOT**: ClassVar[str] = 'bot_instance_replaced_by_ptb_persistence'

Placeholder for `telegram.Bot` instances replaced in saved data.

Type str

**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.

**abstract get_bot_data**()

Will be called by `telegram.ext.Dispatcher` upon creation with a persistence object. It should return the bot_data if stored, or an empty `telegram.ext.utils.types.BD`

Returns The restored bot data.

Return type `telegram.ext.utils.types.BD`

**get_callback_data**()

Will be called by `telegram.ext.Dispatcher` upon creation with a persistence object. If callback data was stored, it should be returned.

New in version 13.6.

Returns The restored meta data or None, if no data was stored.

Return type Optional[`telegram.ext.utils.types.CDCData`]

**abstract 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(telegram.ext.utils.types.CD)` with integer keys.

Returns The restored chat data.

Return type `DefaultDict[int, telegram.ext.utils.types.CD]`

**abstract 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

abstract 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(telegram.ext.utils.types.UD) with integer keys.

Returns  The restored user data.

Return type  DefaultDict[int,telegram.ext.utils.types.UD]

insert_bot(obj)  
Replaces all instances of REPLACED_BOT that occur within the passed object with bot. Currently, this handles objects of type list, tuple, set, frozenset, dict, defaultdict and objects that have a __dict__ or __slots__ attribute, excluding classes and objects that can’t be copied with copy.copy. If the parsing of an object fails, the object will be returned unchanged and the error will be logged.

Parameters  obj (object) – The object

Returns  Copy of the object with Bot instances inserted.

Return type  obj

refresh_bot_data(bot_data)  
Will be called by the telegram.ext.Dispatcher before passing the bot_data to a callback. Can be used to update data stored in bot_data from an external source.

New in version 13.6.

Parameters  bot_data (telegram.ext.utils.types.BD) – The bot_data.

refresh_chat_data(chat_id, chat_data)  
Will be called by the telegram.ext.Dispatcher before passing the chat_data to a callback. Can be used to update data stored in chat_data from an external source.

New in version 13.6.

Parameters  
• chat_id (int) – The chat ID this chat_data is associated with.

• chat_data (telegram.ext.utils.types.CD) – The chat_data of a single chat.

refresh_user_data(user_id, user_data)  
Will be called by the telegram.ext.Dispatcher before passing the user_data to a callback. Can be used to update data stored in user_data from an external source.

New in version 13.6.

Parameters  
• user_id (int) – The user ID this user_data is associated with.

• user_data (telegram.ext.utils.types.UD) – The user_data of a single user.

classmethod replace_bot(obj)  
Replaces all instances of telegram.Bot that occur within the passed object with REPLACED_BOT. Currently, this handles objects of type list, tuple, set, frozenset, dict, defaultdict and objects that have a __dict__ or __slots__ attribute, excluding classes and objects that can’t be copied with copy.copy. If the parsing of an object fails, the object will be returned unchanged and the error will be logged.

Parameters  obj (object) – The object

Returns  Copy of the object with Bot instances replaced.
Return type  obj

```python
set_bot (bot)
```

Set the Bot to be used by this persistence instance.

**Parameters**

- **bot** (*telegram.Bot*) – The bot.

```python
abstract update_bot_data (data)
```

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

**Parameters**

- **data** (*telegram.ext.utils.types.BD*) – The `telegram.ext.Dispatcher.bot_data`.

```python
update_callback_data (data)
```

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

New in version 13.6.

**Parameters**

- **data** (*telegram.ext.utils.types.CDCData*) – The relevant data to restore `telegram.ext.CallbackDataCache`.

```python
abstract 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.

```python
abstract update_conversation (name, key, new_state)
```

Will be called when a `telegram.ext.ConversationHandler` changes states. This allows the storage of the new state in the persistence.

**Parameters**

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

```python
abstract 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** (*telegram.ext.utils.types.UD*) – The `telegram.ext.Dispatcher.user_data[user_id]`.

---

**telegram.ext.PicklePersistence**

```python
class telegram.ext.PicklePersistence (*args, **kwargs)
```

**Bases:** `telegram.ext.basepersistence.BasePersistence[telegram.ext.utils.types.UD, telegram.ext.utils.types.CD, telegram.ext.utils.types.BD]`

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

**Warning:** `PicklePersistence` will try to replace `telegram.Bot` instances by `REPLACED_BOT` and insert the bot set with `telegram.ext.BasePersistence.set_bot()` upon loading of the data. This is to ensure that changes to the bot apply to the saved objects, too. If you change the bots token, this may lead to e.g. `Chat not found` errors. For the limitations on replacing bots see `telegram.ext.BasePersistence.replace_bot()` and `telegram.ext.BasePersistence.insert_bot()`. 
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 chat_data should be saved by this persistence class. Default is True.

- **store_bot_data** (bool, optional) – Whether bot_data should be saved by this persistence class. Default is True.

- **store_callback_data** (bool, optional) – Whether callback_data should be saved by this persistence class. Default is False.

New in version 13.6.

- **single_file** (bool, optional) – When False will store 5 separate files of `filename_user_data`, `filename_bot_data`, `filename_chat_data`, `filename_callback_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 to `flush()` . Default is False.

- **context_types** (`telegram.ext.ContextTypes`, optional) – Pass an instance of `telegram.ext.ContextTypes` to customize the types used in the context interface. If not passed, the defaults documented in `telegram.ext.ContextTypes` will be used.

New in version 13.6.

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

Type *str*

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

Type *bool*

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

Type *bool*

**store_bot_data**
Optional. Whether bot_data should be saved by this persistence class.

Type *bool*

**store_callback_data**
Optional. Whether callback_data be saved by this persistence class.

New in version 13.6.

Type *bool*

**single_file**
Optional. When False will store 5 separate files of `filename_user_data`, `filename_bot_data`, `filename_chat_data`, `filename_callback_data` and `filename_conversations`. Default is True.

Type *bool*

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

context_types
  Container for the types used in the context interface.
  New in version 13.6.
  Type telegram.ext.ContextTypes

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

get_bot_data()
  Returns the bot_data from the pickle file if it exists or an empty object of type telegram.ext.
  utils.types.BD.
  Returns The restored bot data.
  Return type telegram.ext.utils.types.BD

get_callback_data()
  Returns the callback data from the pickle file if it exists or None.
  New in version 13.6.
  Returns The restored meta data or None, if no data was stored.
  Return type Optional[telegram.ext.utils.types.CDCData]

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[int,telegram.ext.utils.types.CD]

get_conversations(name)
  Returns the conversations from the pickle file if it exists or an empty dict.
  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[int,telegram.ext.utils.types.UD]

refresh_bot_data(bot_data)
  Does nothing.
  New in version 13.6.
  See also:
  telegram.ext.BasePersistence.refresh_bot_data()

refresh_chat_data(chat_id, chat_data)
  Does nothing.
  New in version 13.6.
  See also:
  telegram.ext.BasePersistence.refresh_chat_data()

refresh_user_data(user_id, user_data)
  Does nothing.
  New in version 13.6.
See also:

`telegram.ext.BasePersistence.refresh_user_data()`

**update_bot_data** *(data)*

Will update the bot_data and depending on `on_flush` save the pickle file.

**Parameters**

- `data` *(telegram.ext.utils.types.BD)* - The `telegram.ext.Dispatcher.bot_data`.

**update_callback_data** *(data)*

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

New in version 13.6.

**Parameters**

- `data` *(telegram.ext.utils.types.CDData)* - The relevant data to restore `telegram.ext.CallbackDataCache`.

**update_chat_data** *(chat_id, data)*

Will update the chat_data and depending on `on_flush` save the pickle file.

**Parameters**

- `chat_id` *(int)* - The chat the data might have been changed for.

**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 handler’s 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 and depending on `on_flush` save the pickle file.

**Parameters**

- `user_id` *(int)* - The user the data might have been changed for.
- `data` *(telegram.ext.utils.types.UD)* - The `telegram.ext.Dispatcher.user_data[user_id]`.

---

**telegram.ext.DictPersistence**

**class** `telegram.ext.DictPersistence(*args, **kwargs)`

**Bases:** `Generic[telegram.ext.utils.types.UD, telegram.ext.utils.types.CD, telegram.ext.utils.types.BD], abc.ABC`

Using Python’s `dict` and `json` for making your bot persistent.

**Note:** This class does not implement a `flush()` method, meaning that data managed by `DictPersistence` is in-memory only and will be lost when the bot shuts down. This is, because `DictPersistence` is mainly intended as starting point for custom persistence classes that need to JSON-serialize the stored data before writing them to file/database.
Warning: DictPersistence will try to replace telegram.Bot instances by REPLACED_BOT and insert the bot set with telegram.ext.BasePersistence.set_bot() upon loading of the data. This is to ensure that changes to the bot apply to the saved objects, too. If you change the bots token, this may lead to e.g. Chat not found errors. For the limitations on replacing bots see telegram.ext.BasePersistence.replace_bot() and telegram.ext.BasePersistence.insert_bot().

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.
- `store_bot_data` (bool, optional) – Whether bot_data should be saved by this persistence class. Default is True.
- `store_callback_data` (bool, optional) – Whether callback_data should be saved by this persistence class. Default is False. New in version 13.6.
- `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 "".
- `bot_data_json` (str, optional) – JSON string that will be used to reconstruct bot_data on creating this persistence. Default is "".
- `callback_data_json` (str, optional) – JSON string that will be used to reconstruct callback_data on creating this persistence. Default is "". New in version 13.6.
- `conversations_json` (str, optional) – JSON string that will be used to reconstruct conversation on creating this persistence. Default is "".

`store_user_data`

Whether user_data should be saved by this persistence class.

Type: bool

`store_chat_data`

Whether chat_data should be saved by this persistence class.

Type: bool

`store_bot_data`

Whether bot_data should be saved by this persistence class.

Type: bool

`store_callback_data`

Whether callback_data should be saved by this persistence class.

New in version 13.6.

Type: bool

`property bot_data`

The bot_data as a dict.

Type: dict
property bot_data_json
The bot_data serialized as a JSON-string.
Type str

property callback_data
The meta data on the stored callback data.
New in version 13.6.
Type telegram.ext.utils.types.CDCData

property callback_data_json
The meta data on the stored callback data as a JSON-string.
New in version 13.6.
Type str

property chat_data
The chat_data as a dict.
Type dict

property chat_data_json
The chat_data serialized as a JSON-string.
Type str

property conversations
The conversations as a dict.
Type dict

property conversations_json
The conversations serialized as a JSON-string.
Type str

get_bot_data()
Returns the bot_data created from the bot_data_json or an empty dict.
Returns The restored bot data.
Return type dict

get_callback_data()
Returns the callback_data created from the callback_data_json or None.
New in version 13.6.
Returns The restored meta data or None, if no data was stored.
Return type Optional[telegram.ext.utils.types.CDCData]

get_chat_data()
Returns the chat_data created from the chat_data_json or an empty defaultdict.
Returns The restored chat data.
Return type defaultdict

get_conversations(name)
Returns the conversations created from the conversations_json or an empty dict.
Returns The restored conversations data.
Return type dict

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

refresh_bot_data (bot_data)
   Does nothing.
   New in version 13.6.
   See also:
   telegram.ext.BasePersistence.refresh_bot_data()

refresh_chat_data (chat_id, chat_data)
   Does nothing.
   New in version 13.6.
   See also:
   telegram.ext.BasePersistence.refresh_chat_data()

refresh_user_data (user_id, user_data)
   Does nothing.
   New in version 13.6.
   See also:
   telegram.ext.BasePersistence.refresh_user_data()

update_bot_data (data)
   Will update the bot_data (if changed).
   Parameters data (dict) – The telegram.ext.Dispatcher.bot_data.

update_callback_data (data)
   Will update the callback_data (if changed).
   New in version 13.6.
   Parameters data (telegram.ext.utils.types.CDCData) – The relevant data to restore telegram.ext.CallbackDataCache.

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 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 handler’s 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 telegram.ext.Dispatcher.user_data [user_id].

property user_data
   The user_data as a dict.
Type `dict`

property `user_data_json`
The user_data serialized as a JSON-string.

Type `str`

### 3.1.14 Arbitrary Callback Data

**telegram.ext.CallbackDataCache**

class `telegram.ext.CallbackDataCache`

Bases: `object`

A custom cache for storing the callback data of a `telegram.ext.ExtBot`. Internally, it keeps two mappings with fixed maximum size:

- One for mapping the data received in callback queries to the cached objects
- One for mapping the IDs of received callback queries to the cached objects

The second mapping allows to manually drop data that has been cached for keyboards of messages sent via inline mode. If necessary, will drop the least recently used items.

New in version 13.6.

**Parameters**

- **bot** (`telegram.ext.ExtBot`) – The bot this cache is for.
- **maxsize** (`int`, optional) – Maximum number of items in each of the internal mappings. Defaults to 1024.
- **persistent_data** (`telegram.ext.utils.types.CDCData`, optional) – Data to initialize the cache with, as returned by `telegram.ext.BasePersistence.get_callback_data()`.

**bot**
The bot this cache is for.

Type `telegram.ext.ExtBot`

**maxsize**
maximum size of the cache.

Type `int`

**clear_callback_data** (`time_cutoff=None`)
Clears the stored callback data.

Parameters **time_cutoff** (`float` | `datetime.datetime`, optional) – Pass a UNIX timestamp or a `datetime.datetime` to clear only entries which are older. For timezone naive `datetime.datetime` objects, the default timezone of the bot will be used.

**clear_callback_queries**()
Clears the stored callback query IDs.

**drop_data** (`callback_query`)
Deletes the data for the specified callback query.

**Note:** Will not raise exceptions in case the callback data is not found in the cache. Will raise `KeyError` in case the callback query can not be found in the cache.

Parameters **callback_query** (`telegram.CallbackQuery`) – The callback query.
Raises `KeyError` – If the callback query can not be found in the cache

```python
static extract_uids(callback_data)
```
Extracts the keyboard uuid and the button uuid from the given `callback_data`.

**Parameters**
- `callback_data` (`str`) – The `callback_data` as present in the button.

**Returns**
- Tuple of keyboard and button uuid

**Return type** (`str, str`)

```python
property persistence_data
```
The data that needs to be persisted to allow caching callback data across bot reboots.

**Type** `telegram.ext.utils.types.CDCData`

```python
process_callback_query(callback_query)
```
Replaces the data in the callback query and the attached messages keyboard with the cached objects, if necessary. If the data could not be found, `telegram.ext.InvalidCallbackData` will be inserted. If `callback_query.data` or `callback_query.message` is present, this also saves the callback queries ID in order to be able to resolve it to the stored data.

**Note:** Also considers inserts data into the buttons of `telegram.Message.reply_to_message` and `telegram.Message.pinned_message` if necessary.

**Warning:** *In place*, i.e. the passed `telegram.CallbackQuery` will be changed!

**Parameters**

```python
process_keyboard(reply_markup)
```
Registers the reply markup to the cache. If any of the buttons have `callback_data`, stores that data and builds a new keyboard with the correspondingly replaced buttons. Otherwise does nothing and returns the original reply markup.

**Parameters**
- `reply_markup` (`telegram.InlineKeyboardMarkup`) – The keyboard.

**Returns**
- The keyboard to be passed to Telegram.

**Return type** `telegram.InlineKeyboardMarkup`

```python
process_message(message)
```
Replaces the data in the inline keyboard attached to the message with the cached objects, if necessary. If the data could not be found, `telegram.ext.InvalidCallbackData` will be inserted.

**Note:** Checks `telegram.Message.via_bot` and `telegram.Message.from_user` to check if the reply markup (if any) was actually sent by this caches bot. If it was not, the message will be returned unchanged.

Note that this will fail for channel posts, as `telegram.Message.from_user` is `None` for those! In the corresponding reply markups the callback data will be replaced by `telegram.ext.InvalidCallbackData`.

**Warning:**
- *In place*, i.e. the passed `telegram.Message` will be changed!

**telegram.ext.InvalidCallbackData**

```python
class telegram.ext.InvalidCallbackData(callback_data=None):
    Bases: telegram.error.TelegramError
    
    Raised when the received callback data has been tempered with or deleted from cache.
    
    New in version 13.6.
    
    Parameters `callback_data (int, optional)` – The button data of which the callback data could not be found.
    
    `callback_data`  
    Optional. The button data of which the callback data could not be found.
    
    Type `int`
```

### 3.1.15 `utils`

**telegram.ext.utils.promise.Promise**

```python
class telegram.ext.utils.promise.Promise(pooled_function, args, kwargs, update=None, error_handling=True):
    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`.
    
    • `update (telegram.Update|object, optional)` – The update this promise is associated with.
    
    • `error_handling (bool, optional)` – Whether exceptions raised by `func` may be handled by error handlers. Defaults to `True`.
    
    `pooled_function`
    The callable that will be called concurrently.
    
    Type `callable`
    
    `args`
    Positional arguments for `pooled_function`.
    
    Type `list|tuple`
    
    `kwargs`
    Keyword arguments for `pooled_function`.
    
    Type `dict`
    
    `done`
    Is set when the result is available.
    
    Type `threading.Event`
    
    `update`
    Optional. The update this promise is associated with.
    
    Type `telegram.Update|object`
```
error_handling
Optional. Whether exceptions raised by func may be handled by error handlers. Defaults to True.
Type bool

add_done_callback(callback)
Callback to be run when telegram.ext.utils.promise.Promise becomes done.

Note: Callback won’t be called if pooled_function raises an exception.

Parameters
• callback (callable) – The callable that will be called when promise is done.
• will be called by passing Promise.result() as only positional argument. (callback)

property 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 object exception raised by pooled_function:

run()
Calls the pooled_function callable.

telegram.ext.utils.types Module
This module contains custom typing aliases.
New in version 13.6.
telegram.ext.utils.types.BD
Type of the bot data.
New in version 13.6.
alias of TypeVar('BD')
telegram.ext.utils.types.CCT
An instance of telegram.ext.CallbackContext or a custom subclass.
New in version 13.6.
alias of TypeVar('CCT')
telegram.ext.utils.types.CD
Type of the chat data for a single user.
New in version 13.6.
alias of TypeVar('CD')
telegram.ext.utils.types.CDCData
Data returned by telegram.ext.CallbackDataCache.persistence_data.
New in version 13.6.
Type Tuple[List[Tuple[str, float, Dict[str, any]]], Dict[str, str]]
alias of Tuple[List[Tuple[str, float, Dict[str, Any]]], Dict[str, str]]

```
from telegram.ext.utils.types import ConversationDict
```

Dicts as maintained by the `ConversationHandler`.
New in version 13.6.

alias of Dict[Tuple[int, ...], Optional[object]]

```
from telegram.ext.utils.types import UD
```

Type of the user data for a single user.
New in version 13.6.

alias of TypeVar('UD')

### 3.2 telegram package

#### 3.2.1 telegram.Animation

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

Bases: `telegram.base.TelegramObject`

This object represents an animation file (GIF or H.264/MPEG-4 AVC video without sound).

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `file_unique_id` is equal.

**Parameters**

- `file_id (str)` – Identifier for this file, which can be used to download or reuse the file.
- `file_unique_id (str)` – Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the 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)` – 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.
- `bot (telegram.Bot, optional)` – The Bot to use for instance methods.
- `**kwargs (dict)` – Arbitrary keyword arguments.

**file_id**
File identifier.

Type `str`

**file_unique_id**
Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the file.

Type `str`
width
    Video width as defined by sender.
    Type int

height
    Video height as defined by sender.
    Type int

duration
    Duration of the video in seconds as defined by sender.
    Type int

thumb
    Optional. Animation thumbnail as defined by sender.
    Type telegram.PhotoSize

file_name
    Optional. Original animation filename as defined by sender.
    Type str

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

file_size
    Optional. File size.
    Type int

bot
    Optional. The Bot to use for instance methods.
    Type telegram.Bot

classmethod de_json(self, data, bot)
    See telegram.TelegramObject.de_json().

get_file(self, timeout=None, api_kwargs=None)
    Convenience wrapper over telegram.Bot.get_file
    For the documentation of the arguments, please see telegram.Bot.get_file().
    Returns telegram.File
    Raises telegram.error.TelegramError

3.2.2 telegram.Audio

class telegram.Audio(file_id, file_unique_id, duration, performer=None, title=None, mime_type=None, file_size=None, thumb=None, bot=None, file_name=None, **kwargs)
    Bases: telegram.base.TelegramObject

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their file_unique_id is equal.

Parameters
    * file_id (str) – Identifier for this file, which can be used to download or reuse the file.
• **file_unique_id** (str) – Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the 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.

• **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.

• **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.

**file_id**

Identifier for this file.

Type str

**file_unique_id**

Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the file.

Type str

**duration**

Duration of the audio in seconds.

Type int

**performer**

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

Type str

**title**

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

Type str

**file_name**

Optional. Original filename as defined by sender.

Type str

**mime_type**

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

Type str

**file_size**

Optional. File size.

Type int

**thumb**

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

Type **telegram.PhotoSize**

**bot**

Optional. The Bot to use for instance methods.

Type **telegram.Bot**
classmethod de_json(data, bot)
See `telegram.TelegramObject.de_json()`.

get_file(timeout=None, api_kwargs=None)
Convenience wrapper over `telegram.Bot.get_file`
For the documentation of the arguments, please see `telegram.Bot.get_file()`.

Returns `telegram.File`
Raises `telegram.error.TelegramError`

3.2.3 telegram.Bot

class telegram.Bot(token, base_url=None, base_file_url=None, request=None, private_key=None, private_key_password=None, defaults=None)
Bases: `telegram.base.TelegramObject`

This object represents a Telegram Bot.
New in version 13.2: Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `bot` is equal.

Note: Most bot methods have the argument `api_kwargs` which allows to pass arbitrary keywords to the Telegram API. This can be used to access new features of the API before they were incorporated into PTB. However, this is not guaranteed to work, i.e. it will fail for passing files.

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.
- `defaults` (`telegram.ext.Defaults`, optional) – An object containing default values to be used if not set explicitly in the bot methods.

Deprecated since version 13.6: Passing `telegram.ext.Defaults` to `telegram.Bot` is deprecated. If you want to use `telegram.ext.Defaults`, please use `telegram.ext.ExtBot` instead.

addStickerToSet(user_id, name, emojis, png_sticker=None, mask_position=None, timeout=20, tgs_sticker=None, api_kwargs=None, webm_sticker=None)
Alias for `add_sticker_to_set()`

add_sticker_to_set(user_id, name, emojis, png_sticker=None, mask_position=None, timeout=20, tgs_sticker=None, api_kwargs=None, webm_sticker=None)
Use this method to add a new sticker to a set created by the bot. You must use exactly one of the fields `png_sticker`, `tgs_sticker` or `webm_sticker`. Animated stickers can be added to animated sticker sets and only to them. Animated sticker sets can have up to 50 stickers. Static sticker sets can have up to 120 stickers.
Warning: As of API 4.7 `png_sticker` is an optional argument and therefore the order of the arguments had to be changed. Use keyword arguments to make sure that the arguments are passed correctly.

Note: The `png_sticker` and `tgs_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 | bytes | pathlib.Path, optional)` – 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.
  
  Changed in version 13.2: Accept `bytes` as input.
- `tgs_sticker (str | filelike object | bytes | pathlib.Path, optional)` – TGS animation with the sticker, uploaded using multipart/form-data. See [https://core.telegram.org/stickers#animated-sticker-requirements](https://core.telegram.org/stickers#animated-sticker-requirements) for technical requirements.
  
  Changed in version 13.2: Accept `bytes` as input.
- `webm_sticker (str | file object | bytes | pathlib.Path, optional)` – WEBM video with the sticker, uploaded using multipart/form-data. See [https://core.telegram.org/stickers#video-sticker-requirements](https://core.telegram.org/stickers#video-sticker-requirements) for technical requirements.
  
  New in version 13.11.
- `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).
- `api_kwargs (dict, optional)` – Arbitrary keyword arguments to be passed to the Telegram API.

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

**Return type** `bool`

**Raises** `telegram.error.TelegramError` –

- `answerCallbackQuery (callback_query_id, text=None, show_alert=False, url=None, cache_time=None, timeout=None, api_kwargs=None)`
  
  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, current_offset=None, api_kwargs=None)`
  
  Alias for `answer_inline_query()`
- `answerPreCheckoutQuery (pre_checkout_query_id, ok, error_message=None, timeout=None, api_kwargs=None)`
  
  Alias for `answer_pre_checkout_query()`

3.2. telegram package 97
answerShippingQuery(shipping_query_id, ok, shipping_options=None, error_message=None, timeout=None, api_kwargs=None)

Alias for answer_shipping_query()

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

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).
- api_kwargs (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns bool On success, True is returned.

Raises telegram.error.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, current_offset=None, api_kwargs=None)

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

Warning: In most use cases current_offset should not be passed manually. Instead of calling this method directly, use the shortcut telegram.InlineQuery.answer() with auto_pagination=True, which will take care of passing the correct value.

Parameters

- inline_query_id (str) – Unique identifier for the answered query.
- results (List[telegram.InlineQueryResult] | Callable) – A list of results for the inline query. In case current_offset is passed, results may also be a callable that accepts the current page index starting from 0. It must return either a list of telegram.InlineQueryResult instances or None if there are no more results.
- 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.

• **current_offset** *(str, optional)* – The `telegram.InlineQuery.offset` of the inline query to answer. If passed, PTB will automatically take care of the pagination for you, i.e. pass the correct `next_offset` and truncate the results list/get the results from the callable you passed.

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

• **api_kwargs** *(dict, optional)* – Arbitrary keyword arguments to be passed to the Telegram API.

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

**Return type**  bool

**Raises**  `telegram.error.TelegramError`

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

Once the user has confirmed their payment and shipping details, the Bot API sends the final confirmation in the form of an `telegram.Update` with the field `Update.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 check-out (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).

• **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

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

**Return type** bool

**Raises** `telegram.error.TelegramError` –

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

If you sent an invoice requesting a shipping address and the parameter is_flexible was specified, the Bot API will send an `telegram.Update` with a `Update.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).

• **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

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

**Return type** bool

**Raise** `telegram.error.TelegramError` –

**approveChatJoinRequest** (chat_id, user_id, timeout=None, api_kwargs=None)

**alias** for **approve_chat_join_request()**

**approve_chat_join_request** (chat_id, user_id, timeout=None, api_kwargs=None)

Use this method to approve a chat join request.

The bot must be an administrator in the chat for this to work and must have the `telegram.ChatPermissions.can_invite_users` administrator right.

New in version 13.8.

**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).

• **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

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

**Return type** bool

**Raises** `telegram.error.TelegramError`

### banChatMember

`banChatMember(chat_id, user_id, timeout=None, until_date=None, api_kwargs=None, revoke_messages=None)`

Alias for `ban_chat_member()`

### banChatSenderChat

`banChatSenderChat(chat_id, sender_chat_id, timeout=None, api_kwargs=None)`

Alias for `ban_chat_sender_chat()`

### ban_chat_member

Use this method to ban a user from a group, supergroup or a channel. In the case of supergroups and channels, 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 chat for this to work and must have the appropriate admin rights.

New in version 13.7.

**Parameters**

• **chat_id** (int | str) – Unique identifier for the target group or username of the target supergroup or 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. Applied for supergroups and channels only. For timezone naive `datetime.datetime` objects, the default timezone of the bot will be used.

• **revoke_messages** (bool, optional) – Pass True to delete all messages from the chat for the user that is being removed. If False, the user will be able to see messages in the group that were sent before the user was removed. Always True for supergroups and channels.

New in version 13.4.

• **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

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

**Return type** bool

**Raises** `telegram.error.TelegramError`

### ban_chat_sender_chat

Use this method to ban a channel chat in a supergroup or a channel. Until the chat is unbanned, the owner of the banned chat won’t be able to send messages on behalf of any of their channels. The bot
must be an administrator in the supergroup or channel for this to work and must have the appropriate
administrator rights.

New in version 13.9.

Parameters

- \texttt{chat} \_\texttt{id} (int | str) – Unique identifier for the target group or username of the
  target supergroup or channel (in the format @channelusername).

- \texttt{sender} \_\texttt{chat} \_\texttt{id} (int) – Unique identifier of the target sender chat.

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

- \texttt{api} \_\texttt{kw} \_\texttt{args} (dict, optional) – Arbitrary keyword arguments to be passed to the
  Telegram API.

Returns On success, True is returned.

Return type bool

Raises \texttt{telegram.error.TelegramError} –

property \texttt{bot}
User instance for the bot as returned by \texttt{get\_me()}. 
Type \texttt{telegram.User}

property \texttt{can} \_\texttt{join} \_\texttt{groups}
Bot’s \texttt{telegram.User.can\_join\_groups} attribute.
Type bool

property \texttt{can\_read} \_\texttt{all} \_\texttt{group} \_\texttt{messages}
Bot’s \texttt{telegram.User.can\_read\_all\_group\_messages} attribute.
Type bool

\texttt{close}(timeout=None)
Use this method to close the bot instance before moving it from one local server to another. You need
to delete the webhook before calling this method to ensure that the bot isn’t launched again after server
restart. The method will return error 429 in the first 10 minutes after the bot is launched.

Parameters \texttt{timeout} (int | float, optional) – If 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

Return type True

Raises \texttt{telegram.error.TelegramError} –

property \texttt{commands}
Bot’s commands as available in the default scope.
Deprecated since version 13.7: This property has been deprecated since there can be different com-
mands available for different scopes.
Type List[\texttt{BotCommand}]

\texttt{copyMessage}(\texttt{chat} \_\texttt{id}, \texttt{from} \_\texttt{chat} \_\texttt{id}, \texttt{message} \_\texttt{id}, \texttt{caption}=None, \texttt{parse} \_\texttt{mode}=None, \texttt{caption} \_\texttt{entities}=None, \texttt{disable} \_\texttt{notification}=None, \texttt{reply} \_\texttt{to} \_\texttt{message} \_\texttt{id}=None, \texttt{allow} \_\texttt{sending} \_\texttt{without} \_\texttt{reply}=None, \texttt{reply} \_\texttt{markup}=None, \texttt{timeout}=None, \texttt{api} \_\texttt{kw} \_\texttt{args}=None, \texttt{protect} \_\texttt{content}=None)

Alias for \texttt{copy\_message()}
copy_message(chat_id, from_chat_id, message_id, caption=None, parse_mode=None, caption_entities=None, disable_notification=None, reply_to_message_id=None, allow_sending_without_reply=None, reply_markup=None, timeout=None, api_kwargs=None, protect_content=None)

Use this method to copy messages of any kind. Service messages and invoice messages can’t be copied. The method is analogous to the method forward_message(), but the copied message doesn’t have a link to the original message.

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).

• **message_id**(int) – Message identifier in the chat specified in from_chat_id.

• **caption**(str, optional) – New caption for media, 0-1024 characters after entities parsing. If not specified, the original caption is kept.

• **parse_mode**(str, optional) – Mode for parsing entities in the new caption. See the constants in telegram.ParseMode for the available modes.

• **caption_entities**(telegram.utils.types.SLT[MessageEntity]) – List of special entities that appear in the new caption, which can be specified instead of parse_mode

• **disable_notification**(bool, optional) – Sends the message silently. Users will receive a notification with no sound.

• **protect_content**(bool, optional) – Protects the contents of the sent message from forwarding and saving.

New in version 13.10.

• **reply_to_message_id**(int, optional) – If the message is a reply, ID of the original message.

• **allow_sending_without_reply**(bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.

• **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).

• **api_kwargs**(dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success

Return type **telegram.MessageId**

Raises **telegram.error.TelegramError** –

createChatInviteLink(chat_id, expire_date=None, member_limit=None, timeout=None, api_kwargs=None, name=None, creates_join_request=None)

Alias for create_chat_invite_link()

createFromNewStickerSet(user_id, name, title, emojis, png_sticker=None, contains_masks=None, mask_position=None, tgs_sticker=None, webm_sticker=None)

Alias for create_new_sticker_set()
create_chat_invite_link \( (\text{chat_id}, \text{expire_date}=\text{None, member_limit}=\text{None, timeout}=\text{None, api_kwargs}=\text{None, name}=\text{None, creates_join_request}=\text{None}) \)

Use this method to create an additional invite link for a chat. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights. The link can be revoked using the method \text{revoke_chat_invite_link}().

New in version 13.4.

Parameters

- \text{chat_id} (\text{int} | \text{str}) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- \text{expire_date} (\text{int} | \text{datetime.datetime}, optional) – Date when the link will expire. Integer input will be interpreted as Unix timestamp. For timezone naive datetime.datetime objects, the default timezone of the bot will be used.
- \text{member_limit} (\text{int}, optional) – Maximum number of users that can be members of the chat simultaneously after joining the chat via this invite link; 1-99999.
- \text{timeout} (\text{int} | \text{float}, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- \text{api_kwargs} (\text{dict}, optional) – Arbitrary keyword arguments to be passed to the Telegram API.
- \text{name} (\text{str}, optional) – Invite link name; 0-32 characters.

New in version 13.8.

- \text{creates_join_request} (\text{bool}, optional) – True, if users joining the chat via the link need to be approved by chat administrators. If True, member_limit can’t be specified.

New in version 13.8.

Returns \text{telegram.ChatInviteLink}

Raises \text{telegram.error.TelegramError}

create_new_sticker_set \( (\text{user_id}, \text{name}, \text{title}, \text{emojis}, \text{png_sticker}=\text{None, contains_masks}=\text{None, mask_position}=\text{None, timeout}=\text{20, tgs_sticker}=\text{None, api_kwargs}=\text{None, webm_sticker}=\text{None}) \)

Use this method to create new sticker set owned by a user. The bot will be able to edit the created sticker set. You must use exactly one of the fields \text{png_sticker}, \text{tgs_sticker}, or \text{webm_sticker}.

Warning: As of API 4.7 png_sticker is an optional argument and therefore the order of the arguments had to be changed. Use keyword arguments to make sure that the arguments are passed correctly.

Note: The \text{png_sticker} and \text{tgs_sticker} argument can be either a \text{file_id}, an \text{URL} or a \text{file} from disk \text{open(filename, 'rb')}
• **title** (str) – Sticker set title, 1-64 characters.

• **png_sticker** (str | filelike object | bytes | pathlib.Path, optional) – 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.

  Changed in version 13.2: Accept bytes as input.

• **tgs_sticker** (str | filelike object | bytes | pathlib.Path, optional) – TGS animation with the sticker, uploaded using multipart/form-data. See https://core.telegram.org/stickers#animated-sticker-requirements for technical requirements.

  Changed in version 13.2: Accept bytes as input.

• **webm_sticker** (str | file object | bytes | pathlib.Path, optional) – WEBM video with the sticker, uploaded using multipart/form-data. See https://core.telegram.org/stickers#video-sticker-requirements for technical requirements.

  New in version 13.11.

• **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).

• **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns | On success, True is returned.
Return type | bool

Raises | telegram.error.TelegramError –

declineChatJoinRequest (chat_id, user_id, timeout=None, api_kwargs=None)

Alias for decline_chat_join_request()

decline_chat_join_request (chat_id, user_id, timeout=None, api_kwargs=None)

Use this method to decline a chat join request.

The bot must be an administrator in the chat for this to work and must have the telegram.
ChatPermissions.can_invite_users administrator right.

New in version 13.8.

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).

• **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns | On success, True is returned.
Return type  bool

Raises  telegram.error.TelegramError –

deleteChatPhoto (chat_id, timeout=None, api_kwargs=None)
  Alias for  delete_chat_photo()

deleteChatStickerSet (chat_id, timeout=None, api_kwargs=None)
  Alias for  delete_chat_sticker_set()

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

deleteMyCommands (scope=None, language_code=None, api_kwargs=None, timeout=None)
  Alias for  delete_my_commands()

deleteStickerFromSet (sticker, timeout=None, api_kwargs=None)
  Alias for  delete_sticker_from_set()

deleteWebhook (timeout=None, api_kwargs=None, drop_pending_updates=None)
  Alias for  delete_webhook()

delete_chat_photo (chat_id, timeout=None, api_kwargs=None)
  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).

• api_kwargs (dict, optional) – Arbitrary keyword arguments to be passed to the
  Telegram API.

Returns  On success, True is returned.

Return type  bool

Raises  telegram.error.TelegramError –

delete_chat_sticker_set (chat_id, timeout=None, api_kwargs=None)
  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).

• api_kwargs (dict, optional) – Arbitrary keyword arguments to be passed to the
  Telegram API.

Returns  On success, True is returned.

Return type  bool

delete_message (chat_id, message_id, timeout=None, api_kwargs=None)
  Use this method to delete a message, including service messages, with the following limitations:

• A message can only be deleted if it was sent less than 48 hours ago.
• A dice message in a private chat can only be deleted if it was sent more than 24 hours ago.
• Bots can delete outgoing messages in private chats, groups, and supergroups.
• Bots can delete incoming messages in private chats.
• Bots granted `telegram.ChatMember.can_post_messages` permissions can delete outgoing messages in channels.
• If the bot is an administrator of a group, it can delete any message there.
• If the bot has `telegram.ChatMember.can_delete_messages` permission in a supergroup or a channel, it can delete any message there.

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).
• `api_kargs` (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns On success, `True` is returned.

Return type `bool`

Raises `telegram.error.TelegramError` –

`delete_my_commands` (scope=None, language_code=None, api_kargs=None, timeout=None)
Use this method to delete the list of the bot’s commands for the given scope and user language. After deletion, higher level commands will be shown to affected users.

New in version 13.7.

Parameters

• `scope` (telegram.BotCommandScope, optional) – A JSON-serialized object, describing scope of users for which the commands are relevant. Defaults to `telegram.BotCommandScopeDefault`
• `language_code` (str, optional) – A two-letter ISO 639-1 language code. If empty, commands will be applied to all users from the given scope, for whose language there are no dedicated commands.
• `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
• `api_kargs` (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns On success, `True` is returned.

Return type `bool`

Raises `telegram.error.TelegramError` –

`delete_sticker_from_set` (sticker, timeout=None, api_kargs=None)
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).

• `api_kwargs` (dict, optional) – Arbitrary keyword arguments to be passed to the
Telegram API.

Returns On success, True is returned.

Return type bool

Raises `telegram.error.TelegramError`

`delete_webhook` (timeout=None, api_kwargs=None, drop_pending_updates=None)

Use this method to remove webhook integration if you decide to switch back to `get_updates()`.

Parameters

• `drop_pending_updates` (bool, optional) – Pass True to drop all pending up-
dates.

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

• `api_kwargs` (dict, optional) – Arbitrary keyword arguments to be passed to the
Telegram API.

Returns On success, True is returned.

Return type bool

Raises `telegram.error.TelegramError`

`editChatInviteLink` (chat_id, invite_link, expire_date=None, member_limit=None,
timeout=None, api_kwargs=None, name=None, creates_join_request=None)

Alias for `edit_chat_invite_link()`

`editMessageCaption` (chat_id=None, message_id=None, inline_message_id=None,
caption=None, reply_markup=None, timeout=None, parse_mode=None,
api_kwargs=None, caption_entities=None)

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, api_kwargs=None, horizontal_accuracy=None, heading=None,
proximity_alert_radius=None)

Alias for `edit_message_live_location()`

`editMessageMedia` (chat_id=None, message_id=None, inline_message_id=None,
media=None, reply_markup=None, timeout=None, api_kwargs=None)

Alias for `edit_message_media()`

`editMessageReplyMarkup` (chat_id=None, message_id=None, inline_message_id=None,
reply_markup=None, timeout=None, api_kwargs=None)

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, api_kwargs=None, entities=None)

Alias for `edit_message_text()`

`edit_chat_invite_link` (chat_id, invite_link, expire_date=None, member_limit=None,
timeout=None, api_kwargs=None, name=None, creates_join_request=None)

Use this method to edit a non-primary invite link created by the bot. The bot must be an administrator
in the chat for this to work and must have the appropriate admin rights.
Note: Though not stated explicitly in the official docs, Telegram changes not only the optional parameters that are explicitly passed, but also replaces all other optional parameters to the default values. However, since not documented, this behaviour may change unbeknown to PTB.

New in version 13.4.

Parameters

- **chat_id**(int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **invite_link**(str) – The invite link to edit.
- **expire_date**(int | datetime.datetime, optional) – Date when the link will expire. For timezone naive datetime.datetime objects, the default timezone of the bot will be used.
- **member_limit**(int, optional) – Maximum number of users that can be members of the chat simultaneously after joining the chat via this invite link; 1-99999.
- **timeout**(int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **api_kwargs**(dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.
- **name**(str, optional) – Invite link name; 0-32 characters.

New in version 13.8.

- **creates_join_request**(bool, optional) – True, if users joining the chat via the link need to be approved by chat administrators. If True, member_limit can’t be specified.

New in version 13.8.

Returns **telegram.ChatInviteLink**

Raises **telegram.error.TelegramError** –

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

Use this method to edit captions of messages.

Parameters

- **chat_id**(int | str, optional) – Required if inline_message_id is not specified. Unique identifier for the target chat or username of the target channel (in the format @channelusername)
- **message_id**(int, optional) – Required if inline_message_id is not specified. Identifier of the message to edit.
- **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, 0-1024 characters after entities parsing.
- **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.
• **caption_entities** (List[telegram.MessageEntity], optional) – List of special entities that appear in message text, which can be specified instead of `parse_mode`.

• **reply_markup** (telegram.InlineKeyboardMarkup, optional) – A JSON-serialized object for an inline keyboard.

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

• **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns**
On success, if edited message is not an inline message, the edited message is returned, otherwise `True` is returned.

**Return type** telegram.Message

**Raises** telegram.error.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, api_kwargs=None, horizontal_accuracy=None, heading=None, proximity_alert_radius=None)

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 `telegram.Location.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, optional) – Required if `inline_message_id` is not specified. Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).

• **message_id** (int, optional) – Required if `inline_message_id` is not specified. Identifier of the message to edit.

• **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.

• **horizontal_accuracy** (float, optional) – The radius of uncertainty for the location, measured in meters; 0-1500.

• **heading** (int, optional) – Direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.

• **proximity_alert_radius** (int, optional) – Maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and 100000 if specified.

• **reply_markup** (telegram.InlineKeyboardMarkup, optional) – A JSON-serialized object for a new inline keyboard.

• **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
• `api_kwargs` *(dict, optional)* – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns** On success, if edited message is not an inline message, the edited message is returned, otherwise `True` is returned.

**Return type** `telegram.Message`

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

Use this method to edit animation, audio, document, photo, or video messages. If a message is part of a message album, then it can be edited only to an audio for audio albums, only to a document for document albums and to a photo or a video otherwise. When an inline message is edited, a new file can’t be uploaded. Use a previously uploaded file via its `file_id` or specify a URL.

**Parameters**

• `chat_id` *(int | str, optional)* – Required if inline_message_id is not specified. Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).

• `message_id` *(int, optional)* – Required if inline_message_id is not specified. Identifier of the message to edit.

• `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.InlineKeyboardMarkup, optional)* – A JSON-serialized object for an inline keyboard.

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

• `api_kwargs` *(dict, optional)* – Arbitrary keyword arguments to be passed to the Telegram API.

**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.error.TelegramError`

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

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, optional)* – Required if inline_message_id is not specified. Unique identifier for the target chat or username of the target channel (in the format `@channelusername`).

• `message_id` *(int, optional)* – Required if inline_message_id is not specified. Identifier of the message to edit.

• `inline_message_id` *(str, optional)* – Required if chat_id and message_id are not specified. Identifier of the inline message.

• `reply_markup` *(telegram.InlineKeyboardMarkup, optional)* – A JSON-serialized object for an inline keyboard.
• **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).

• **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns** On success, if edited message is not an inline message, the edited message is returned, otherwise `True` is returned.

**Return type** `telegram.Message`

**Raises** `telegram.error.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, api_kwargs=None, entities=None)*

Use this method to edit text and game messages.

**Parameters**

• **chat_id** (int | str, optional) – Required if inline_message_id is not specified. Unique identifier for the target chat or username of the target channel (in the format @channelusername)

• **message_id** (int, optional) – Required if inline_message_id is not specified. Identifier of the message to edit.

• **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, 1-4096 characters after entities parsing.

• **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. See the constants in `telegram.ParseMode` for the available modes.

• **entities** (List[telegram.MessageEntity], optional) – List of special entities that appear in message text, which can be specified instead of `parse_mode`.

• **disable_web_page_preview** (bool, optional) – Disables link previews for links in this message.

• **reply_markup** (telegram.InlineKeyboardMarkup, optional) – A JSON-serialized object for an inline keyboard.

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

• **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns** On success, if edited message is not an inline message, the edited message is returned, otherwise `True` is returned.

**Return type** `telegram.Message`

**Raises** `telegram.error.TelegramError` –

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

Alias for `export_chat_invite_link()`

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

Use this method to generate a new primary invite link for a chat; any previously generated link is revoked. 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).

• `api_kwargs (dict, optional)` – Arbitrary keyword arguments to be passed to the Telegram API.

**Note:** Each administrator in a chat generates their own invite links. Bots can’t use invite links generated by other administrators. If you want your bot to work with invite links, it will need to generate its own link using `export_chat_invite_link()` or by calling the `get_chat()` method. If your bot needs to generate a new primary invite link replacing its previous one, use `export_chat_invite_link` again.

**Returns** New invite link on success.

**Return type** `str`

**Raises** `telegram.error.TelegramError`

---

**property first_name**

Bot’s first name.

**Type** `str`

---

```python
forwardMessage(chat_id, from_chat_id, message_id, disable_notification=None, timeout=None, api_kwargs=None, protect_content=None)
```

Alias for `forward_message()`

```python
forward_message(chat_id, from_chat_id, message_id, disable_notification=None, timeout=None, api_kwargs=None, protect_content=None)
```

Use this method to forward messages of any kind. Service messages can’t be forwarded.

**Note:** Since the release of Bot API 5.5 it can be impossible to forward messages from some chats. Use the attributes `telegram.Message.has_protected_content` and `telegram.Chat.has_protected_content` to check this.

As a workaround, it is still possible to use `copy_message()`. However, this behaviour is undocumented and might be changed by Telegram.

---

**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).

- `message_id (int)` – Message identifier in the chat specified in from_chat_id.

- `disable_notification (bool, optional)` – Sends the message silently. Users will receive a notification with no sound.

- `protect_content (bool, optional)` – Protects the contents of the sent message from forwarding and saving.

New in version 13.10.

- `timeout (int | float, optional)` – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
• `api_kwargs` (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.error.TelegramError`

---

`getChat` *(chat_id, timeout=None, api_kwargs=None)*

Alias for `get_chat()`

`getChatAdministrators` *(chat_id, timeout=None, api_kwargs=None)*

Alias for `get_chat_administrators()`

`getChatMember` *(chat_id, user_id, timeout=None, api_kwargs=None)*

Alias for `get_chat_member()`

`getChatMemberCount` *(chat_id, timeout=None, api_kwargs=None)*

Alias for `get_chat_member_count()`

`getChatMembersCount` *(chat_id, timeout=None, api_kwargs=None)*

Alias for `get_chat_members_count()`

`getFile` *(file_id, timeout=None, api_kwargs=None)*

Alias for `get_file()`

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

Alias for `get_game_high_scores()`

`getMe` *(timeout=None, api_kwargs=None)*

Alias for `get_me()`

`getMyCommands` *(timeout=None, api_kwargs=None, scope=None, language_code=None)*

Alias for `get_my_commands()`

`getStickerSet` *(name, timeout=None, api_kwargs=None)*

Alias for `get_sticker_set()`

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

Alias for `get_updates()`

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

Alias for `get_user_profile_photos()`

`getWebhookInfo` *(timeout=None, api_kwargs=None)*

Alias for `get_webhook_info()`

`get_chat` *(chat_id, timeout=None, api_kwargs=None)*

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 supergroup or 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).

• `api_kwargs` (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns `telegram.Chat`

Raises `telegram.error.TelegramError`
**get_chat_administrators** *(chat_id, timeout=None, api_kwargs=None)*

Use this method to get a list of administrators in a chat.

**Parameters**

- **chat_id** *(int | str)* – Unique identifier for the target chat or username of the target supergroup or 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).
- **api_kwargs** *(dict, optional)* – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns** On success, returns a list 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.

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

**Raises** *telegram.error.TelegramError*

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

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 supergroup or 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).
- **api_kwargs** *(dict, optional)* – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns** *telegram.ChatMember*

**Raises** *telegram.error.TelegramError*

**get_chat_member_count** *(chat_id, timeout=None, api_kwargs=None)*

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

**New in version 13.7.**

**Parameters**

- **chat_id** *(int | str)* – Unique identifier for the target chat or username of the target supergroup or 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).
- **api_kwargs** *(dict, optional)* – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns** *Number of members in the chat.*

**Return type** *int*

**Raises** *telegram.error.TelegramError*
get_chat_members_count(chat_id, timeout=None, api_kwargs=None)

Deprecated, use get_chat_member_count() instead.

Deprecated since version 13.7.

get_file(file_id, timeout=None, api_kwargs=None)

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.

Note: This function may not preserve the original file name and MIME type. You should save the file’s MIME type and name (if available) when the File object is received.

Parameters

  - Either the file identifier or an object that has a file_id attribute to get file information about.

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

- api_kwargs (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns telegram.File

Raises telegram.error.TelegramError –

get_game_high_scores(user_id, chat_id=None, message_id=None, in-line_message_id=None, timeout=None, api_kwargs=None)

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

Note: This method will currently return scores for the target user, plus two of their closest neighbors on each side. Will also return the top three users if the user and his neighbors are not among them. Please note that this behavior is subject to change.

Parameters

- user_id (int) – Target user id.

- 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).

- api_kwargs (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.
Returns  List[telegram.GameHighScore]

Raises  telegram.error.TelegramError –

get_me (timeout=None, api_kwars=None)
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).

•  api_kwars (dict, optional) – Arbitrary keyword arguments to be passed to the
Telegram API.

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

Return type  telegram.User

Raises  telegram.error.TelegramError –

get_my_commands (timeout=None, api_kwars=None, scope=None, language_code=None)
Use this method to get the current list of the bot’s commands for the given scope and user language.

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).

•  api_kwars (dict, optional) – Arbitrary keyword arguments to be passed to the
Telegram API.

•  scope (telegram.BotCommandScope, optional) – A JSON-serialized object,
   New in version 13.7.

•  language_code (str, optional) – A two-letter ISO 639-1 language code or an
   empty string.
   New in version 13.7.

Returns  On success, the commands set for the bot. An empty list is returned if commands
are not set.

Return type  List[telegram.BotCommand]

Raises  telegram.error.TelegramError –

get_sticker_set (name, timeout=None, api_kwars=None)
Use this method to get a sticker set.

Parameters

•  name (str) – Name of the 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).

•  api_kwars (dict, optional) – Arbitrary keyword arguments to be passed to the
Telegram API.

Returns  telegram.StickerSet

Raises  telegram.error.TelegramError –
**get_updates** *(offset=None, limit=100, timeout=0, read_latency=2.0, allowed_updates=None, api_kwargs=None)*

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 `telegram.Update.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 be 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.

- **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. Defaults to 2.

- **allowed_updates** *(List[str], optional)* – A JSON-serialized 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 except `telegram.Update.chat_member` (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.

- **api_kwargs** *(dict, optional)* – Arbitrary keyword arguments to be passed to the Telegram API.

**Note:**

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.error.TelegramError*

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

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).

• **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns**  
telegram.UserProfilePhotos

**Raises**  
telegram.error.TelegramError

**get_webhook_info** (timeout=None, api_kwargs=None)

Use this method to get current webhook status. Requires no parameters.  
If the bot is using `get_updates()`, will return an object with the `telegram.WebhookInfo.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).

• **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns**  
telegram.WebhookInfo

**property id**

Unique identifier for this bot.

**Type**  
int

**kickChatMember** (chat_id, user_id, timeout=None, until_date=None, api_kwargs=None, revoke_messages=None)

Alias for `kick_chat_member()`

**kick_chat_member** (chat_id, user_id, timeout=None, until_date=None, api_kwargs=None, revoke_messages=None)

Deprecated, use `ban_chat_member()` instead.

Deprecated since version 13.7.

**property last_name**

Optional. Bot’s last name.

**Type**  
str

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

Alias for `leave_chat()`

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

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 supergroup or 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).

• **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

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

**Return type**  
bool

**Raises**  
telegram.error.TelegramError

3.2. telegram package 119
property link
Convenience property. Returns the t.me link of the bot.

Type str

logOut (timeout=None)
Alias for log_out()

log_out (timeout=None)
Use this method to log out from the cloud Bot API server before launching the bot locally. You must
log out the bot before running it locally, otherwise there is no guarantee that the bot will receive
updates. After a successful call, you can immediately log in on a local server, but will not be able to
log in back to the cloud Bot API server for 10 minutes.

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 On success
Return type True

Raises telegram.error.TelegramError

property name
Bot’s @username.

Type str

pinChatMessage (chat_id, message_id, disable_notification=None, timeout=None,
api_kwargs=None)
Alias for pin_chat_message()

pin_chat_message (chat_id, message_id, disable_notification=None, timeout=None,
api_kwargs=None)
Use this method to add a message to the list of pinned messages in a chat. If the chat is not a private
chat, the bot must be an administrator in the chat for this to work and must have the telegram.
ChatMember.can_pin_messages admin right in a supergroup or telegram.ChatMember.
can_edit_messages admin right in a channel.

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 chat members about the new pinned message. Notifications
are always disabled in channels and private chats.

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

• api_kwargs (dict, optional) – Arbitrary keyword arguments to be passed to the
Telegram API.

Returns On success, True is returned.

Return type bool

Raises telegram.error.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, can_manage_chat=None, can_manage_voice_chats=None)

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, can_manage_chat=None, can_manage_voice_chats=None)

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 channel (in the format @channelusername).
- **user_id (int)** – Unique identifier of the target user.
- **is_anonymous (bool, optional)** – Pass True, if the administrator’s presence in the chat is hidden.
- **can_manage_chat (bool, optional)** – Pass True, if the administrator can access the chat event log, chat statistics, message statistics in channels, see channel members, see anonymous administrators in supergroups and ignore slow mode. Implied by any other administrator privilege. New in version 13.4.
- **can_manage_voice_chats (bool, optional)** – Pass True, if the administrator can manage voice chats. New in version 13.4.
- **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 and can pin messages, 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).

• **api_kwargs** (`dict`, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

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

**Return type**  
`bool`

**Raises**  
`telegram.error.TelegramError`

**restrictChatMember** *(`chat_id`, `user_id`, `permissions`, `until_date=None`, `timeout=None`, `api_kwargs=None)*  

Alias for `restrict_chat_member()`

**restrict_chat_member** *(`chat_id`, `user_id`, `permissions`, `until_date=None`, `timeout=None`, `api_kwargs=None)*  

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.

**Note:** Since Bot API 4.4, `restrict_chat_member()` takes the new user permissions in a single argument of type `telegram.ChatPermissions`. The old way of passing parameters will not keep working forever.

**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. For timezone naive `datetime.datetime` objects, the default timezone of the bot will be used.

• **permissions** (`telegram.ChatPermissions`) – A JSON-serialized object for new user permissions.

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

• **api_kwargs** (`dict`, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

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

**Return type**  
`bool`

**Raises**  
`telegram.error.TelegramError`

**revokeChatInviteLink** *(`chat_id`, `invite_link`, `timeout=None`, `api_kwargs=None)*  

Alias for `revoke_chat_invite_link()`

**revoke_chat_invite_link** *(`chat_id`, `invite_link`, `timeout=None`, `api_kwargs=None)*  

Use this method to revoke an invite link created by the bot. If the primary link is revoked, a new link is automatically generated. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights.

New in version 13.4.
**Parameters**

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

- **invite_link**(str) – The invite link to edit.

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

- **api_kwargs**(dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns** `telegram.ChatInviteLink`

**Raises** `telegram.error.TelegramError`

```python
sendAnimation(**kwargs)  
sendAudio(**kwargs)  
sendChatAction(**kwargs)  
sendContact(**kwargs)  
sendDice(**kwargs)  
sendDocument(**kwargs)  
sendGame(**kwargs)
```

Alias for `send_animation()`

Alias for `send_audio()`

Alias for `send_chat_action()`

Alias for `send_contact()`

Alias for `send_dice()`

Alias for `send_document()`

Alias for `send_game()`
sendInvoice(chat_id, title, description, payload, provider_token, currency, prices, start_parameter=None, 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=None, reply_to_message_id=None, reply_markup=None, provider_data=None, send_phone_number_to_provider=None, send_email_to_provider=None, timeout=None, api_kwargs=None, allow_sending_without_reply=None, max_tip_amount=None, suggested_tip_amounts=None, protect_content=None)

Alias for send_invoice()

sendLocation(chat_id, latitude=None, longitude=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, location=None, live_period=None, api_kwargs=None, horizontal_accuracy=None, heading=None, proximity_alert_radius=None, allow_sending_without_reply=None, protect_content=None)

Alias for send_location()

sendMediaGroup(chat_id, media, disable_notification=None, reply_to_message_id=None, timeout=20, api_kwargs=None, allow_sending_without_reply=None, protect_content=None)

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, api_kwargs=None, entities=None, protect_content=None)

Alias for send_message()

sendPhoto(chat_id, photo, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, api_kwargs=None, allow_sending_without_reply=None, caption_entities=None, filename=None, protect_content=None)

Alias for send_photo()

sendPoll(chat_id, question, options, is_anonymous=True, type='regular', allows_multiple_answers=False, correct_option_id=None, is_closed=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, explanation=None, explanation_parse_mode=None, open_period=None, allow_sending_without_reply=None, correct_option_id=None, explanation_entities=None, protect_content=None)

Alias for send_poll()

sendSticker(chat_id, sticker, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, api_kwargs=None, allow_sending_without_reply=None, protect_content=None)

Alias for send_sticker()

sendVenue(chat_id, latitude=None, longitude=None, title=None, address=None, foursquare_id=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, venue=None, foursquare_type=None, api_kwargs=None, google_place_id=None, google_place_type=None, allow_sending_without_reply=None, protect_content=None)

Alias for send_venue()

sendVideo(chat_id, video, duration=None, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, width=None, height=None, parse_mode=None, supports_streaming=None, thumb=None, api_kwargs=None, allow_sending_without_reply=None, caption_entities=None, filename=None, protect_content=None)

Alias for send_video()
sendVideoNote(chat_id, video_note, duration=None, length=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, thumb=None, api_kwargs=None, allow_sending_without_reply=None, filename=None, protect_content=None)

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, api_kwargs=None, allow_sending_withoutReply=None, caption_entities=None, filename=None, protect_content=None)

Alias for send_voice()

send_animation(chat_id, animation, duration=None, width=None, height=None, thumb=None, caption=None, parse_mode=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, api_kwargs=None, allow_sending_without_reply=None, caption_entities=None, filename=None, protect_content=None)

Use this method to send animation files (GIF or H.264/MPEG-4 AVC video without sound). Bots can currently send animation files of up to 50 MB in size, this limit may be changed in the future.

Note: thumb will be ignored for small files, for which Telegram can easily generate thumb nails. However, this behaviour is undocumented and might be changed by Telegram.

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 | bytes | pathlib.Path | 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.

  Changed in version 13.2: Accept bytes as input.

- **filename** (str, optional) – Custom file name for the animation, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the tempfile module.


- **duration** (int, optional) – Duration of sent animation in seconds.

- **width** (int, optional) – Animation width.

- **height** (int, optional) – Animation height.

- **thumb** (filelike object | bytes | pathlib.Path, optional) – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can’t be reused and can be only uploaded as a new file.

  Changed in version 13.2: Accept bytes as input.

- **caption** (str, optional) – Animation caption (may also be used when resending animations by file_id), 0-1024 characters after entities parsing.

- **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.
• **caption_entities** (List[`telegram.MessageEntity`], optional) – List of special entities that appear in message text, which can be specified instead of `parse_mode`.

• **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.

• **protect_content** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

    New in version 13.10.

• **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.

• **allow_sending_without_reply** (bool, optional) – Pass `True`, if the message should be sent even if the specified replied-to message is not found.

• **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).

• **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.error.TelegramError` –

### send_audio

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 or .m4a format.

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 `send_voice()` 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 | bytes | `pathlib.Path` | `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.

    Changed in version 13.2: Accept `bytes` as input.

- **filename** (str, optional) – Custom file name for the audio, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the `tempfile` module.

• **caption** *(str, optional)* – Audio caption, 0-1024 characters after entities parsing.

• **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.

• **caption_entities** *(List[telegram.MessageEntity], optional)* – List of special entities that appear in message text, which can be specified instead of `parse_mode`.

• **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.

• **protect_content** *(bool, optional)* – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.

• **reply_to_message_id** *(int, optional)* – If the message is a reply, ID of the original message.

• **allow_sending_without_reply** *(bool, optional)* – Pass `True`, if the message should be sent even if the specified replied-to message is not found.

• **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 | bytes | pathlib.Path, optional)* – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can’t be reused and can be only uploaded as a new file.

  Changed in version 13.2: Accept `bytes` as input.

• **timeout** *(int | float, optional)* – Send file timeout (default: 20 seconds).

• **api_kwargs** *(dict, optional)* – Arbitrary keyword arguments to be passed to the Telegram API.

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.error.TelegramError`

`send_chat_action` *(chat_id, action, timeout=None, api_kwargs=None)*

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). Telegram only recommends using this method when a response from the bot will take a noticeable amount of time to arrive.

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`
send_contact(chat_id, phone_number=None, first_name=None, last_name=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, contact=None, vcard=None, api_kwargs=None, allow_sending_without_reply=None, protect_content=None)

Use this method to send phone contacts.

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

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **phone_number** (str, optional) – Contact’s phone number.
- **first_name** (str, optional) – Contact’s first name.
- **last_name** (str, optional) – Contact’s last name.
- **vcard** (str, optional) – Additional data about the contact in the form of a vCard, 0-2048 bytes.
- **contact** (telegram.Contact, optional) – The contact to send.
- **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

New in version 13.10.

- **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.
- **allow_sending_without_reply** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- **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).
- **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns On success, the sent Message is returned.

Return type telegram.Message

Raises telegram.error.TelegramError –
 send_dice(chat_id, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, emoji=None, api_kwargs=None, allow_sending_without_reply=None, protect_content=None)

Use this method to send an animated emoji that will display a random value.

Parameters

- **chat_id**(int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **emoji**(str, optional) – Emoji on which the dice throw animation is based. Currently, must be one of """, """, """, """, or """". Dice can have values 1-6 for """, """", and """, values 1-5 for """" and """, and values 1-64 for """". Defaults to """".

Changed in version 13.4: Added the """" emoji.
- **disable_notification**(bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content**(bool, optional) – Protects the contents of the sent message from forwarding and saving.

New in version 13.10.
- **reply_to_message_id**(int, optional) – If the message is a reply, ID of the original message.
- **allow_sending_without_reply**(bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- **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).
- **api_kwargs**(dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns On success, the sent Message is returned.

Return type telegram.Message

Raises telegram.error.TelegramError –

 send_document(chat_id, document, filename=None, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, thumb=None, api_kwargs=None, disable_content_type_detection=None, allow_sending_without_reply=None, caption_entities=None, protect_content=None)

Use this method to send general files.

Bots can currently send files of any type of up to 50 MB in size, this limit may be changed in the future.

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 | bytes | pathlib.Path | 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.

   Changed in version 13.2: Accept *bytes* as input.

• **filename** *(str, optional)* – Custom file name for the document, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the *tempfile* module.

• **caption** *(str, optional)* – Document caption (may also be used when resending documents by file_id), 0-1024 characters after entities parsing.

• **disable_content_type_detection** *(bool, optional)* – Disables automatic server-side content type detection for files uploaded using multipart/form-data.

• **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.

• **caption_entities** *(List[telegram.MessageEntity], optional)* – List of special entities that appear in message text, which can be specified instead of *parse_mode*.

• **disable_notification** *(bool, optional)* – Sends the message silently. Users will receive a notification with no sound.

• **protect_content** *(bool, optional)* – Protects the contents of the sent message from forwarding and saving.

   New in version 13.10.

• **reply_to_message_id** *(int, optional)* – If the message is a reply, ID of the original message.

• **allow_sending_without_reply** *(bool, optional)* – Pass *True*, if the message should be sent even if the specified replied-to message is not found.

• **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|bytes|pathlib.Path, optional)* – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can’t be reused and can be only uploaded as a new file.

   Changed in version 13.2: Accept *bytes* as input.

• **timeout** *(int | float, optional)* – Send file timeout (default: 20 seconds).

• **api_kwargs** *(dict, optional)* – Arbitrary keyword arguments to be passed to the Telegram API.

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

**Return type** *telegram.Message*

**Raises** *telegram.error.TelegramError* –

`send_game` *(chat_id, game_short_name, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, api_kwargs=None, allow_sending_without_reply=None, protect_content=None)*

Use this method to send a game.
Parameters

- `chat_id` (int | str) – Unique identifier for the target chat.
- `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.
- `protect_content` (bool, optional) – Protects the contents of the sent message from forwarding and saving.
  New in version 13.10.
- `reply_to_message_id` (int, optional) – If the message is a reply, ID of the original message.
- `allow_sending_without_reply` (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- `reply_markup` (telegram.InlineKeyboardMarkup, optional) – A JSON-serialized object for a new inline keyboard. If empty, one ‘Play game_title’ button will be shown. If not empty, the first button must launch the game.
- `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- `api_kwargs` (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns
On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.error.TelegramError` –

`send_invoice` (chat_id, title, description, payload, provider_token, currency, prices, start_parameter=None, 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=None, reply_to_message_id=None, reply_markup=None, provider_data=None, send_phone_number_to_provider=None, send_email_to_provider=None, timeout=None, api_kwargs=None, allow_sending_without_reply=None, max_tip_amount=None, suggested_tip_amounts=None, protect_content=None)

Use this method to send invoices.

**Warning:** As of API 5.2 `start_parameter` is an optional argument and therefore the order of the arguments had to be changed. Use keyword arguments to make sure that the arguments are passed correctly.

Changed in version 13.5: As of Bot API 5.2, the parameter `start_parameter` is optional.

Parameters

- `chat_id` (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- `title` (str) – Product name, 1-32 characters.
- `description` (str) – Product description, 1-255 characters.
- `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.

• **currency** *(str)* – Three-letter ISO 4217 currency code.

• **prices** *(List[telegram.LabeledPrice])* – Price breakdown, a JSON-serialized list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.).

• **max_tip_amount** *(int, optional)* – The maximum accepted amount for tips in the smallest units of the currency (integer, not float/double). For example, for a maximum tip of US$ 1.45 pass `max_tip_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). Defaults to 0.

    New in version 13.5.

• **suggested_tip_amounts** *(List[int], optional)* – A JSON-serialized array of suggested amounts of tips in the smallest units of the currency (integer, not float/double). At most 4 suggested tip amounts can be specified. The suggested tip amounts must be positive, passed in a strictly increased order and must not exceed `max_tip_amount`.

    New in version 13.5.

• **start_parameter** *(str, optional)* – Unique deep-linking parameter. If left empty, forwarded copies of the sent message will have a Pay button, allowing multiple users to pay directly from the forwarded message, using the same invoice. If non-empty, forwarded copies of the sent message will have a URL button with a deep link to the bot (instead of a Pay button), with the value used as the start parameter.

    Changed in version 13.5: As of Bot API 5.2, this parameter is optional.

• **provider_data** *(str | object, optional)* – JSON-serialized 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.

• **protect_content** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.

• **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.

• **allow_sending_without_reply** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.

• **reply_markup** (**telegram.InlineKeyboardMarkup**, optional) – A JSON-serialized object for 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).

• **api_kwargs** (**dict**, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

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

**Return type** **telegram.Message**

**Raises** **telegram.error.TelegramError** –

**send_location** (**chat_id**, latitude=None, longitude=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, location=None, live_period=None, api_kwargs=None, horizontal_accuracy=None, heading=None, proximity_alert_radius=None, allow_sending_without_reply=None, protect_content=None)

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.

• **horizontal_accuracy** (**int**, optional) – The radius of uncertainty for the location, measured in meters; 0-1500.

• **live_period** (**int**, optional) – Period in seconds for which the location will be updated, should be between 60 and 86400.

• **heading** (**int**, optional) – For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.

• **proximity_alert_radius** (**int**, optional) – For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and 100000 if specified.

• **disable_notification** (**bool**, optional) – Sends the message silently. Users will receive a notification with no sound.
• `protect_content` (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.

• `reply_to_message_id` (int, optional) – If the message is a reply, ID of the original message.

• `allow_sending_without_reply` (bool, optional) – Pass `True`, if the message should be sent even if the specified replied-to message is not found.

• `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).

• `api_kwargs` (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.error.TelegramError` –

**send_media_group** *(chat_id, media, disable_notification=None, reply_to_message_id=None, timeout=20, api_kwargs=None, allow_sending_without_reply=None, protect_content=None)*

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.InputMediaAudio, telegram.InputMediaDocument, telegram.InputMediaPhoto, telegram.InputMediaVideo]) – An array describing messages to be sent, must include 2–10 items.

• `disable_notification` (bool, optional) – Sends the message silently. Users will receive a notification with no sound.

• `protect_content` (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.

• `reply_to_message_id` (int, optional) – If the message is a reply, ID of the original message.

• `allow_sending_without_reply` (bool, optional) – Pass `True`, if the message should be sent even if the specified replied-to message is not found.

• `timeout` (int | float, optional) – Send file timeout (default: 20 seconds).

• `api_kwargs` (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns An array of the sent Messages.

Return type List[telegram.Message]

Raises `telegram.error.TelegramError` –
send_message(chat_id, text, parse_mode=None, disable_web_page_preview=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, api_kwargs=None, allow_sending_without_reply=None, entities=None, protect_content=None)

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 after entities parsing. 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.
- **entities** (List[telegram.MessageEntity], optional) – List of special entities that appear in message text, which can be specified instead of parse_mode.
- **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.
- **protect_content** (bool, optional) – Protects the contents of sent messages from forwarding and saving.
  New in version 13.10.
- **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.
- **allow_sending_without_reply** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- **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).
- **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, the sent message is returned.

Return type **telegram.Message**

Raises **telegram.error.TelegramError** –

send_photo(chat_id, photo, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, api_kwargs=None, allow_sending_without_reply=None, caption_entities=None, filename=None, protect_content=None)

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 | file) – Photo to send. Providing a file_id is very efficient, because the photo will be uploaded only once. Alternatively, you can pass a URL or a file from disk open(filename, 'rb')
- **caption** (str, optional) – Caption of the photo 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).
- **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. See the constants in telegram.ParseMode for the available modes.
- **caption_entities** (List[telegram.MessageEntity], optional) – List of special entities that appear in message text, which can be specified instead of parse_mode.
- **filename** (str, optional) – Photo filename.
- **protect_content** (bool, optional) – Protects the contents of sent messages from forwarding and saving.
  New in version 13.10.
• **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).

• **photo** (str | filelike object | bytes | pathlib.Path | 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.

  Changed in version 13.2: Accept bytes as input.

• **filename** (str, optional) – Custom file name for the photo, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the tempfile module.


• **caption** (str, optional) – Photo caption (may also be used when resending photos by file_id), 0-1024 characters after entities parsing.

• **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.

• **caption_entities** (List[telegram.MessageEntity], optional) – List of special entities that appear in message text, which can be specified instead of parse_mode.

• **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.

• **protect_content** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.

• **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.

• **allow_sending_without_reply** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.

• **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).

• **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns On success, the sent Message is returned.

Return type telegram.Message

Raises telegram.error.TelegramError –

**send_poll** *(chat_id, question, options, is_anonymous=True, type='regular', allows_multiple_answers=False, correct_option_id=None, is_closed=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, explanation=None, explanation_parse_mode=None, open_period=None, close_date=None, api_kwargs=None, allow_sending_without_reply=None, explanation_entities=None, protect_content=None)*

Use this method to send a native poll.

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

• `question(str)` – Poll question, 1-300 characters.

• `options(List[str])` – List of answer options, 2-10 strings 1-100 characters each.

• `is_anonymous(bool, optional)` – True, if the poll needs to be anonymous, defaults to True.

• `type(str, optional)` – Poll type, `telegram.Poll.QUIZ` or `telegram.Poll.REGULAR`, defaults to `telegram.Poll.REGULAR`.

• `allows_multiple_answers(bool, optional)` – True, if the poll allows multiple answers, ignored for polls in quiz mode, defaults to False.

• `correct_option_id(int, optional)` – 0-based identifier of the correct answer option, required for polls in quiz mode.

• `explanation(str, optional)` – Text that is shown when a user chooses an incorrect answer or taps on the lamp icon in a quiz-style poll, 0-200 characters with at most 2 line feeds after entities parsing.

• `explanation_parse_mode(str, optional)` – Mode for parsing entities in the explanation. See the constants in `telegram.ParseMode` for the available modes.

• `explanation_entities(List[telegram.MessageEntity], optional)` – List of special entities that appear in message text, which can be specified instead of `parse_mode`.

• `open_period(int, optional)` – Amount of time in seconds the poll will be active after creation, 5-600. Can’t be used together with `close_date`.

• `close_date(int | datetime.datetime, optional)` – Point in time (Unix timestamp) when the poll will be automatically closed. Must be at least 5 and no more than 600 seconds in the future. Can’t be used together with `open_period`. For timezone naive `datetime.datetime` objects, the default timezone of the bot will be used.

• `is_closed(bool, optional)` – Pass True, if the poll needs to be immediately closed. This can be useful for poll preview.

• `disable_notification(bool, optional)` – Sends the message silently. Users will receive a notification with no sound.

• `protect_content(bool, optional)` – Protects the contents of the sent message from forwarding and saving.

New in version 13.10.

• `reply_to_message_id(int, optional)` – If the message is a reply, ID of the original message.

• `allow_sending_without_reply(bool, optional)` – Pass True, if the message should be sent even if the specified replied-to message is not found.

• `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).

• `api_kwargs(dict, optional)` – Arbitrary keyword arguments to be passed to the Telegram API.

`Returns` On success, the sent Message is returned.
Return type `telegram.Message`

Raises `telegram.error.TelegramError`

**send_sticker**

```python
(chat_id, sticker, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, api_kwargs=None, allow_sending_without_reply=None, protect_content=None)
```

Use this method to send static .WEBP, animated .TGS, or video .WEBM 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` | `bytes` | `pathlib.Path` | `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.
  
  Changed in version 13.2: Accept `bytes` as input.
- **disable_notification** (`bool`, optional) – Sends the message silently. Users will receive a notification with no sound.
- **protect_content** (`bool`, optional) – Protects the contents of the sent message from forwarding and saving.
  
  New in version 13.10.
- **reply_to_message_id** (`int`, optional) – If the message is a reply, ID of the original message.
- **allow_sending_without_reply** (`bool`, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.
- **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).
- **api_kwargs** (`dict`, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

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

**Return type** `telegram.Message`

**Raise** `telegram.error.TelegramError`

**send_venue**

```python
(chat_id, latitude=None, longitude=None, title=None, address=None, foursquare_id=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, venue=None, foursquare_type=None, api_kwargs=None, google_place_id=None, google_place_type=None, allow_sending_without_reply=None, protect_content=None)
```

Use this method to send information about a venue.

**Note:** The sticker argument can be either a file_id, an URL or a file from disk `open(filename, 'rb')`
• You can either supply `venue`, or `latitude`, `longitude`, `title` and `address` and optionally `foursquare_id` and `foursquare_type` or optionally `google_place_id` and `google_place_type`.

• Foursquare details and Google Place details are mutually exclusive. However, this behaviour is undocumented and might be changed by Telegram.

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”)
- `google_place_id` (str, optional) – Google Places identifier of the venue.
- `google_place_type` (str, optional) – Google Places type of the venue. (See supported types.)
- `venue` (`telegram.Venue`, optional) – The venue to send.
- `disable_notification` (bool, optional) – Sends the message silently. Users will receive a notification with no sound.
- `protect_content` (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.
- `reply_to_message_id` (int, optional) – If the message is a reply, ID of the original message.
- `allow_sending_without_reply` (bool, optional) – Pass `True`, if the message should be sent even if the specified replied-to message is not found.
- `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).
- `api_kwargs` (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, the sent `Message` is returned.

Return type `telegram.Message`

Raises `telegram.error.TelegramError`
send_video(chat_id, video, duration=None, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, width=None, height=None, parse_mode=None, supports_streaming=None, thumb=None, api_kwargs=None, allow_sending_without_reply=None, filename=None, protect_content=None)

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

Bots can currently send video files of up to 50 MB in size, this limit may be changed in the future.

---

**Note:**

- The video argument can be either a file_id, an URL or a file from disk `open(filename, 'rb')`
- `thumb` will be ignored for small video files, for which Telegram can easily generate thumbnails. However, this behaviour is undocumented and might be changed by Telegram.

---

**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 | bytes | pathlib.Path | 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.

  Changed in version 13.2: Accept `bytes` as input.
- **filename** (str, optional) – Custom file name for the video, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the `tempfile` module.

- **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 after entities parsing.
- **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.
- **caption_entities** (List[telegram.MessageEntity], optional) – List of special entities that appear in message text, which can be specified instead of `parse_mode`.
- **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.
- **protect_content** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

New in version 13.10.
• **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.

• **allow_sending_without_reply** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.

• **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|bytes|pathlib.Path*, optional) – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can’t be reused and can be only uploaded as a new file.

    Changed in version 13.2: Accept bytes as input.

• **timeout** (int|float, optional) – Send file timeout (default: 20 seconds).

• **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns**

On success, the sent Message is returned.

**Return type** *telegram.Message*

**Raises** *telegram.error.TelegramError*

### send_video_note

.. code-block::

    send_video_note(chat_id, video_note, duration=None, length=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, thumb=None, api_kwargs=None, allow_sending_without_reply=None, filename=None, protect_content=None)

As of v.4.0, Telegram clients support rounded square mp4 videos of up to 1 minute long. 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')
- thumb will be ignored for small video files, for which Telegram can easily generate thumbnail nails. However, this behaviour is undocumented and might be changed by Telegram.

**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|bytes|pathlib.Path | *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.

    Changed in version 13.2: Accept bytes as input.

- **filename** (str, optional) – Custom file name for the video note, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the tempfile module.


- **duration** (int, optional) – Duration of sent video in seconds.
• **length** (int, optional) – Video width and height, i.e. diameter of the video message.

• **disable_notification** (bool, optional) – Sends the message silently. Users will receive a notification with no sound.

• **protect_content** (bool, optional) – Protects the contents of the sent message from forwarding and saving.

  New in version 13.10.

• **reply_to_message_id** (int, optional) – If the message is a reply, ID of the original message.

• **allow_sending_without_reply** (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.

• **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 | bytes | pathlib.Path, optional) – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can’t be reused and can be only uploaded as a new file.

  Changed in version 13.2: Accept bytes as input.

• **timeout** (int | float, optional) – Send file timeout (default: 20 seconds).

• **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns On success, the sent Message is returned.

Return type telegram.Message

Raises telegram.error.TelegramError –

**send_voice** (chat_id, voice, duration=None, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, api_kwargs=None, allow_sending_without_reply=None, caption_entities=None, filename=None, protect_content=None)

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). Bots can currently send voice messages of up to 50 MB in size, this limit may be changed in the future.

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 | bytes | pathlib.Path | 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.

  Changed in version 13.2: Accept bytes as input.
• `filename` (str, optional) – Custom file name for the voice, when uploading a new file. Convenience parameter, useful e.g., when sending files generated by the `tempfile` module.


• `caption` (str, optional) – Voice message caption, 0-1024 characters after entities parsing.

• `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.

• `caption_entities` (List[telegram.MessageEntity], optional) – List of special entities that appear in message text, which can be specified instead of `parse_mode`.

• `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.

• `protect_content` (bool, optional) – Protects the contents of the sent message from forwarding and saving.

    New in version 13.10.

• `reply_to_message_id` (int, optional) – If the message is a reply, ID of the original message.

• `allow_sending_without_reply` (bool, optional) – Pass True, if the message should be sent even if the specified replied-to message is not found.

• `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).

• `api_kwargs` (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns On success, the sent Message is returned.

Return type `telegram.Message`

Raises `telegram.error.TelegramError`

`setChatAdministratorCustomTitle` *(chat_id, user_id, custom_title, timeout=None, api_kwargs=None)*

Alias for `set_chat_administrator_custom_title()`

`setChatDescription` *(chat_id, description, timeout=None, api_kwargs=None)*

Alias for `set_chat_description()`

`setChatPermissions` *(chat_id, permissions, timeout=None, api_kwargs=None)*

Alias for `set_chat_permissions()`

`setChatPhoto` *(chat_id, photo, timeout=20, api_kwargs=None)*

Alias for `set_chat_photo()`

`setChatStickerSet` *(chat_id, sticker_set_name, timeout=None, api_kwargs=None)*

Alias for `set_chat_sticker_set()`

`setChatTitle` *(chat_id, title, timeout=None, api_kwargs=None)*

Alias for `set_chat_title()`

---

3.2. telegram package 143
setGameScore(user_id, score, chat_id=None, message_id=None, inline_message_id=None, force=None, disable_edit_message=None, timeout=None, api_kwargs=None)

Alias for set_game_score()

setMyCommands(commands, timeout=None, api_kwargs=None, scope=None, language_code=None)

Alias for set_my_commands()

setPassportDataErrors(user_id, errors, timeout=None, api_kwargs=None)

Alias for set_passport_data_errors()

setStickerPositionInSet(sticker, position, timeout=None, api_kwargs=None)

Alias for set_sticker_position_in_set()

setStickerSetThumb(name, user_id, thumb=None, timeout=None, api_kwargs=None)

Alias for set_sticker_set_thumb()

setWebhook(url=None, certificate=None, timeout=None, max_connections=40, allowed_updates=None, api_kwargs=None, ip_address=None, drop_pending_updates=None)

Alias for set_webhook()

set_chat_administrator_custom_title(chat_id, user_id, custom_title, timeout=None, api_kwargs=None)

Use this method to set a custom title for administrators promoted by the bot in a supergroup. The bot
must be an administrator for this to work.

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 administrator.

• custom_title (str) – New custom title for the administrator; 0-16 characters, emoji
  are not 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).

• api_kwargs (dict, optional) – Arbitrary keyword arguments to be passed to the
  Telegram API.

Returns On success, True is returned.

Return type bool

Raises telegram.error.TelegramError –

set_chat_description(chat_id, description, timeout=None, api_kwargs=None)

Use this method to change the description of a group, 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, 0-255 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).

• api_kwargs (dict, optional) – Arbitrary keyword arguments to be passed to the
  Telegram API.

Returns On success, True is returned.
set_chat_permissions

Use this method to set default chat permissions for all members. The bot must be an administrator in the group or a supergroup for this to work and must have the `telegram.ChatMember.can_restrict_members` admin rights.

**Parameters**

- `chat_id` (int | str) – Unique identifier for the target chat or username of the target supergroup (in the format `@supergroupusername`).
- `permissions` (telegram.ChatPermissions) – New default chat permissions.
- `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- `api_kwargs` (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns**

On success, `True` is returned.

Return type `bool`

Raises `telegram.error.TelegramError` –

set_chat_photo

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 | bytes | pathlib.Path) – New chat photo.
  Changed in version 13.2: Accept `bytes` as input.
- `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- `api_kwargs` (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns**

On success, `True` is returned.

Return type `bool`

Raises `telegram.error.TelegramError` –

set_chat_sticker_set

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.
set_chat_title(chat_id, title, timeout=None, api_kwargs=None)

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).
• api_kwargs (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns On success, True is returned.

Return type bool

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

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

Parameters

• user_id (int) – User identifier.
• score (int) – New score, must be non-negative.
• 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).
• api_kwargs (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

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

Return type telegram.Message
Raises `telegram.error.TelegramError` – If the new score is not greater than the user’s current score in the chat and `force` is `False`.

**set_my_commands** *(commands, timeout=None, api_kwargs=None, scope=None, language_code=None)*

Use this method to change the list of the bot’s commands. See the Telegram docs for more details about bot commands.

**Parameters**

- **commands** *(List[BotCommand|str, str])* – A JSON-serialized list of bot commands to be set as the list of the bot’s commands. At most 100 commands can be specified.
- **timeout** *(int|float, optional)* – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **api_kwargs** *(dict, optional)* – Arbitrary keyword arguments to be passed to the Telegram API.
- **language_code** *(str, optional)* – A two-letter ISO 639-1 language code. If empty, commands will be applied to all users from the given scope, for whose language there are no dedicated commands. New in version 13.7.

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

**Return type** `bool`

**Raises** `telegram.error.TelegramError` –

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

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).

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).
- **api_kwargs** *(dict, optional)* – Arbitrary keyword arguments to be passed to the Telegram API.

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

**Return type** `bool`

**Raises** `telegram.error.TelegramError` –
set_sticker_position_in_set(sticker, position, timeout=None, api_kwargs=None)

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).
- **api_kwargs** *(dict, optional)* – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, True is returned.

Return type *bool*

Raises *telegram.error.TelegramError*

set_sticker_set_thumb(name, user_id, thumb=None, timeout=None, api_kwargs=None)

Use this method to set the thumbnail of a sticker set. Animated thumbnails can be set for animated sticker sets only. Video thumbnails can be set only for video sticker sets only.

Parameters

- **name** *(str)* – Sticker set name
- **user_id** *(int)* – User identifier of created sticker set owner.
- **thumb** *(str | filelike object | bytes | pathlib.Path, optional)* – A PNG image with the thumbnail, must be up to 128 kilobytes in size and have width and height exactly 100px, or a TGS animation with the thumbnail up to 32 kilobytes in size; see https://core.telegram.org/stickers#animated-sticker-requirements for animated sticker technical requirements, or a WEBM video with the thumbnail up to 32 kilobytes in size; see https://core.telegram.org/stickers#video-sticker-requirements for video sticker technical requirements. 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. Animated sticker set thumbnails can’t be uploaded via HTTP URL.
  
  Changed in version 13.2: Accept bytes as input.
- **timeout** *(int | float, optional)* – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **api_kwargs** *(dict, optional)* – Arbitrary keyword arguments to be passed to the Telegram API.

Returns

On success, True is returned.

Return type *bool*

Raises *telegram.error.TelegramError*

set_webhook(url=None, certificate=None, timeout=None, max_connections=40, allowed_updates=None, api_kwargs=None, ip_address=None, drop_pending_updates=None)

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

If you’d like to make sure that the Webhook request comes from Telegram, Telegram recommends 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](https://goo.gl/rw7w6Y))

- **ip_address** *(str, optional)* – The fixed IP address which will be used to send webhook requests instead of the IP address resolved through DNS.

- **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)* – A JSON-serialized 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 except `telegram.Update.chat_member` (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.

- **drop_pending_updates** *(bool, optional)* – Pass True to drop all pending updates.

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

- **api_kwargs** *(dict, optional)* – Arbitrary keyword arguments to be passed to the Telegram API.

Note:

1. You will not be able to receive updates using `get_updates()` for 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.

If you’re having any trouble setting up webhooks, please check out this guide to Webhooks.

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

**Raises** `telegram.error.TelegramError`
**stopMessageLiveLocation**

```python
stopMessageLiveLocation(chat_id=None, message_id=None, inline_message_id=None, reply_markup=None, timeout=None, api_kwargs=None)
```

Alias for `stop_message_live_location()`

**stopPoll**

```python
stopPoll(chat_id, message_id, reply_markup=None, timeout=None, api_kwargs=None)
```

Alias for `stop_poll()`

**stop_message_live_location**

```python
stop_message_live_location(chat_id=None, message_id=None, inline_message_id=None, reply_markup=None, timeout=None, api_kwargs=None)
```

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) – Required if inline_message_id is not specified. Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- `message_id` (int, optional) – Required if inline_message_id is not specified. Identifier of the sent message with live location to stop.
- `inline_message_id` (str, optional) – Required if chat_id and message_id are not specified. Identifier of the inline message.
- `reply_markup` (telegram.InlineKeyboardMarkup, optional) – A JSON-serialized object for a new inline keyboard.
- `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- `api_kwargs` (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns**

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

**Return type** `telegram.Message`

**stop_poll**

```python
stop_poll(chat_id, message_id, reply_markup=None, timeout=None, api_kwargs=None)
```

Use this method to stop a poll which was sent by the bot.

**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 original message with the poll.
- `reply_markup` (telegram.InlineKeyboardMarkup, optional) – A JSON-serialized object for a new message inline keyboard.
- `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- `api_kwargs` (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns**

On success, the stopped Poll with the final results is returned.

**Return type** `telegram.Poll`

**Raises** `telegram.error.TelegramError` –

**property supports_inline_queries**

Bot’s `telegram.User.supports_inline_queries` attribute.

**Type** `bool`
to_dict()  
See telegram.TelegramObject.to_dict().

unbanChatMember(chat_id, user_id, timeout=None, api_kwargs=None, only_if_banned=None)  
Alias for unban_chat_member()  
unbanChatSenderChat(chat_id, sender_chat_id, timeout=None, api_kwargs=None)  
Alias for unban_chat_sender_chat()  
unban_chat_member(chat_id, user_id, timeout=None, api_kwargs=None, only_if_banned=None)  
Use this method to unban a previously kicked user in a supergroup or channel.  
The user will not return to the group or channel automatically, but will be able to join via link, etc. The bot must be an administrator for this to work. By default, this method guarantees that after the call the user is not a member of the chat, but will be able to join it. So if the user is a member of the chat they will also be removed from the chat. If you don’t want this, use the parameter only_if_banned.

Parameters
- chat_id (int | str) – Unique identifier for the target chat or username of the target supergroup or channel (in the format @channelusername).
- user_id (int) – Unique identifier of the target user.
- only_if_banned (bool, optional) – Do nothing if the user is not banned.
- timeout (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- api_kwargs (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns On success, True is returned.
Return type bool

Raises telegram.error.TelegramError –

unban_chat_sender_chat(chat_id, sender_chat_id, timeout=None, api_kwargs=None)  
Use this method to unban a previously banned channel in a supergroup or channel. The bot must be an administrator for this to work and must have the appropriate administrator rights.

New in version 13.9.

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

Returns On success, True is returned.
Return type bool

Raises telegram.error.TelegramError –

unpinAllChatMessages(chat_id, timeout=None, api_kwargs=None)  
Alias for unpin_all_chat_messages()  
unpinChatMessage(chat_id, timeout=None, api_kwargs=None, message_id=None)  
Alias for unpin_chat_message()
unpin_all_chat_messages(chat_id, timeout=None, api_kwargs=None)

Use this method to clear the list of pinned messages in a chat. If the chat is not a private chat, the bot must be an administrator in the chat for this to work and must have the `telegram.ChatMember.can_pin_messages` admin right in a supergroup or `telegram.ChatMember.can_edit_messages` admin right in a 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).
- **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

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

**Return type** bool

unpin_chat_message(chat_id, timeout=None, api_kwargs=None, message_id=None)

Use this method to remove a message from the list of pinned messages in a chat. If the chat is not a private chat, the bot must be an administrator in the chat for this to work and must have the `telegram.ChatMember.can_pin_messages` admin right in a supergroup or `telegram.ChatMember.can_edit_messages` admin right in a channel.

**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) – Identifier of a message to unpin. If not specified, the most recent pinned message (by sending date) will be unpinned.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

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

**Return type** bool

Raises `telegram.error.TelegramError` –

uploadStickerFile

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 | bytes | pathlib.Path) –* 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.

  Changed in version 13.2: Accept bytes as input.

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

• **`api_kwargs`** *(dict, optional) –* Arbitrary keyword arguments to be passed to the Telegram API.

  Returns On success, the uploaded File is returned.

  Return type *telegram.File*

  Raises *telegram.error.TelegramError –

  **property username**
  
  Bot’s username.

  Type *str*

### 3.2.4 telegram.BotCommand

class telegram.BotCommand(*command, description, **kwargs*)

  Bases: telegram.base.TelegramObject

  This object represents a bot command.

  Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `command` and `description` are equal.

  Parameters

  • **`command`** *(str) –* Text of the command, 1-32 characters. Can contain only lowercase English letters, digits and underscores.

  • **`description`** *(str) –* Description of the command, 1-256 characters.

  **command**
  
  Text of the command.

  Type *str*

  **description**
  
  Description of the command.

  Type *str*

### 3.2.5 telegram.BotCommandScope

class telegram.BotCommandScope(*type, **kwargs*)

  Bases: telegram.base.TelegramObject

  Base class for objects that represent the scope to which bot commands are applied. Currently, the following 7 scopes are supported:

  • **`telegram.BotCommandScopeDefault`**

  • **`telegram.BotCommandScopeAllPrivateChats`**

  • **`telegram.BotCommandScopeAllGroupChats`**

  • **`telegram.BotCommandScopeAllChatAdministrators`**

3.2. telegram package
Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `type` is equal. For subclasses with additional attributes, the notion of equality is overridden.

**Note:** Please see the official docs on how Telegram determines which commands to display.

New in version 13.7.

**Parameters**

- `type (str)` – Scope type.

**Type**

- `str`

```python
ALL_CHAT_ADMINISTRATORS = 'all_chat_administrators'
telegram.constants.BOT_COMMAND_SCOPE_ALL_CHAT_ADMINISTRATORS

ALL_GROUP_CHATS = 'all_group_chats'
telegram.constants.BOT_COMMAND_SCOPE_ALL_GROUP_CHATS

ALL_PRIVATE_CHATS = 'all_private_chats'
telegram.constants.BOT_COMMAND_SCOPE_ALL_PRIVATE_CHATS

CHAT = 'chat'
telegram.constants.BOT_COMMAND_SCOPE_CHAT

CHAT_ADMINISTRATORS = 'chat_administrators'
telegram.constants.BOT_COMMAND_SCOPE_CHAT_ADMINISTRATORS

CHAT_MEMBER = 'chat_member'
telegram.constants.BOT_COMMAND_SCOPE_CHAT_MEMBER

DEFAULT = 'default'
telegram.constants.BOT_COMMAND_SCOPE_DEFAULT
```

**Method de_json (data, bot)**

Converts JSON data to the appropriate `BotCommandScope` object, i.e. takes care of selecting the correct subclass.

**Parameters**

- `data (Dict[str, ...])` – The JSON data.
- `bot (telegram.Bot)` – The bot associated with this object.

**Returns**

The Telegram object.

### 3.2.6 `telegram.BotCommandScopeDefault`

**class**

- `telegram.BotCommandScopeDefault (**kwargs)`

**Bases:**

- `telegram.botcommandscope.BotCommandScope`

Represents the default scope of bot commands. Default commands are used if no commands with a narrower scope are specified for the user.

New in version 13.7.

**Parameters**

- `type (Scope type telegram.BotCommandScope.DEFAULT)`

**Type**

- `str`
3.2.7 telegram.BotCommandScopeAllPrivateChats

class telegram.BotCommandScopeAllPrivateChats(**kwargs)
    Bases: telegram.botcommandscope.BotCommandScope
    Represents the scope of bot commands, covering all private chats.
    New in version 13.7.

    type
    Scope type telegram.BotCommandScope.ALL_PRIVATE_CHATS.
    Type str

3.2.8 telegram.BotCommandScopeAllGroupChats

class telegram.BotCommandScopeAllGroupChats(**kwargs)
    Bases: telegram.botcommandscope.BotCommandScope
    Represents the scope of bot commands, covering all group and supergroup chats.
    New in version 13.7.

    type
    Scope type telegram.BotCommandScope.ALL_GROUP_CHATS.
    Type str

3.2.9 telegram.BotCommandScopeAllChatAdministrators

class telegram.BotCommandScopeAllChatAdministrators(**kwargs)
    Bases: telegram.botcommandscope.BotCommandScope
    Represents the scope of bot commands, covering all group and supergroup chat administrators.
    New in version 13.7.

    type
    Scope type telegram.BotCommandScope.ALL_CHAT_ADMINISTRATORS.
    Type str

3.2.10 telegram.BotCommandScopeChat

class telegram.BotCommandScopeChat(chat_id, **kwargs)
    Bases: telegram.botcommandscope.BotCommandScope
    Represents the scope of bot commands, covering a specific chat.
    Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their type and chat_id are equal.
    New in version 13.7.

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

    type
    Scope type telegram.BotCommandScope.CHAT.
    Type str

    chat_id
    Unique identifier for the target chat or username of the target supergroup (in the format
    @supergroupusername)
3.2.11 telegram.BotCommandScopeChatAdministrators

class telegram.BotCommandScopeChatAdministrators(chat_id, **kwargs)
    Bases: telegram.botcommandscope.BotCommandScope

    Represents the scope of bot commands, covering all administrators of a specific group or supergroup chat.
    Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if
    their type and chat_id are equal.
    New in version 13.7.

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

    type
        Scope type telegram.BotCommandScope.CHAT_ADMINISTRATORS.

    chat_id
        Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername)

        Type str|int

3.2.12 telegram.BotCommandScopeChatMember

class telegram.BotCommandScopeChatMember(chat_id, user_id, **kwargs)
    Bases: telegram.botcommandscope.BotCommandScope

    Represents the scope of bot commands, covering a specific member of a group or supergroup chat.
    Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if
    their type, chat_id and user_id are equal.
    New in version 13.7.

    Parameters

    • chat_id (str|int) – 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.

    type
        Scope type telegram.BotCommandScope.CHAT_MEMBER.

    chat_id
        Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername)

        Type str|int

    user_id
        Unique identifier of the target user.

        Type int
3.2.13 `telegram.CallbackQuery`

```python
class telegram.CallbackQuery(id, from_user, chat_instance, message=None, data=None, inline_message_id=None, game_short_name=None, bot=None, **kwargs)
```

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.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `id` is equal.

**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.
- 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).
- If you’re using `Bot.arbitrary_callback_data`, `data` may be an instance of `telegram.ext.InvalidCallbackData`. This will be the case, if the data associated with the button triggering the `telegram.CallbackQuery` was already deleted or if `data` was manipulated by a malicious client.

New in version 13.6.

**Parameters**

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

**id**

Unique identifier for this query.

**Type** str

**from_user**

Sender.
Type `telegram.User`

**chat_instance**
Global identifier, uniquely corresponding to the chat to which the message with the callback button was sent.
Type `str`

**message**
Optional. Message with the callback button that originated the query.
Type `telegram.Message`

**data**
Optional. Data associated with the callback button.
Type `str` | `object`

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

**game_short_name**
Optional. Short name of a Game to be returned.
Type `str`

**bot**
The Bot to use for instance methods.
Type `telegram.Bot`, optional

`MAX_ANSWER_TEXT_LENGTH`: `ClassVar[int] = 200`
`telegram.constants.MAX_ANSWER_CALLBACK_QUERY_TEXT_LENGTH`
New in version 13.2.

**answer**
```
(text=None, show_alert=False, url=None, cache_time=None, timeout=None, api_kwargs=None)
```
Shortcut for:
```
bot.answer_callback_query(update.callback_query.id, *args, **kwargs)
```
For the documentation of the arguments, please see `telegram.Bot.answer_callback_query()`.

Returns On success, True is returned.
Return type bool

**copy_message**
```
(chat_id, caption=None, parse_mode=None, caption_entities=None, disable_notification=None, reply_to_message_id=None, allow_sending_without_reply=None, reply_markup=None, timeout=None, api_kwargs=None, protect_content=None)
```
Shortcut for:
```
update.callback_query.message.copy({
    chat_id,
    from_chat_id=update.message.chat_id,
    message_id=update.message.message_id,
    *args,
    **kwargs})
```
For the documentation of the arguments, please see `telegram.Message.copy()`.

Returns On success, returns the MessageId of the sent message.
Return type `telegram.MessageId`
**classmethod de_json**(data, bot)

See `telegram.TelegramObject.de_json()`.

**delete_message**(timeout=None, api_kwargs=None)

Shortcut for:

```python
update.callback_query.message.delete(*args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Message.delete()`.

**Returns**

On success, True is returned.

**Return type** `bool`

**edit_message_caption**(caption=None, reply_markup=None, timeout=None, parse_mode=None, api_kwargs=None, caption_entities=None)

Shortcut for either:

```python
update.callback_query.message.edit_caption(caption, *args, **kwargs)
```

or:

```python
bot.edit_message_caption(caption=caption
                        inline_message_id=update.callback_query.inline_message_id,
                        *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.edit_message_caption()` and `telegram.Message.edit_caption()`.

**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_live_location**(latitude=None, longitude=None, location=None, reply_markup=None, timeout=None, horizontal_accuracy=None, heading=None, proximity_alert_radius=None)

Shortcut for either:

```python
update.callback_query.message.edit_live_location(*args, **kwargs)
```

or:

```python
bot.edit_message_live_location(
    inline_message_id=update.callback_query.inline_message_id,
    *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.edit_message_live_location()` and `telegram.Message.edit_live_location()`.

**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_media**(media=None, reply_markup=None, timeout=None, api_kwargs=None)

Shortcut for either:

```python
update.callback_query.message.edit_media(*args, **kwargs)
```

or:

```python
bot.edit_message_media(media=media
                        inline_message_id=update.callback_query.inline_message_id,
                        *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.edit_message_media()` and `telegram.Message.edit_media()`.

**Returns**

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

**Return type** `telegram.Message`
bot.edit_message_media(inline_message_id=update.callback_query.inline_message_id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.edit_message_media() and telegram.Message.edit_media().

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

**Return type** telegram.Message

eat_message_reply_markup(replyMarkup=None, timeout=None, api_kwargs=None)

Shortcut for either:

```python
update.callback_query.message.edit_reply_markup(
    reply_markup=reply_markup,
    *args, **kwargs
)
```  
or:

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

For the documentation of the arguments, please see telegram.Bot.edit_message_reply_markup() and telegram.Message.edit_reply_markup().

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

**Return type** telegram.Message

eat_message_text(text, parse_mode=None, disable_web_page_preview=None, reply_markup=None, timeout=None, api_kwargs=None, entities=None)

Shortcut for either:

```python
update.callback_query.message.edit_text(text, *args, **kwargs)
```  
or:

```python
bot.edit_message_text(text, inline_message_id=update.callback_query.inline_message_id,
                      *args, **kwargs)
```  

For the documentation of the arguments, please see telegram.Bot.edit_message_text() and telegram.Message.edit_text().

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

**Return type** telegram.Message

gate_game_high_scores(userid, timeout=None, api_kwargs=None)

Shortcut for either:

```python
update.callback_query.message.get_game_high_score(*args, **kwargs)
```  
or:

```python
update.callback_query.message.get_game_high_score(userid, timeout=None, api_kwargs=None)
```
bot.get_game_high_scores(inline_message_id=update.callback_query.inline_message_id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.get_game_high_scores() and telegram.Message.get_game_high_score().

**Returns** List[telegram.GameHighScore]

**pin_message** (disable_notification=None, timeout=None, api_kwars=None)

Shortcut for:

update.callback_query.message.pin(*args, **kwargs)

For the documentation of the arguments, please see telegram.Message.pin().

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

**Return type** bool

**set_game_score** (user_id, score, force=None, disable_edit_message=None, timeout=None, api_kwars=None)

Shortcut for either:

update.callback_query.message.set_game_score(*args, **kwargs)

or:

bot.set_game_score(inline_message_id=update.callback_query.inline_message_id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.set_game_score() and telegram.Message.set_game_score().

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

**Return type** telegram.Message

**stop_message_live_location** (reply_markup=None, timeout=None, api_kwars=None)

Shortcut for either:

update.callback_query.message.stop_live_location(*args, **kwargs)

or:

bot.stop_message_live_location(inline_message_id=update.callback_query.inline_message_id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.stop_message_live_location() and telegram.Message.stop_live_location().

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

**Return type** telegram.Message

**unpin_message** (timeout=None, api_kwars=None)

Shortcut for:
update.callback_query.message.unpin(*args, **kwargs)

For the documentation of the arguments, please see `telegram.Message.unpin()`.

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

**Return type**  `bool`

### 3.2.14 `telegram.Chat`

```python
class telegram.Chat(id, type, title=None, username=None, first_name=None, last_name=None,
bot=None, photo=None, description=None, invite_link=None, pinned_message=None,
permissions=None, sticker_set_name=None, can_set_sticker_set=None, slow_mode_delay=None,
bio=None, linked_chat_id=None, location=None, message_auto_delete_time=None,
has_private_forwards=None, has_protected_content=None, **kwargs)
```

**Bases:** `telegram.base.TelegramObject`

This object represents a chat. Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `id` is equal.

**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.
- **bio** (`str`, optional) – Bio of the other party in a private chat. Returned only in `telegram.Bot.get_chat()`.
- **has_private_forwards** (`bool`, optional) – `True`, if privacy settings of the other party in the private chat allows to use `tg://user?id=<user_id>` links only in chats with the user. Returned only in `telegram.Bot.get_chat()`.
  
  New in version 13.9.
- **description** (`str`, optional) – Description, for groups, supergroups and channel chats. Returned only in `telegram.Bot.get_chat()`.
- **invite_link** (`str`, optional) – Primary invite link, for groups, supergroups and channel. Returned only in `telegram.Bot.get_chat()`.
- **pinned_message** (`telegram.Message`, optional) – The most recent pinned message (by sending date). Returned only in `telegram.Bot.get_chat()`.
- **permissions** (`telegram.ChatPermissions`) – Optional. Default chat member permissions, for groups and supergroups. Returned only in `telegram.Bot.get_chat()`.
- slow_mode_delay (int, optional) – For supergroups, the minimum allowed delay between consecutive messages sent by each unprivileged user. Returned only in `telegram.Bot.get_chat()`.

- message_auto_delete_time (int, optional) – The time after which all messages sent to the chat will be automatically deleted; in seconds. Returned only in `telegram.Bot.get_chat()`.
  
  New in version 13.4.

- has_protected_content (bool, optional) – True, if messages from the chat can’t be forwarded to other chats. Returned only in `telegram.Bot.get_chat()`.
  
  New in version 13.9.

- 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 `telegram.Bot.get_chat()`.

- can_set_sticker_set (bool, optional) – True, if the bot can change group the sticker set. Returned only in `telegram.Bot.get_chat()`.

- linked_chat_id (int, optional) – Unique identifier for the linked chat, i.e. the discussion group identifier for a channel and vice versa; for supergroups and channel chats. Returned only in `telegram.Bot.get_chat()`.

- location (telegram.ChatLocation, optional) – For supergroups, the location to which the supergroup is connected. Returned only in `telegram.Bot.get_chat()`.

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

  id
  
  Unique identifier for this chat.

  Type int

  type
  
  Type of chat.

  Type str

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

  Type str

  username
  
  Optional. Username.

  Type str

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

  Type str

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

  Type str

  photo
  
  Optional. Chat photo.

  Type `telegram.ChatPhoto`

  bio
  

3.2. telegram package
Type `str`

**has_private_forwards**
Optional. True, if privacy settings of the other party in the private chat allows to use `tg://user?id=<user_id>` links only in chats with the user.

New in version 13.9.

Type `bool`

**description**
Optional. Description, for groups, supergroups and channel chats.

Type `str`

**invite_link**
Optional. Primary invite link, for groups, supergroups and channel. Returned only in `telegram.Bot.get_chat()`.

Type `str`

**pinned_message**
Optional. The most recent pinned message (by sending date). Returned only in `telegram.Bot.get_chat()`.

Type `telegram.Message`

**permissions**
Optional. Default chat member permissions, for groups and supergroups. Returned only in `telegram.Bot.get_chat()`.

Type `telegram.ChatPermissions`

**slow_mode_delay**
Optional. For supergroups, the minimum allowed delay between consecutive messages sent by each unprivileged user. Returned only in `telegram.Bot.get_chat()`.

Type `int`

**message_auto_delete_time**
Optional. The time after which all messages sent to the chat will be automatically deleted; in seconds. Returned only in `telegram.Bot.get_chat()`.

New in version 13.4.

Type `int`

**has_protected_content**
Optional. True, if messages from the chat can’t be forwarded to other chats.

New in version 13.9.

Type `bool`

**sticker_set_name**
Optional. For supergroups, name of Group sticker set.

Type `str`

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

Type `bool`

**linked_chat_id**
Optional. Unique identifier for the linked chat, i.e. the discussion group identifier for a channel and vice versa; for supergroups and channel chats. Returned only in `telegram.Bot.get_chat()`.

Type `int`
location
Optional. For supergroups, the location to which the supergroup is connected. Returned only in 
telegram.Bot.get_chat().

Type telegram.ChatLocation

CHANNEL: ClassVar[str] = 'channel'
telegram.constants.CHAT_CHANNEL

GROUP: ClassVar[str] = 'group'
telegram.constants.CHAT_GROUP

PRIVATE: ClassVar[str] = 'private'
telegram.constants.CHAT_PRIVATE

SENDER: ClassVar[str] = 'sender'
telegram.constants.CHAT_SENDER

New in version 13.5.

SUPERGROUP: ClassVar[str] = 'supergroup'
telegram.constants.CHAT_SUPERGROUP

approve_join_request (user_id, timeout=None, api_kwargs=None)
Shortcut for:

bot.approve_chat_join_request(chat_id=update.effective_chat.id, *args,
**kwargs)

For the documentation of the arguments, please see telegram.Bot.
approve_chat_join_request().

New in version 13.8.

Returns On success, True is returned.

Return type bool

ban_chat (chat_id, timeout=None, api_kwargs=None)
Shortcut for:

bot.ban_chat_sender_chat(sender_chat_id=update.effective_chat.id, *args,
**kwargs)

For the documentation of the arguments, please see telegram.Bot.
ban_chat_sender_chat().

New in version 13.9.

Returns On success, True is returned.

Return type bool

ban_member (user_id, timeout=None, until_date=None, api_kwargs=None, re-
voke_messages=None)
Shortcut for:

bot.ban_chat_member(update.effective_chat.id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.
ban_chat_member().

Returns On success, True is returned.

Return type bool

ban_sender_chat (sender_chat_id, timeout=None, api_kwargs=None)
Shortcut for:
bot.ban_chat_sender_chat(chat_id=update.effective_chat.id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.

New in version 13.9.

Returns On success, True is returned.

Return type bool
copy_message(chat_id, message_id, caption=None, parse_mode=None, caption_entities=None, disable_notification=None, reply_to_message_id=None, allow_sending_without_reply=None, reply_markup=None, timeout=None, api_kwargs=None, protect_content=None)

Shortcut for:

bot.copy_message(from_chat_id=update.effective_chat.id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.copy_message().

Returns On success, instance representing the message posted.

Return type telegram.Message
create_invite_link(expire_date=None, member_limit=None, timeout=None, api_kwargs=None, name=None, creates_join_request=None)

Shortcut for:

bot.create_chat_invite_link(chat_id=update.effective_chat.id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.

create_chat_invite_link().

New in version 13.4.

Changed in version 13.8: Edited signature according to the changes of telegram.Bot.

create_chat_invite_link().

Returns telegram.ChatInviteLink
classmethod de_json(data, bot)
See telegram.TelegramObject.de_json().
decline_join_request(user_id, timeout=None, api_kwargs=None)

Shortcut for:

bot.decline_chat_join_request(chat_id=update.effective_chat.id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.

decline_chat_join_request().

New in version 13.8.

Returns On success, True is returned.

Return type bool
edit_invite_link(invite_link, expire_date=None, member_limit=None, timeout=None, api_kwargs=None, name=None, creates_join_request=None)

Shortcut for:

bot.edit_chat_invite_link(chat_id=update.effective_chat.id, *args, **kwargs)
For the documentation of the arguments, please see `telegram.Bot.edit_chat_invite_link()`.

New in version 13.4.

Changed in version 13.8: Edited signature according to the changes of `telegram.Bot.edit_chatInviteLink()`.

**Returns** `telegram.ChatInviteLink`

`export_invite_link` *(timeout=None, api_kwargs=None)*

Shortcut for:

```python
bot.export_chat_invite_link(chat_id=update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.export_chat_invite_link()`.

New in version 13.4.

**Returns** New invite link on success.

**Return type** `str`

**property full_name**

Convenience property. If `first_name` is not `None` gives, `first_name` followed by (if available) `last_name`.

**Note:** `full_name` will always be `None`, if the chat is a (super)group or channel.

New in version 13.2.

**Type** `str`

`get_administrators` *(timeout=None, api_kwargs=None)*

Shortcut for:

```python
bot.get_chat_administrators(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.get_chat_administrators()`.

**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` *(user_id, timeout=None, api_kwargs=None)*

Shortcut for:

```python
bot.get_chat_member(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.get_chat_member()`.

**Returns** `telegram.ChatMember`

`get_member_count` *(timeout=None, api_kwargs=None)*

Shortcut for:

```python
bot.get_chat_member_count(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.get_chat_member_count()`.

### 3.2. `telegram` package
Returns int

get_members_count (timeout=None, api_kwargs=None)
  Deprecated, use get_member_count() instead.
  Deprecated since version 13.7.

kick_member (user_id, timeout=None, until_date=None, api_kwargs=None, revoke_messages=None)
  Deprecated, use ban_member() instead.
  Deprecated since version 13.7.

leave (timeout=None, api_kwargs=None)
  Shortcut for:
  
  ```
  bot.leave_chat(update.effective_chat.id, *args, **kwargs)
  ```

  For the documentation of the arguments, please see `telegram.Bot.leave_chat()`.

  Returns On success, True is returned.

  Return type bool

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

  Type str

pin_message (message_id, disable_notification=None, timeout=None, api_kwargs=None)
  Shortcut for:
  
  ```
  bot.pin_chat_message(chat_id=update.effective_chat.id, *args, **kwargs)
  ```

  For the documentation of the arguments, please see `telegram.Bot.pin_chat_message()`.

  Returns On success, True is returned.

  Return type bool

promote_member (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, api_kwargs=None, is_anonymous=None, can_manage_chat=None, can_manage_voice_chats=None)
  Shortcut for:
  
  ```
  bot.promote_chat_member(update.effective_chat.id, *args, **kwargs)
  ```

  For the documentation of the arguments, please see `telegram.Bot.promote_chat_member()`.

  New in version 13.2.

  Returns On success, True is returned.

  Return type bool

restrict_member (user_id, permissions, until_date=None, timeout=None, api_kwargs=None)
  Shortcut for:
  
  ```
  bot.restrict_chat_member(update.effective_chat.id, *args, **kwargs)
  ```

  For the documentation of the arguments, please see `telegram.Bot.restrict_chat_member()`.
New in version 13.2.

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

**Return type** bool

### revoke_invite_link

`revoke_invite_link(invite_link, timeout=None, api_kwargs=None)`

Shortcut for:

```python
bot.revoke_chat_invite_link(chat_id=update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.revoke_chat_invite_link()`.

New in version 13.4.

**Returns** `telegram.ChatInviteLink`

### send_action

`send_action(action, timeout=None, api_kwargs=None)`

Alias for `send_chat_action`

### send_animation

`send_animation(animation, duration=None, width=None, height=None, thumb=None, caption=None, parse_mode=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, api_kwargs=None, allow_sending_without_reply=None, caption_entities=None, filename=None, protect_content=None)`

Shortcut for:

```python
bot.send_animation(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_animation()`.

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

**Return type** `telegram.Message`

### send_audio

`send_audio(audio, duration=None, performer=None, title=None, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, api_kwargs=None, allow_sending_without_reply=None, caption_entities=None, filename=None, protect_content=None)`

Shortcut for:

```python
bot.send_audio(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_audio()`.

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

**Return type** `telegram.Message`

### send_chat_action

`send_chat_action(action, timeout=None, api_kwargs=None)`

Shortcut for:

```python
bot.send_chat_action(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_chat_action()`.

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

**Return type** bool

### send_contact

`send_contact(phone_number=None, first_name=None, last_name=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, contact=None, vcard=None, api_kwargs=None, allow_sending_without_reply=None, protect_content=None)`

Shortcut for:

```python
bot.send_contact(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_contact()`.
For the documentation of the arguments, please see `telegram.Bot.send_contact()`.

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

**Return type** `telegram.Message`

`sending` (from chat id, message id, caption=None, parse_mode=None, caption_entities=None, disable_notification=None, reply_to_message_id=None, allow_sending_without_reply=None, reply_markup=None, timeout=None, api_kwargs=None, protect_content=None)

Shortcut for:

```python
bot.copy_message(chat_id=update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.copy_message()`.

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

**Return type** `telegram.Message`

`send_dice` (disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, emoji=None, api_kwargs=None, allow_sending_without_reply=None, protect_content=None)

Shortcut for:

```python
bot.send_dice(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_dice()`.

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

**Return type** `telegram.Message`

`send_document` (document, filename=None, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, thumb=None, api_kwargs=None, disable_content_type_detection=None, allow_sending_without_reply=None, caption_entities=None, protect_content=None)

Shortcut for:

```python
bot.send_document(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_document()`.

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

**Return type** `telegram.Message`

`send_game` (game_short_name, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, api_kwargs=None, allow_sending_without_reply=None, protect_content=None)

Shortcut for:

```python
bot.send_game(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_game()`.

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

**Return type** `telegram.Message`
send_invoice(title, description, payload, provider_token, currency, prices, start_parameter=None, 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=None, reply_to_message_id=None, reply_markup=None, provider_data=None, send_phone_number_to_provider=None, send_email_to_provider=None, timeout=None, api_kwargs=None, allow_sending_without_reply=None, max_tip_amount=None, suggested_tip_amounts=None, protect_content=None)

Shortcut for:

```python
bot.send_invoice(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see telegram.Bot.send_invoice().

**Warning:** As of API 5.2 `start_parameter` is an optional argument and therefore the order of the arguments had to be changed. Use keyword arguments to make sure that the arguments are passed correctly.

Changed in version 13.5: As of Bot API 5.2, the parameter `start_parameter` is optional.

Returns On success, instance representing the message posted.

Return type telegram.Message

send_location(latitude=None, longitude=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, location=None, live_period=None, api_kwargs=None, horizontal_accuracy=None, heading=None, proximity_alert_radius=None, allow_sending_without_reply=None, protect_content=None)

Shortcut for:

```python
bot.send_location(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see telegram.Bot.send_location().

Returns On success, instance representing the message posted.

Return type telegram.Message

send_media_group(media, disable_notification=None, reply_to_message_id=None, timeout=20, api_kwargs=None, allow_sending_without_reply=None, protect_content=None)

Shortcut for:

```python
bot.send_media_group(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see telegram.Bot.send_media_group().

Returns On success, instance representing the message posted.

Return type List[telegram.Message]

send_message(text, parse_mode=None, disable_web_page_preview=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, api_kwargs=None, allow_sending_without_reply=None, entities=None, protect_content=None)

Shortcut for:

```python
bot.send_message(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see telegram.Bot.send_message().

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

```python
python-telegram-bot Documentation, Release 13.11

Return type  telegram.Message

send_photo(photo, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, api_kwargs=None, allow_sending_without_reply=None, caption_entities=None, filename=None, protect_content=None)
```

Shortcut for:
```
bot.send_photo(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_photo()`.

**Returns**

On success, instance representing the message posted.

**send_poll**

```python
python-telegram-bot Documentation, Release 13.11

Return type  telegram.Message

send_poll(question, options, is_anonymous=True, type='regular', allows_multiple_answers=False, correct_option_id=None, is_closed=None, disable_notification=None, timeout=None, explanation=None, explanation_parse_mode=None, open_period=None, close_date=None, api_kwargs=None, allow_sending_without_reply=None, explanation_entities=None, protect_content=None)
```

Shortcut for:
```
bot.send_poll(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_poll()`.

**Returns**

On success, instance representing the message posted.

**send_sticker**

```python
python-telegram-bot Documentation, Release 13.11

Return type  telegram.Message

send_sticker(sticker, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, api_kwargs=None, allow_sending_without_reply=None, protect_content=None)
```

Shortcut for:
```
bot.send_sticker(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_sticker()`.

**Returns**

On success, instance representing the message posted.

**send_venue**

```python
python-telegram-bot Documentation, Release 13.11

Return type  telegram.Message

send_venue(latitude=None, longitude=None, title=None, address=None, foursquare_id=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, venue=None, foursquare_type=None, api_kwargs=None, google_place_id=None, google_place_type=None, allow_sending_without_reply=None, protect_content=None)
```

Shortcut for:
```
bot.send_venue(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_venue()`.

**Returns**

On success, instance representing the message posted.

**send_video**

```python
python-telegram-bot Documentation, Release 13.11

Return type  telegram.Message

send_video(video, duration=None, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, width=None, height=None, parse_mode=None, supports_streaming=None, thumb=None, api_kwargs=None, allow_sending_without_reply=None, caption_entities=None, filename=None, protect_content=None)
```

Shortcut for:
bot.send_video(update.effective_chat.id, *args, **kwargs)

For the documentation of the arguments, please see \texttt{telegram.Bot.send\_video()}.  

\textbf{Returns} On success, instance representing the message posted.  

\textbf{Return type} \texttt{telegram.Message}

\texttt{send\_video\_note}(\texttt{video\_note}, duration=\texttt{None}, length=\texttt{None}, disable\_notification=\texttt{None}, reply\_to\_message\_id=\texttt{None}, reply\_markup=\texttt{None}, timeout=\texttt{20}, thumb=\texttt{None}, api\_kwargs=\texttt{None}, allow\_sending\_without\_reply=\texttt{None}, filename=\texttt{None}, protect\_content=\texttt{None})

Shortcut for:

bot.send_video_note(update.effective_chat.id, *args, **kwargs)

For the documentation of the arguments, please see \texttt{telegram.Bot.send\_video\_note()}.  

\textbf{Returns} On success, instance representing the message posted.  

\textbf{Return type} \texttt{telegram.Message}

\texttt{send\_voice}(\texttt{voice}, duration=\texttt{None}, caption=\texttt{None}, disable\_notification=\texttt{None}, reply\_to\_message\_id=\texttt{None}, reply\_markup=\texttt{None}, timeout=\texttt{20}, parse\_mode=\texttt{None}, api\_kwargs=\texttt{None}, allow\_sending\_without\_reply=\texttt{None}, caption\_entities=\texttt{None}, filename=\texttt{None}, protect\_content=\texttt{None})

Shortcut for:

bot.send_voice(update.effective_chat.id, *args, **kwargs)

For the documentation of the arguments, please see \texttt{telegram.Bot.send\_voice()}.  

\textbf{Returns} On success, instance representing the message posted.  

\textbf{Return type} \texttt{telegram.Message}

\texttt{set\_administrator\_custom\_title}(\texttt{user\_id}, custom\_title, timeout=\texttt{None}, api\_kwargs=\texttt{None})

Shortcut for:

bot.set_chat_administrator_custom_title(update.effective_chat.id, *args, **kwargs)

For the documentation of the arguments, please see \texttt{telegram.Bot.set\_chat\_administrator\_custom\_title()}.  

\textbf{Returns} On success, \texttt{True} is returned.  

\textbf{Return type} \texttt{bool}

\texttt{set\_permissions}(\texttt{permissions}, timeout=\texttt{None}, api\_kwargs=\texttt{None})

Shortcut for:

bot.set_chat_permissions(update.effective_chat.id, *args, **kwargs)

For the documentation of the arguments, please see \texttt{telegram.Bot.set\_chat\_permissions()}.  

\textbf{Returns} On success, \texttt{True} is returned.  

\textbf{Return type} \texttt{bool}

\texttt{unban\_chat}(\texttt{chat\_id}, timeout=\texttt{None}, api\_kwargs=\texttt{None})

Shortcut for:

bot.unban_chat_sender_chat(sender_chat_id=update.effective_chat.id, *args, **kwargs)
For the documentation of the arguments, please see `telegram.Bot.unban_chat_sender_chat()`.

New in version 13.9.

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

**Return type** bool

### unban_member

`unban_member(user_id, timeout=None, api_kwargs=None, only_if_banned=None)`

Shortcut for:

```python
bot.unban_chat_member(update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.unban_chat_member()`.

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

**Return type** bool

### unban_sender_chat

`unban_sender_chat(sender_chat_id, timeout=None, api_kwargs=None)`

Shortcut for:

```python
bot.unban_chat_sender_chat(chat_id=update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.unban_chat_sender_chat()`.

New in version 13.9.

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

**Return type** bool

### unpin_all_messages

`unpin_all_messages(timeout=None, api_kwargs=None)`

Shortcut for:

```python
bot.unpin_all_chat_messages(chat_id=update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.unpin_all_chat_messages()`.

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

**Return type** bool

### unpin_message

`unpin_message(timeout=None, api_kwargs=None, message_id=None)`

Shortcut for:

```python
bot.unpin_chat_message(chat_id=update.effective_chat.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.unpin_chat_message()`.

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

**Return type** bool
### 3.2.15 telegram.ChatAction

**class** `telegram.ChatAction`

**Bases:** `object`

Helper class to provide constants for different chat actions.

- **CHOOSE_STICKER**: `ClassVar[str] = 'choose_sticker'`
  
  
  `telegram.constants.CHOOSE_STICKER`

  New in version 13.8.

- **FIND_LOCATION**: `ClassVar[str] = 'find_location'`
  
  
  `telegram.constants.CHATACTION_FIND_LOCATION`

- **RECORD_AUDIO**: `ClassVar[str] = 'record_audio'`
  
  
  `telegram.constants.CHATACTION_RECORD_AUDIO`

  Deprecated since version 13.5: Deprecated by Telegram. Use `RECORD_VOICE` instead.

- **RECORD_VIDEO**: `ClassVar[str] = 'record_video'`
  
  
  `telegram.constants.CHATACTION_RECORD_VIDEO`

- **RECORD_VIDEO_NOTE**: `ClassVar[str] = 'record_video_note'`
  
  
  `telegram.constants.CHATACTION_RECORD_VIDEO_NOTE`

- **RECORD_VOICE**: `ClassVar[str] = 'record_voice'`
  
  
  `telegram.constants.CHATACTION_RECORD_VOICE`

  New in version 13.5.

- **TYPING**: `ClassVar[str] = 'typing'`
  
  
  `telegram.constants.CHATACTION_TYPING`

- **UPLOAD_AUDIO**: `ClassVar[str] = 'upload_audio'`
  
  
  `telegram.constants.CHATACTION_UPLOAD_AUDIO`

  Deprecated since version 13.5: Deprecated by Telegram. Use `UPLOAD_VOICE` instead.

- **UPLOAD_DOCUMENT**: `ClassVar[str] = 'upload_document'`
  
  
  `telegram.constants.CHATACTION_UPLOAD_DOCUMENT`

- **UPLOAD_PHOTO**: `ClassVar[str] = 'upload_photo'`
  
  
  `telegram.constants.CHATACTION_UPLOAD_PHOTO`

- **UPLOAD_VIDEO**: `ClassVar[str] = 'upload_video'`
  
  
  `telegram.constants.CHATACTION UPLOAD_VIDEO`

- **UPLOAD_VIDEO_NOTE**: `ClassVar[str] = 'upload_video_note'`
  
  
  `telegram.constants.CHATACTION UPLOAD_VIDEO_NOTE`

- **UPLOAD_VOICE**: `ClassVar[str] = 'upload_voice'`
  
  
  `telegram.constants.CHATACTION UPLOAD_VOICE`

  New in version 13.5.

### 3.2.16 telegram.ChatInviteLink

**class** `telegram.ChatInviteLink`

**Bases:** `telegram.base.TelegramObject`

This object represents an invite link for a chat.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `invite_link`, `creator`, `is_primary` and `is_revoked` are equal.
New in version 13.4.

Parameters

- **invite_link**(str) – The invite link.
- **creator**(telegram.User) – Creator of the link.
- **is_primary**(bool) – True, if the link is primary.
- **is_revoked**(bool) – True, if the link is revoked.
- **expire_date**(datetime.datetime, optional) – Date when the link will expire or has been expired.
- **member_limit**(int, optional) – Maximum number of users that can be members of the chat simultaneously after joining the chat via this invite link; 1-99999.
- **name**(str, optional) – Invite link name.
  New in version 13.8.
- **creates_join_request**(bool, optional) – True, if users joining the chat via the link need to be approved by chat administrators.
  New in version 13.8.
- **pending_join_request_count**(int, optional) – Number of pending join requests created using this link.
  New in version 13.8.

**invite_link**

The invite link. If the link was created by another chat administrator, then the second part of the link will be replaced with ‘...’.

Type: str

**creator**

Creator of the link.

Type: telegram.User

**is_primary**

True, if the link is primary.

Type: bool

**is_revoked**

True, if the link is revoked.

Type: bool

**expire_date**

Optional. Date when the link will expire or has been expired.

Type: datetime.datetime

**member_limit**

Optional. Maximum number of users that can be members of the chat simultaneously after joining the chat via this invite link; 1-99999.

Type: int

**name**

Optional. Invite link name.

New in version 13.8.

Type: str
creates_join_request
   Optional. True, if users joining the chat via the link need to be approved by chat administrators.
   New in version 13.8.
   Type bool

pending_join_request_count
   Optional. Number of pending join requests created using this link.
   New in version 13.8.
   Type int

classmethod de_json(data, bot)
   See telegram.TelegramObject.de_json().

to_dict()
   See telegram.TelegramObject.to_dict().

3.2.17 telegram.ChatJoinRequest

class telegram.ChatJoinRequest(chat, from_user, date, bio=None, invite_link=None, bot=None, **_kwargs)
Bases: telegram.base.TelegramObject

This object represents a join request sent to a chat.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their chat, from_user and date are equal.

Note: Since Bot API 5.5, bots are allowed to contact users who sent a join request to a chat where the bot is an administrator with the can_invite_users administrator right – even if the user never interacted with the bot before.

New in version 13.8.

Parameters

- **chat** (telegram.Chat) – Chat to which the request was sent.
- **from_user** (telegram.User) – User that sent the join request.
- **date** (datetime.datetime) – Date the request was sent.
- **bio** (str, optional) – Bio of the user.
- **invite_link** (telegram.ChatInviteLink, optional) – Chat invite link that was used by the user to send the join request.
- **bot** (telegram.Bot, optional) – The Bot to use for instance methods.

chat
   Chat to which the request was sent.
   Type telegram.Chat

from_user
   User that sent the join request.
   Type telegram.User

date
   Date the request was sent.
   Type datetime.datetime
bio
Optional. Bio of the user.

Type str

invite_link
Optional. Chat invite link that was used by the user to send the join request.

Type telegram.ChatInviteLink

approve (timeout=None, api_kwargs=None)
Shortcut for:

bot.approve_chat_join_request(chat_id=update.effective_chat.id,
user_id=update.effective_user.id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.
approve_chat_join_request().

Returns On success, True is returned.

Return type bool

classmethod de_json (data, bot)
See telegram.TelegramObject.de_json().

decline (timeout=None, api_kwargs=None)
Shortcut for:

bot.decline_chat_join_request(chat_id=update.effective_chat.id,
user_id=update.effective_user.id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.
decline_chat_join_request().

Returns On success, True is returned.

Return type bool
to_dict ()
See telegram.TelegramObject.to_dict().

3.2.18 telegram.ChatLocation

class telegram.ChatLocation (location, address, **kwargs)
Bases: telegram.base.TelegramObject

This object represents a location to which a chat is connected.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their location is equal.

Parameters

- location (telegram.Location) – The location to which the supergroup is connected. Can’t be a live location.
- address (str) – Location address; 1-64 characters, as defined by the chat owner
- **kwargs (dict) – Arbitrary keyword arguments.

location
The location to which the supergroup is connected.

Type telegram.Location

address
Location address, as defined by the chat owner
Type `str`

`classmethod de_json(data, bot)`

See `telegram.TelegramObject.de_json()`.

### 3.2.19 `telegram.ChatMember`

```python
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_polls=None, can_send_other_messages=None, can_add_web_page_previews=None, is_member=None, custom_title=None, is_anonymous=None, can_manage_chat=None, can_manage_voice_chats=None, **kwargs)
```

Bases: `telegram.base.TelegramObject`

Base class for Telegram ChatMember Objects. Currently, the following 6 types of chat members are supported:

- `telegram.ChatMemberOwner`
- `telegram.ChatMemberAdministrator`
- `telegram.ChatMemberMember`
- `telegram.ChatMemberRestricted`
- `telegram.ChatMemberLeft`
- `telegram.ChatMemberBanned`

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `user` and `status` are equal.

**Note:** As of Bot API 5.3, `ChatMember` is nothing but the base class for the subclasses listed above and is no longer returned directly by `get_chat()`. Therefore, most of the arguments and attributes were deprecated and you should no longer use `ChatMember` directly.

**Parameters**

- `user (telegram.User)` – Information about the user.

- `status (str)` – The member’s status in the chat. Can be ADMINISTRATOR, CREATOR, KICKED, LEFT, MEMBER or RESTRICTED.

- `custom_title (str, optional)` – Owner and administrators only. Custom title for this user.

  Deprecated since version 13.7.

- `is_anonymous (bool, optional)` – Owner and administrators only. True, if the user’s presence in the chat is hidden.

  Deprecated since version 13.7.

- `until_date (datetime.datetime, optional)` – Restricted and kicked only. Date when restrictions will be lifted for this user.

  Deprecated since version 13.7.
• `can_be_edited` (bool, optional) – Administrators only. `True`, if the bot is allowed to edit administrator privileges of that user.
  Deprecated since version 13.7.

• `can_manage_chat` (bool, optional) – Administrators only. `True`, if the administrator can access the chat event log, chat statistics, message statistics in channels, see channel members, see anonymous administrators in supergroups and ignore slow mode. Implied by any other administrator privilege.
  New in version 13.4.
  Deprecated since version 13.7.

• `can_manage_voice_chats` (bool, optional) – Administrators only. `True`, if the administrator can manage voice chats.
  New in version 13.4.
  Deprecated since version 13.7.

• `can_change_info` (bool, optional) – Administrators and restricted only. `True`, if the user can change the chat title, photo and other settings.
  Deprecated since version 13.7.

• `can_post_messages` (bool, optional) – Administrators only. `True`, if the administrator can post in the channel, channels only.
  Deprecated since version 13.7.

• `can_edit_messages` (bool, optional) – Administrators only. `True`, if the administrator can edit messages of other users and can pin messages; channels only.
  Deprecated since version 13.7.

• `can_delete_messages` (bool, optional) – Administrators only. `True`, if the administrator can delete messages of other users.
  Deprecated since version 13.7.

• `can_invite_users` (bool, optional) – Administrators and restricted only. `True`, if the user can invite new users to the chat.
  Deprecated since version 13.7.

• `can_restrict_members` (bool, optional) – Administrators only. `True`, if the administrator can restrict, ban or unban chat members.
  Deprecated since version 13.7.

• `can_pin_messages` (bool, optional) – Administrators and restricted only. `True`, if the user can pin messages, groups and supergroups only.
  Deprecated since version 13.7.

• `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).
  Deprecated since version 13.7.

• `is_member` (bool, optional) – Restricted only. `True`, if the user is a member of the chat at the moment of the request.
  Deprecated since version 13.7.

• `can_send_messages` (bool, optional) – Restricted only. `True`, if the user can send text messages, contacts, locations and venues.
  Deprecated since version 13.7.
• **can_send_media_messages** *(bool, optional)* – Restricted only. True, if the user can send audios, documents, photos, videos, video notes and voice notes.
  
  Deprecated since version 13.7.

• **can_send_polls** *(bool, optional)* – Restricted only. True, if the user is allowed to send polls.
  
  Deprecated since version 13.7.

• **can_send_other_messages** *(bool, optional)* – Restricted only. True, if the user can send animations, games, stickers and use inline bots.
  
  Deprecated since version 13.7.

• **can_add_web_page_previews** *(bool, optional)* – Restricted only. True, if user may add web page previews to his messages.
  
  Deprecated since version 13.7.

**user**

Information about the user.

*Type* `telegram.User`

**status**

The member’s status in the chat.

*Type* `str`

**custom_title**

Optional. Custom title for owner and administrators.

Deprecated since version 13.7.

*Type* `str`

**is_anonymous**

Optional. True, if the user’s presence in the chat is hidden.

Deprecated since version 13.7.

*Type* `bool`

**until_date**

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

Deprecated since version 13.7.

*Type* `datetime.datetime`

**can_be_edited**

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

Deprecated since version 13.7.

*Type* `bool`

**can_manage_chat**

Optional. If the administrator can access the chat event log, chat statistics, message statistics in channels, see channel members, see anonymous administrators in supergroups and ignore slow mode.

New in version 13.4.

Deprecated since version 13.7.

*Type* `bool`

**can_manage_voice_chats**

Optional. If the administrator can manage voice chats.

New in version 13.4.
can_change_info
  Optional. If the user can change the chat title, photo and other settings.
  Deprecated since version 13.7.
  Type bool

can_post_messages
  Optional. If the administrator can post in the channel.
  Deprecated since version 13.7.
  Type bool

can_edit_messages
  Optional. If the administrator can edit messages of other users.
  Deprecated since version 13.7.
  Type bool

can_delete_messages
  Optional. If the administrator can delete messages of other users.
  Deprecated since version 13.7.
  Type bool

can_invite_users
  Optional. If the user can invite new users to the chat.
  Deprecated since version 13.7.
  Type bool

can_restrict_members
  Optional. If the administrator can restrict, ban or unban chat members.
  Deprecated since version 13.7.
  Type bool

can_pin_messages
  Optional. If the user can pin messages.
  Deprecated since version 13.7.
  Type bool

can_promote_members
  Optional. If the administrator can add new administrators.
  Deprecated since version 13.7.
  Type bool

is_member
  Optional. Restricted only. True, if the user is a member of the chat at the moment of the request.
  Deprecated since version 13.7.
  Type bool

can_send_messages
  Optional. If the user can send text messages, contacts, locations and venues.
  Deprecated since version 13.7.
  Type bool
can_send_media_messages
  Optional. If the user can send media messages, implies can_send_messages.
  Deprecated since version 13.7.
  Type bool

can_send_polls
  Optional. True, if the user is allowed to send polls.
  Deprecated since version 13.7.
  Type bool

can_send_other_messages
  Optional. If the user can send animations, games, stickers and use inline bots, implies can_send_media_messages.
  Deprecated since version 13.7.
  Type bool

can_add_web_page_previews
  Optional. If user may add web page previews to his messages, implies can_send_media_messages
  Deprecated since version 13.7.
  Type bool

ADMINISTRATOR: ClassVar[str] = 'administrator'
  telegram.constants.CHATMEMBER_ADMINISTRATOR

CREATOR: ClassVar[str] = 'creator'
  telegram.constants.CHATMEMBER_CREATOR

KICKED: ClassVar[str] = 'kicked'
  telegram.constants.CHATMEMBER_KICKED

LEFT: ClassVar[str] = 'left'
  telegram.constants.CHATMEMBER_LEFT

MEMBER: ClassVar[str] = 'member'
  telegram.constants.CHATMEMBER_MEMBER

RESTRICTED: ClassVar[str] = 'restricted'
  telegram.constants.CHATMEMBER_RESTRICTED

classmethod de_json(data, bot)
  See telegram.TelegramObject.de_json().

to_dict()
  See telegram.TelegramObject.to_dict().

3.2.20 telegram.ChatMemberOwner

class telegram.ChatMemberOwner(user, custom_title=None, is_anonymous=None, **kwargs)
  Bases: telegram.chatmember.ChatMember

  Represents a chat member that owns the chat and has all administrator privileges.
  New in version 13.7.

  Parameters
  - user (telegram.User) – Information about the user.
  - custom_title (str, optional) – Custom title for this user.
  - is_anonymous (bool, optional) – True, if the user’s presence in the chat is hidden.
status
The member’s status in the chat, always `telegram.ChatMember.CREATOR`.

    Type str

user
Information about the user.

    Type telegram.User

custom_title
Optional. Custom title for this user.

    Type str

is_anonymous
Optional. True, if the user’s presence in the chat is hidden.

    Type bool

### 3.2.21 `telegram.ChatMemberAdministrator`

**class** `telegram.ChatMemberAdministrator`

```
(user, can_be_edited=None, custom_title=None,
 is_anonymous=None, can_manage_chat=None,
 can_post_messages=None,
 can_edit_messages=None,
 can_delete_messages=None,
 can_manage_voice_chats=None,
 can_restrict_members=None,
 can_promote_members=None,
 can_change_info=None,
 can_invite_users=None,
 can_pin_messages=None, **_kwargs)
```

**Bases:** `telegram.chatmember.ChatMember`

Represents a chat member that has some additional privileges.

New in version 13.7.

**Parameters**

- **user** *(telegram.User)* – Information about the user.
- **can_be_edited** *(bool, optional)* – True, if the bot is allowed to edit administrator privileges of that user.
- **custom_title** *(str, optional)* – Custom title for this user.
- **is_anonymous** *(bool, optional)* – True, if the user’s presence in the chat is hidden.
- **can_manage_chat** *(bool, optional)* – True, if the administrator can access the chat event log, chat statistics, message statistics in channels, see channel members, see anonymous administrators in supergroups and ignore slow mode. Implied by any other administrator privilege.
- **can_post_messages** *(bool, optional)* – True, if the administrator can post in the channel, channels only.
- **can_edit_messages** *(bool, optional)* – True, if the administrator can edit messages of other users and can pin messages; channels only.
- **can_delete_messages** *(bool, optional)* – True, if the administrator can delete messages of other users.
- **can_manage_voice_chats** *(bool, optional)* – True, if the administrator can manage voice chats.
• **can_restrict_members** (bool, optional) – True, if the administrator can restrict, ban or unban chat members.

• **can_promote_members** (bool, optional) – 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_change_info** (bool, optional) – True, if the user can change the chat title, photo and other settings.

• **can_invite_users** (bool, optional) – True, if the user can invite new users to the chat.

• **can_pin_messages** (bool, optional) – True, if the user is allowed to pin messages; groups and supergroups only.

**status**
The member's status in the chat, always `telegram.ChatMember.ADMINISTRATOR`.

**user**
Information about the user.

**can_be_edited**
Optional. True, if the bot is allowed to edit administrator privileges of that user.

**custom_title**
Optional. Custom title for this user.

**is_anonymous**
Optional. True, if the user’s presence in the chat is hidden.

**can_manage_chat**
Optional. True, if the administrator can access the chat event log, chat statistics, message statistics in channels, see channel members, see anonymous administrators in supergroups and ignore slow mode. Implied by any other administrator privilege.

**can_post_messages**
Optional. True, if the administrator can post in the channel, channels only.

**can_edit_messages**
Optional. True, if the administrator can edit messages of other users and can pin messages; channels only.

**can_delete_messages**
Optional. True, if the administrator can delete messages of other users.

**can_manage_voice_chats**
Optional. True, if the administrator can manage voice chats.
can_restrict_members
   Optional. True, if the administrator can restrict, ban or unban chat members.
   Type bool

can_promote_members
   Optional. 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).
   Type bool

can_change_info
   Optional. True, if the user can change the chat title, photo and other settings.
   Type bool

can_invite_users
   Optional. True, if the user can invite new users to the chat.
   Type bool

can_pin_messages
   Optional. True, if the user is allowed to pin messages; groups and supergroups only.
   Type bool

3.222 telegram.ChatMemberMember

class telegram.ChatMemberMember(user, **kwargs)
   Bases: telegram.chatmember.ChatMember

   Represents a chat member that has no additional privileges or restrictions.
   New in version 13.7.

   Parameters
   user (telegram.User) – Information about the user.

   status
   The member's status in the chat, always telegram.ChatMember.MEMBER.
   Type str

   user
   Information about the user.
   Type telegram.User

3.223 telegram.ChatMemberRestricted

class telegram.ChatMemberRestricted(user, is_member=None, can_change_info=None, can_invite_users=None, can_pin_messages=None, can_send_messages=None, can_send_media_messages=None, can_send_polls=None, can_send_other_messages=None, can_add_web_page_previews=None, until_date=None, **kwargs)
   Bases: telegram.chatmember.ChatMember

   Represents a chat member that is under certain restrictions in the chat. Supergroups only.
   New in version 13.7.

   Parameters
   - user (telegram.User) – Information about the user.
• **is_member** *(bool, optional)* – True, if the user is a member of the chat at the moment of the request.

• **can_change_info** *(bool, optional)* – True, if the user can change the chat title, photo and other settings.

• **can_invite_users** *(bool, optional)* – True, if the user can invite new users to the chat.

• **can_pin_messages** *(bool, optional)* – True, if the user is allowed to pin messages; groups and supergroups only.

• **can_send_messages** *(bool, optional)* – True, if the user is allowed to send text messages, contacts, locations and venues.

• **can_send_media_messages** *(bool, optional)* – True, if the user is allowed to send audios, documents, photos, videos, video notes and voice notes.

• **can_send_polls** *(bool, optional)* – True, if the user is allowed to send polls.

• **can_send_other_messages** *(bool, optional)* – True, if the user is allowed to send animations, games, stickers and use inline bots.

• **can_add_web_page_previews** *(bool, optional)* – True, if the user is allowed to add web page previews to their messages.

• **until_date** *(datetime.datetime, optional)* – Date when restrictions will be lifted for this user.

**status**

The member’s status in the chat, always `telegram.ChatMember.RESTRICTED`.

Type str

**user**

Information about the user.

Type `telegram.User`

**is_member**

Optional. True, if the user is a member of the chat at the moment of the request.

Type bool

**can_change_info**

Optional. True, if the user can change the chat title, photo and other settings.

Type bool

**can_invite_users**

Optional. True, if the user can invite new users to the chat.

Type bool

**can_pin_messages**

Optional. True, if the user is allowed to pin messages; groups and supergroups only.

Type bool

**can_send_messages**

Optional. True, if the user is allowed to send text messages, contacts, locations and venues.

Type bool

**can_send_media_messages**

Optional. True, if the user is allowed to send audios, documents, photos, videos, video notes and voice notes.

Type bool

3.2. `telegram` package
**can_send_polls**
Optional. True, if the user is allowed to send polls.
Type bool

**can_send_other_messages**
Optional. True, if the user is allowed to send animations, games, stickers and use inline bots.
Type bool

**can_add_web_page_previews**
Optional. True, if the user is allowed to add web page previews to their messages.
Type bool

**until_date**
Optional. Date when restrictions will be lifted for this user.
Type datetime.datetime

### 3.2.24 telegram.ChatMemberLeft

**class** telegram.ChatMemberLeft
`user,**_kwargs)`
Bases: telegram.chatmember.ChatMember

Represents a chat member that isn’t currently a member of the chat, but may join it themselves.
New in version 13.7.

**Parameters**

- **`user` (telegram.User)** – Information about the user.

**status**
The member’s status in the chat, always `telegram.ChatMember.LEFT`.
Type str

**user**
Information about the user.
Type `telegram.User`

### 3.2.25 telegram.ChatMemberBanned

**class** telegram.ChatMemberBanned
`user,until_date=None,**_kwargs)`
Bases: telegram.chatmember.ChatMember

Represents a chat member that was banned in the chat and can’t return to the chat or view chat messages.
New in version 13.7.

**Parameters**

- **`user` (telegram.User)** – Information about the user.
- **`until_date` (datetime.datetime, optional)** – Date when restrictions will be lifted for this user.

**status**
The member’s status in the chat, always `telegram.ChatMember.KICKED`.
Type str

**user**
Information about the user.
Type `telegram.User`
until_date
    Optional. Date when restrictions will be lifted for this user.
    Type datetime.datetime

3.2.26 telegram.ChatMemberUpdated

class telegram.ChatMemberUpdated(chat, from_user, date, old_chat_member, new_chat_member, invite_link=None, **kwargs)
Bases: telegram.base.TelegramObject
This object represents changes in the status of a chat member.
Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if
their chat, from_user, date, old_chat_member and new_chat_member are equal.
New in version 13.4.

Note: In Python from is a reserved word, use from_user instead.

Parameters

- chat (telegram.Chat) – Chat the user belongs to.
- from_user (telegram.User) – Performer of the action, which resulted in the change.
- date (datetime.datetime) – Date the change was done in Unix time. Converted to datetime.datetime.
- old_chat_member (telegram.ChatMember) – Previous information about the chat member.
- new_chat_member (telegram.ChatMember) – New information about the chat member.
- invite_link (telegram.ChatInviteLink, optional) – Chat invite link, which was used by the user to join the chat. For joining by invite link events only.

chat
    Chat the user belongs to.
    Type telegram.Chat

from_user
    Performer of the action, which resulted in the change.
    Type telegram.User

date
    Date the change was done in Unix time. Converted to datetime.datetime.
    Type datetime.datetime

old_chat_member
    Previous information about the chat member.
    Type telegram.ChatMember

new_chat_member
    New information about the chat member.
    Type telegram.ChatMember

invite_link
    Optional. Chat invite link, which was used by the user to join the chat.
Type `telegram.ChatInviteLink`

classmethod `de_json`(data, bot)

See `telegram.TelegramObject.de_json()`.

difference()

Computes the difference between `old_chat_member` and `new_chat_member`.

Example

```python
>>> chat_member_updated.difference()
{'custom_title': ('old title', 'new title')}
```

Note: To determine, if the `telegram.ChatMember.user` attribute has changed, every attribute of the user will be checked.

New in version 13.5.

Returns A dictionary mapping attribute names to tuples of the form `(old_value, new_value)`

Return type `Dict[str, Tuple[obj, obj]]`

to_dict()

See `telegram.TelegramObject.to_dict()`.

### 3.2.27 `telegram.ChatPermissions`

class `telegram.ChatPermissions`

can_send_messages=None, can_send_media_messages=None, can_send_polls=None, can_send_other_messages=None, can_add_web_page_previews=None, can_change_info=None, can_invite_users=None, can_pin_messages=None, **_kwargs)

Bases: `telegram.base.TelegramObject`

Describes actions that a non-administrator user is allowed to take in a chat.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `can_send_messages`, `can_send_media_messages`, `can_send_polls`, `can_send_other_messages`, `can_add_web_page_previews`, `can_change_info`, `can_invite_users` and `can_pin_messages` are equal.

Note: Though not stated explicitly in the official docs, Telegram changes not only the permissions that are set, but also sets all the others to `False`. However, since not documented, this behaviour may change unbeknown to PTB.

Parameters

- **can_send_messages** (bool, optional) – True, if the user is allowed to send text messages, contacts, locations and venues.
- **can_send_media_messages** (bool, optional) – True, if the user is allowed to send audios, documents, photos, videos, video notes and voice notes, implies `can_send_messages`.
- **can_send_polls** (bool, optional) – True, if the user is allowed to send polls, implies `can_send_messages`. 

• **can_send_other_messages** (bool, optional) – True, if the user is allowed to send animations, games, stickers and use inline bots, implies **can_send_media_messages**.

• **can_add_web_page_previews** (bool, optional) – True, if the user is allowed to add web page previews to their messages, implies **can_send_media_messages**.

• **can_change_info** (bool, optional) – True, if the user is allowed to change the chat title, photo and other settings. Ignored in public supergroups.

• **can_invite_users** (bool, optional) – True, if the user is allowed to invite new users to the chat.

• **can_pin_messages** (bool, optional) – True, if the user is allowed to pin messages. Ignored in public supergroups.

**can_send_messages**
Optional. True, if the user is allowed to send text messages, contacts, locations and venues.

Type bool

**can_send_media_messages**
Optional. True, if the user is allowed to send audios, documents, photos, videos, video notes and voice notes, implies **can_send_messages**.

Type bool

**can_send_polls**
Optional. True, if the user is allowed to send polls, implies **can_send_messages**.

Type bool

**can_send_other_messages**
Optional. True, if the user is allowed to send animations, games, stickers and use inline bots, implies **can_send_media_messages**.

Type bool

**can_add_web_page_previews**
Optional. True, if the user is allowed to add web page previews to their messages, implies **can_send_media_messages**.

Type bool

**can_change_info**
Optional. True, if the user is allowed to change the chat title, photo and other settings. Ignored in public supergroups.

Type bool

**can_invite_users**
Optional. True, if the user is allowed to invite new users to the chat.

Type bool

**can_pin_messages**
Optional. True, if the user is allowed to pin messages. Ignored in public supergroups.

Type bool
3.2.28 telegram.ChatPhoto

```python
class telegram.ChatPhoto(small_file_id, small_file_unique_id, big_file_id, big_file_unique_id, 
bot=None, **kwargs)
```

**Bases:** telegram.base.TelegramObject

This object represents a chat photo.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `small_file_unique_id` and `big_file_unique_id` are equal.

**Parameters**

- `small_file_id` *(str)* – Unique file identifier of small (160x160) chat photo. This file_id can be used only for photo download and only for as long as the photo is not changed.

- `small_file_unique_id` *(str)* – Unique file identifier of small (160x160) chat photo, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the file.

- `big_file_id` *(str)* – Unique file identifier of big (640x640) chat photo. This file_id can be used only for photo download and only for as long as the photo is not changed.

- `big_file_unique_id` *(str)* – Unique file identifier of big (640x640) chat photo, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the file.

- `bot` *(telegram.Bot, optional)* – The Bot to use for instance methods

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

**small_file_id**

File identifier of small (160x160) chat photo. This file_id can be used only for photo download and only for as long as the photo is not changed.

*Type* str

**small_file_unique_id**

Unique file identifier of small (160x160) chat photo, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the file.

*Type* str

**big_file_id**

File identifier of big (640x640) chat photo. This file_id can be used only for photo download and only for as long as the photo is not changed.

*Type* str

**big_file_unique_id**

Unique file identifier of big (640x640) chat photo, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the file.

*Type* str

**get_big_file**(timeout=None, api_kwargs=None)

Convenience wrapper over `telegram.Bot.get_file` for getting the big (640x640) chat photo

For the documentation of the arguments, please see `telegram.Bot.get_file()`.

*Returns* telegram.File

*Raises* telegram.error.TelegramError –

**get_small_file**(timeout=None, api_kwargs=None)

Convenience wrapper over `telegram.Bot.get_file` for getting the small (160x160) chat photo

For the documentation of the arguments, please see `telegram.Bot.get_file()`.
Returns `telegram.File`

Raises `telegram.error.TelegramError`

### 3.2.29 telegram.constants Module

Constants in the Telegram network.

The following constants were extracted from the Telegram Bots FAQ and Telegram Bots API.

**telegram.constants.BOT_API_VERSION**

5.7. Telegram Bot API version supported by this version of `python-telegram-bot`. Also available as `telegram.bot_api_version`.

- **New in version 13.4.**
  - **Type** `str`

**telegram.constants.MAX_MESSAGE_LENGTH**

4096

- **Type** `int`

**telegram.constants.MAX_CAPTION_LENGTH**

1024

- **Type** `int`

**telegram.constants.SUPPORTED_WEBHOOK_PORTS**

[443, 80, 88, 8443]

- **Type** `List[int]`

**telegram.constants.MAX_FILESIZE_DOWNLOAD**

In bytes (20MB)

- **Type** `int`

**telegram.constants.MAX_FILESIZE_UPLOAD**

In bytes (50MB)

- **Type** `int`

**telegram.constants.MAX_PHOTOSIZE_UPLOAD**

In bytes (10MB)

- **Type** `int`

**telegram.constants.MAX_MESSAGES_PER_SECOND_PER_CHAT**

1. Telegram may allow short bursts that go over this limit, but eventually you’ll begin receiving 429 errors.

- **Type** `int`

**telegram.constants.MAX_MESSAGES_PER_SECOND**

30

- **Type** `int`

**telegram.constants.MAX_MESSAGES_PER_MINUTE_PER_GROUP**

20

- **Type** `int`

**telegram.constants.MAX_INLINE_QUERY_RESULTS**

50

- **Type** `int`
telegram.constants.MAX_ANSWER_CALLBACK_QUERY_TEXT_LENGTH
200
New in version 13.2.

Type int

The following constant have been found by experimentation:

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

Type int

telegram.constants.ANONYMOUS_ADMIN_ID
1087968824 (User id in groups for anonymous admin)

Type int

telegram.constants.SERVICE_CHAT_ID
777000 (Telegram service chat, that also acts as sender of channel posts forwarded to discussion groups)

Type int

telegram.constants.FAKE_CHANNEL_ID
136817688 (User id in groups when message is sent on behalf of a channel).

New in version 13.9.

Type int

The following constants are related to specific classes and are also available as attributes of those classes:

telegram.Chat:

telegram.constants.CHAT_PRIVATE
'private'

Type str

telegram.constants.CHAT_GROUP
'group'

Type str

telegram.constants.CHAT_SUPERGROUP
'supergroup'

Type str

telegram.constants.CHAT_CHANNEL
'channel'

Type str

telegram.constants.CHAT_SENDER
'sender'. Only relevant for telegram.InlineQuery.chat_type.

New in version 13.5.

Type str

telegram.ChatAction:

telegram.constants.CHATACTION_FIND_LOCATION
'find_location'

Type str

telegram.constants.CHATACTION_RECORD_AUDIO
'record_audio'

Deprecated since version 13.5: Deprecated by Telegram. Use CHATACTION_RECORD_VOICE instead.
Type str
telegram.constants.CHATACTION_RECORD_VOICE
'record_voice'
New in version 13.5.

Type str
telegram.constants.CHATACTION_RECORD_VIDEO
'record_video'

Type str
telegram.constants.CHATACTION_RECORD_VIDEO_NOTE
'record_video_note'

Type str
telegram.constants.CHATACTION_TYPING
'typing'

Type str
telegram.constants.CHATACTION_UPLOAD_AUDIO
'upload_audio'
Deprecated since version 13.5: Deprecated by Telegram. Use CHATACTION_UPLOAD_VOICE instead.

Type str
telegram.constants.CHATACTION_UPLOAD_VOICE
'upload_voice'
New in version 13.5.

Type str
telegram.constants.CHATACTION_UPLOAD_DOCUMENT
'upload_document'

Type str
telegram.constants.CHATACTION_CHOOSE_STICKER
'choose_sticker'
New in version 13.8.

Type str
telegram.constants.CHATACTION_UPLOAD_PHOTO
'upload_photo'

Type str
telegram.constants.CHATACTION_UPLOAD_VIDEO
'upload_video'

Type str
telegram.constants.CHATACTION_UPLOAD_VIDEO_NOTE
'upload_video_note'

Type str
telegram.ChatMember:
telegram.constants.CHATMEMBER_ADMINISTRATOR
'administrator'

Type str
telemedicine.constants.CHATMEMBER_CREATOR
    'creator'
    Type str

telemedicine.constants.CHATMEMBER_KICKED
    'kicked'
    Type str

telemedicine.constants.CHATMEMBER_LEFT
    'left'
    Type str

telemedicine.constants.CHATMEMBER_MEMBER
    'member'
    Type str

telemedicine.constants.CHATMEMBER_RESTRICTED
    'restricted'
    Type str

telemedicine.Dice:

telemedicine.constants.DICE_DICE
    '
    Type str

telemedicine.constants.DICE_DARTS
    '
    Type str

telemedicine.constants.DICE_BASKETBALL
    '
    Type str

telemedicine.constants.DICE_FOOTBALL
    '
    Type str

telemedicine.constants.DICE_SLOT_MACHINE
    '
    Type str

telemedicine.constants.DICE_BOWLING
    '
    New in version 13.4.
    Type str

telemedicine.constants.DICE_ALL_EMOJI
    List of all supported base emoji.
    Changed in version 13.4: Added DICE_BOWLING
    Type List[\[str\]]

telemedicine.MessageEntity:

telemedicine.constants.MESSAGEENTITY_MENTION
    'mention'
    Type str
telegram.constants.MESSAGEENTITY_HASHTAG
'hashtag'
    Type str

telegram.constants.MESSAGEENTITY_CASHTAG
'cashtag'
    Type str

telegram.constants.MESSAGEENTITY_PHONE_NUMBER
'phone_number'
    Type str

telegram.constants.MESSAGEENTITY_BOT_COMMAND
'bot_command'
    Type str

telegram.constants.MESSAGEENTITY_URL
'url'
    Type str

telegram.constants.MESSAGEENTITY_EMAIL
'email'
    Type str

telegram.constants.MESSAGEENTITY_BOLD
'bold'
    Type str

telegram.constants.MESSAGEENTITY_ITALIC
'italic'
    Type str

telegram.constants.MESSAGEENTITY_CODE
'code'
    Type str

telegram.constants.MESSAGEENTITY_PRE
'pre'
    Type str

telegram.constants.MESSAGEENTITY_TEXT_LINK
'text_link'
    Type str

telegram.constants.MESSAGEENTITY_TEXT_MENTION
'text_mention'
    Type str

telegram.constants.MESSAGEENTITY_UNDERLINE
'underline'
    Type str

telegram.constants.MESSAGEENTITY_STRIKETHROUGH
'strikethrough'
    Type str
telegram.constants.MESSAGEENTITY_SPOILER
'spoiler'

New in version 13.10.

Type str

telegram.constants.MESSAGEENTITY_ALL_TYPES
List of all the types of message entity.

Type List[str]

telegram.ParseMode:

telegram.constants.PARSEMODE_MARKDOWN
'Markdown'

Type str

telegram.constants.PARSEMODE_MARKDOWN_V2
'MarkdownV2'

Type str

telegram.constants.PARSEMODE_HTML
'HTML'

Type str

telegram.Poll:

telegram.constants.POLL_REGULAR
'regular'

Type str

telegram.constants.POLL QUIZ
'quiz'

Type str

telegram.constants.MAX_POLL_QUESTION_LENGTH
300

Type int

telegram.constants.MAX_POLL_OPTION_LENGTH
100

Type int

telegram.MaskPosition:

telegram.constants.STICKER_FOREHEAD
'forehead'

Type str

telegram.constants.STICKER EYES
'eyes'

Type str

telegram.constants.STICKER MOUTH
'mouth'

Type str

telegram.constants.STICKER CHIN
'chin'

Type str
teleogram.Update:

telegram.constants.UPDATE_MESSAGE
    'message'
        New in version 13.5.
        Type str

telegram.constants.UPDATE_EDITED_MESSAGE
    'edited_message'
        New in version 13.5.
        Type str

telegram.constants.UPDATE_CHANNEL_POST
    'channel_post'
        New in version 13.5.
        Type str

telegram.constants.UPDATE_EDITED_CHANNEL_POST
    'edited_channel_post'
        New in version 13.5.
        Type str

telegram.constants.UPDATE_INLINE_QUERY
    'inline_query'
        New in version 13.5.
        Type str

telegram.constants.UPDATE_CHOSEN_INLINE_RESULT
    'chosen_inline_result'
        New in version 13.5.
        Type str

telegram.constants.UPDATE_CALLBACK_QUERY
    'callback_query'
        New in version 13.5.
        Type str

telegram.constants.UPDATE_SHIPPING_QUERY
    'shipping_query'
        New in version 13.5.
        Type str

telegram.constants.UPDATE_PRE_CHECKOUT_QUERY
    'pre_checkout_query'
        New in version 13.5.
        Type str

telegram.constants.UPDATE_POLL
    'poll'
        New in version 13.5.
        Type str
telegram.constants.UPDATE_POLL_ANSWER
  'poll_answer'
  New in version 13.5.
  Type str

telegram.constants.UPDATE_MY_CHAT_MEMBER
  'my_chat_member'
  New in version 13.5.
  Type str

telegram.constants.UPDATE_CHAT_MEMBER
  'chat_member'
  New in version 13.5.
  Type str

telegram.constants.UPDATE_CHAT_JOIN_REQUEST
  'chat_join_request'
  New in version 13.8.
  Type str

telegram.constants.UPDATE_ALL_TYPES
  List of all update types.
  New in version 13.5.
  Changed in version 13.8.
  Type List[str]

telegram.BotCommandScope:

telegram.constants.BOT_COMMAND_SCOPE_DEFAULT
  'default'
  ..versionadded:: 13.7
  Type str

telegram.constants.BOT_COMMAND_SCOPE_ALL_PRIVATE_CHATS
  'all_private_chats'
  ..versionadded:: 13.7
  Type str

telegram.constants.BOT_COMMAND_SCOPE_ALL_GROUP_CHATS
  'all_group_chats'
  ..versionadded:: 13.7
  Type str

telegram.constants.BOT_COMMAND_SCOPE_ALL_CHAT_ADMINISTRATORS
  'all_chat_administrators'
  ..versionadded:: 13.7
  Type str

telegram.constants.BOT_COMMAND_SCOPE_CHAT
  'chat'
  ..versionadded:: 13.7
  Type str
telegram.constants.BOT_COMMAND_SCOPE_CHAT_ADMINISTRATORS
    'chat_administrators'
    ..versionadded:: 13.7
    Type str
telegram.constants.BOT_COMMAND_SCOPE_CHAT_MEMBER
    'chat_member'
    ..versionadded:: 13.7
    Type str

3.2.30 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.
    Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their phone_number is equal.
    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.

phone_number
    Contact’s phone number.
    Type str

first_name
    Contact’s first name.
    Type str

last_name
    Optional. Contact’s last name.
    Type str

user_id
    Optional. Contact’s user identifier in Telegram.
    Type int

vcard
    Optional. Additional data about the contact in the form of a vCard.
    Type str
class telegram.Dice(value, emoji, **kwargs)
Bases: telegram.base.TelegramObject

This object represents an animated emoji with a random value for currently supported base emoji. (The singular form of “dice” is “die”. However, PTB mimics the Telegram API, which uses the term “dice”.)

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `value` and `emoji` are equal.

Note: If `emoji` is “”, a value of 6 currently represents a bullseye, while a value of 1 indicates that the dartboard was missed. However, this behaviour is undocumented and might be changed by Telegram.

If `emoji` is “”, a value of 4 or 5 currently score a basket, while a value of 1 to 3 indicates that the basket was missed. However, this behaviour is undocumented and might be changed by Telegram.

If `emoji` is “”, a value of 4 to 5 currently scores a goal, while a value of 1 to 3 indicates that the goal was missed. However, this behaviour is undocumented and might be changed by Telegram.

If `emoji` is “”, a value of 6 knocks all the pins, while a value of 1 means all the pins were missed. However, this behaviour is undocumented and might be changed by Telegram.

If `emoji` is “”, each value corresponds to a unique combination of symbols, which can be found at our wiki. However, this behaviour is undocumented and might be changed by Telegram.

Parameters

- **value** (int) – Value of the dice. 1-6 for dice, darts and bowling balls, 1-5 for basketball and football/soccer ball, 1-64 for slot machine.

- **emoji** (str) – Emoji on which the dice throw animation is based.

value

Value of the dice.

Type int

emoji

Emoji on which the dice throw animation is based.

Type str

```
ALL_EMOJI: ClassVar[List[str]] = ['', '', '', '', '']
telegram.constants.DICE_ALL_EMOJI

BASKETBALL: ClassVar[str] = ''
telegram.constants.DICE_BASKETBALL

BOWLING: ClassVar[str] = ''
telegram.constants.DICE_BOWLING

DARTS: ClassVar[str] = ''
telegram.constants.DICE_DARTS

FOOTBALL: ClassVar[str] = ''
telegram.constants.DICE_FOOTBALL

SLOT_MACHINE: ClassVar[str] = ''
telegram.constants.DICE_SLOT_MACHINE
```

New in version 13.4.
3.2.32 telegram.Document

class telegram.Document (file_id, file_unique_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).

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their file_unique_id is equal.

Parameters

- **file_id**(str) – Identifier for this file, which can be used to download or reuse the file.
- **file_unique_id**(str) – Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the file.
- **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.

file_id
    File identifier.
    Type str

file_unique_id
    Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the file.
    Type str

thumb
    Type telegram.PhotoSize

file_name
    Original filename.
    Type str

mime_type
    Optional. MIME type of the file.
    Type str

file_size
    Optional. File size.
    Type int

bot
    Optional. The Bot to use for instance methods.
    Type telegram.Bot

classmethod de_json (data, bot)
    See telegram.TelegramObject.de_json().
get_file(timeout=None, api_kwargs=None)

Convenience wrapper over telegram.Bot.get_file

For the documentation of the arguments, please see telegram.Bot.get_file().

Returns telegram.File

Raises telegram.error.TelegramError

3.2.33 telegram.error module

This module contains an object that represents Telegram errors.

exception telegram.error.BadRequest(message)
    Bases: telegram.error.NetworkError
    Raised when Telegram could not process the request correctly.

exception telegram.error.ChatMigrated(new_chat_id)
    Bases: telegram.error.TelegramError
    Raised when the requested group chat migrated to supergroup and has a new chat id.

    Parameters
    new_chat_id (int) -- The new chat id of the group.

exception telegram.error.Conflict(message)
    Bases: telegram.error.TelegramError
    Raised when a long poll or webhook conflicts with another one.

exception telegram.error.InvalidToken
    Bases: telegram.error.TelegramError
    Raised when the token is invalid.

exception telegram.error.NetworkError(message)
    Bases: telegram.error.TelegramError
    Base class for exceptions due to networking errors.

exception telegram.error.RetryAfter(retry_after)
    Bases: telegram.error.TelegramError
    Raised when flood limits where exceeded.

    Parameters
    retry_after (int) -- Time in seconds, after which the bot can retry the request.

exception telegram.error.TelegramError(message)
    Bases: Exception
    Base class for Telegram errors.

exception telegram.error.TimedOut
    Bases: telegram.error.NetworkError
    Raised when a request took too long to finish.

exception telegram.error.Unauthorized(message)
    Bases: telegram.error.TelegramError
    Raised when the bot has not enough rights to perform the requested action.
3.2.34 telegram.File

```python
class telegram.File(file_id, file_unique_id, bot=None, file_size=None, file_path=None, **kwargs)
```

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 `telegram.Bot.get_file()`.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `file_unique_id` is equal.

Note:
- Maximum file size to download is 20 MB.
- 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()`.

Parameters

- `file_id` (str) – Identifier for this file, which can be used to download or reuse the file.
- `file_unique_id` (str) – Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the 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.

`file_id`

Identifier for this file.

Type str

`file_unique_id`

Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the file.

Type str

`file_size`

Optional. File size.

Type str

`file_path`

Optional. File path. Use `download` to get the file.

Type str

`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 the file has no filename, it the file ID will be used as filename. 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.
• If neither custom_path nor out is provided and file_path is the path of a local file (which is the case when a Bot API Server is running in local mode), this method will just return the path.

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 or the file path of the local file.

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)
Sets the passport credentials for the file.

Parameters credentials (telegram.FileCredentials) – The credentials.

3.2.35 telegram.ForceReply
class telegram.ForceReply (force_reply=True, selective=False, input_field_placeholder=None, **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.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their selective is equal.

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 telegram.Message object.
  2) If the bot’s message is a reply (has reply_to_message_id), sender of the original message.
• input_field_placeholder (str, optional) – The placeholder to be shown in the input field when the reply is active; 1-64 characters.
  New in version 13.7.
• **kwargs (dict) – Arbitrary keyword arguments.
force_reply
  Shows reply interface to the user, as if they manually selected the bots message and tapped 'Reply'.
  Type  True

selective
  Optional. Force reply from specific users only.
  Type  bool

input_field_placeholder
  Optional. The placeholder shown in the input field when the reply is active.
  New in version 13.7.
  Type  str

3.2.36 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, login_url=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents one button of an inline keyboard.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their text, url, login_url, callback_data, switch_inline_query, switch_inline_query_current_chat, callback_game and pay are equal.

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.

• If your bot allows for arbitrary callback data, in keyboards returned in a response from telegram, callback_data may be an instance of telegram.ext.InvalidCallbackData. This will be the case, if the data associated with the button was already deleted.
  New in version 13.6.

• Since Bot API 5.5, it's now allowed to mention users by their ID in inline keyboards. This will only work in Telegram versions released after December 7, 2021. Older clients will display unsupported message.

Warning: If your bot allows your arbitrary callback data, buttons whose callback data is a non-hashable object will become unhashable. Trying to evaluate hash(button) will result in a TypeError.

Changed in version 13.6.

Parameters

• text (str) – Label text on the button.

• url (str, optional) – HTTP or tg://url to be opened when the button is pressed. Links tg://user?id=<user_id> can be used to mention a user by their ID without using a username, if this is allowed by their privacy settings.

  Changed in version 13.9: You can now mention a user using tg://user? id=<user_id>.
**login_url** *(telegraf.LoginUrl, optional) – An HTTP URL used to automatically authorize the user. Can be used as a replacement for the Telegram Login Widget.*

**callback_data** *(str|object, optional) – Data to be sent in a callback query to the bot when button is pressed, UTF-8 1-64 bytes. If the bot instance allows arbitrary callback data, anything can be passed.*

**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 a 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** *(telegraf.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 and can only be used in invoice messages.*

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

text

Label text on the button.

**Type** str

url

Optional. HTTP or tg:// url to be opened when the button is pressed. Links tg://user?id=<user_id> can be used to mention a user by their ID without using a username, if this is allowed by their privacy settings.

Changed in version 13.9: You can now mention a user using tg://user?id=<user_id>.

**Type** str

login_url

Optional. An HTTP URL used to automatically authorize the user. Can be used as a replacement for the Telegram Login Widget.

**Type** telegraf>LoginUrl

callback_data

Optional. Data to be sent in a callback query to the bot when button is pressed, UTF-8 1-64 bytes.

**Type** str|object

switch_inline_query

Optional. Will prompt the user to select one of their chats, open that chat and insert the bot’s username and the specified inline query in the input field. Can be empty, in which case just the bot’s username will be inserted.

**Type** str

switch_inline_query_current_chat

Optional. Will insert the bot’s username and the specified inline query in the current chat’s input field. Can be empty, in which case just the bot’s username will be inserted.

**Type** str
callback_game
Optional. Description of the game that will be launched when the user presses the button.

Type telegram.CallbackGame

pay
Optional. Specify True, to send a Pay button.

Type bool

update_callback_data(callback_data)
Sets callback_data to the passed object. Intended to be used by telegram.ext.CallbackDataCache.

New in version 13.6.

Parameters callback_data(obj) – The new callback data.

3.2.37 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.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their size of inline_keyboard and all the buttons are equal.

Parameters

• inline_keyboard (List[List[telegram.InlineKeyboardButton]]) – List of button rows, each represented by a list of InlineKeyboardButton objects.

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

inline_keyboard
List of button rows, each represented by a list of InlineKeyboardButton objects.

Type List[List[telegram.InlineKeyboardButton]]

classmethod de_json(data, bot)
See telegram.TelegramObject.de_json().

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:

```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.**

```python
to_dict()
```

See `telegram.TelegramObject.to_dict()`.

### 3.2.38 telegram.InputFile

**class** `telegram.InputFile(obj, filename=None, attach=None)`

Bases: `object`

This object represents a Telegram InputFile.

**Parameters**

- `obj` (File handler | bytes) – An open file descriptor or the files content as bytes.
- `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` –

**input_file_content**

The binary content of the file to send.

Type `bytes`

**filename**

Optional. Filename for the file to be sent.

Type `str`

**attach**

Optional. Attach id for sending multiple files.

Type `str`

**static is_image(stream)**

Check if the content file is an image by analyzing its headers.

**Parameters** `stream` (bytes) – A byte stream representing the content of a file.

**Returns** The mime-type of an image, if the input is an image, or `None` else.

**Return type** `str` | `None`

```python
to_dict()
```

See `telegram.TelegramObject.to_dict()`.
### 3.2.39 `telegram.InputMedia`

class `telegram.InputMedia`

Bases: `telegram.base.TelegramObject`

Base class for Telegram InputMedia Objects.


`to_dict()`

See `telegram.TelegramObject.to_dict()`.

### 3.2.40 `telegram.InputMediaAnimation`

class `telegram.InputMediaAnimation` *(media, thumb=None, caption=None, parse_mode=None, width=None, height=None, duration=None, caption_entities=None, filename=None)*

Bases: `telegram.files.inputmedia.InputMedia`

Represents an animation file (GIF or H.264/MPEG-4 AVC video without sound) to be sent.

**Parameters**

- **media** *(str | filelike object | bytes | pathlib.Path | telegram.Animation)*
  - 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.

  Changed in version 13.2: Accept `bytes` as input.

- **filename** *(str, optional)*
  - Custom file name for the animation, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the `tempfile` module.


- **thumb** *(filelike object | bytes | pathlib.Path, optional)*
  - Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can’t be reused and can be only uploaded as a new file.

  Changed in version 13.2: Accept `bytes` as input.

- **caption** *(str, optional)*
  - Caption of the animation to be sent, 0-1024 characters after entities parsing.

- **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.

- **caption_entities** *(List[telegram.MessageEntity], optional)*
  - List of special entities that appear in the caption, which can be specified instead of `parse_mode`.

- **width** *(int, optional)*
  - Animation width.

- **height** *(int, optional)*
  - Animation height.
• **duration** *(int, optional)* – Animation duration.

**type**

   animation.

   Type str

**media**

   Animation to send.

   Type str | `telegram.InputFile`

**caption**

   Optional. Caption of the document to be sent.

   Type str

**parse_mode**

   Optional. The parse mode to use for text formatting.

   Type str

**caption_entities**

   Optional. List of special entities that appear in the caption.

   Type List[`telegram.MessageEntity`]

**thumb**

   Optional. Thumbnail of the file to send.

   Type `telegram.InputFile`

**width**

   Optional. Animation width.

   Type int

**height**

   Optional. Animation height.

   Type int

**duration**

   Optional. Animation duration.

   Type int

### 3.2.41 `telegram.InputMediaAudio`

**class** `telegram.InputMediaAudio`(media, thumb=None, caption=None, parse_mode=None, duration=None, performer=None, title=None, caption_entities=None, filename=None)

**Bases:** `telegram.files.inputmedia.InputMedia`

Represents an audio file to be treated as music to be sent.

---

**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.

**Parameters**

• **media** *(str | filelike object | bytes | pathlib.Path | `telegram.Audio`)* – 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.
Changed in version 13.2: Accept `bytes` as input.

- **filename** *(str, optional)* – Custom file name for the audio, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the `tempfile` module.
  

- **caption** *(str, optional)* – Caption of the audio to be sent, 0-1024 characters after entities parsing.

- **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.

- **caption_entities** *(List[telegram.MessageEntity], optional)* – List of special entities that appear in the caption, which can be specified instead of `parse_mode`.

- **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 | bytes | pathlib.Path, optional)* – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can’t be reused and can be only uploaded as a new file.

  Changed in version 13.2: Accept `bytes` as input.

```python
    type
        audio.
        Type str

    media
        Audio file to send.
        Type str | telegram.InputFile

    caption
        Optional. Caption of the document to be sent.
        Type str

    parse_mode
        Optional. The parse mode to use for text formatting.
        Type str

    caption_entities
        Optional. List of special entities that appear in the caption.
        Type List[telegram.MessageEntity]

    duration
        Duration of the audio in seconds.
        Type int

    performer
        Optional. Performer of the audio as defined by sender or by audio tags.
        Type str
```

3.2. telegram package
**title**
Optional. Title of the audio as defined by sender or by audio tags.

*Type str*

**thumb**
Optional. Thumbnail of the file to send.

*Type telegram.InputFile*

### 3.2.42 telegram.InputMediaDocument

class telegram.InputMediaDocument (media, thumb=None, caption=None, parse_mode=None, disable_content_type_detection=None, caption_entities=None, filename=None)

**Bases:** telegram.files.inputmedia.InputMedia

Represents a general file to be sent.

**Parameters**

- **media** *(str | filelike object | bytes | pathlib.Path | telegram.Document)*
  - 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.

  Changed in version 13.2: Accept bytes as input.

- **filename** *(str, optional)*
  - Custom file name for the document, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the `tempfile` module.


- **caption** *(str, optional)*
  - Caption of the document to be sent, 0-1024 characters after entities parsing.

- **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.

- **caption_entities** *(List[telegram.MessageEntity], optional)*
  - List of special entities that appear in the caption, which can be specified instead of parse_mode.

- **thumb** *(filelike object | bytes | pathlib.Path, optional)*
  - Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can’t be reused and can be only uploaded as a new file.

  Changed in version 13.2: Accept bytes as input.

- **disable_content_type_detection** *(bool, optional)*
  - Disables automatic server-side content type detection for files uploaded using multipart/form-data. Always true, if the document is sent as part of an album.

**type**

document.

*Type str*

**media**
File to send.

*Type str | telegram.InputFile*
caption
Optional. Caption of the document to be sent.
Type str

parse_mode
Optional. The parse mode to use for text formatting.
Type str

caption_entities
Optional. List of special entities that appear in the caption.
Type List[telegram.MessageEntity]

thumb
Optional. Thumbnail of the file to send.
Type telegram.InputFile

disable_content_type_detection
Optional. Disables automatic server-side content type detection for files uploaded using multipart/form-data. Always true, if the document is sent as part of an album.
Type bool

3.2.43 telegram.InputMediaPhoto

class telegram.InputMediaPhoto (media, caption=None, parse_mode=None, caption_entities=None, filename=None)
Bases: telegram.files.inputmedia.InputMedia

Represents a photo to be sent.

Parameters

• media (str | filelike object | bytes | pathlib.Path | telegram.PhotoSize) – 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.

Changed in version 13.2: Accept bytes as input.

• filename (str, optional) – Custom file name for the photo, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the tempfile module.


• caption (str, optional) – Caption of the photo to be sent, 0-1024 characters after entities parsing.

• 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.

• caption_entities (List[telegram.MessageEntity], optional) – List of special entities that appear in the caption, which can be specified instead of parse_mode.

type
type str

media
Photo to send.
Type str | telegram.InputFile
caption
Optional. Caption of the document to be sent.

Type str

parse_mode
Optional. The parse mode to use for text formatting.

Type str

caption_entities
Optional. List of special entities that appear in the caption.

Type List[telegram.MessageEntity]

3.2.44 telegram.InputMediaVideo

class telegram.InputMediaVideo(media, caption=None, width=None, height=None, duration=None, supports_streaming=None, parse_mode=None, thumb=None, caption_entities=None, filename=None)

Bases: telegram.files.inputmedia.InputMedia

Represents a video to be sent.

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.

• thumb will be ignored for small video files, for which Telegram can easily generate thumbnails. However, this behaviour is undocumented and might be changed by Telegram.

Parameters

• media (str|filelike object|bytes|pathlib.Path|telegram.Video) – 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.

  Changed in version 13.2: Accept bytes as input.

• filename (str, optional) – Custom file name for the video, when uploading a new file. Convenience parameter, useful e.g. when sending files generated by the tempfile module.


• caption (str, optional) – Caption of the video to be sent, 0-1024 characters after entities parsing.

• 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.

• caption_entities (List[telegram.MessageEntity], optional) – List of special entities that appear in the caption, which can be specified instead of parse_mode.

• 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 | bytes | pathlib.Path, optional)* – Thumbnail of the file sent; can be ignored if thumbnail generation for the file is supported server-side. The thumbnail should be in JPEG format and less than 200 kB in size. A thumbnail’s width and height should not exceed 320. Ignored if the file is not uploaded using multipart/form-data. Thumbnails can’t be reused and can be only uploaded as a new file.

  Changed in version 13.2: Accept `bytes` as input.

  ```python
  type
      video.
      Type str
  media
      Video file to send.
      Type str | telegram.InputFile
  caption
      Optional. Caption of the document to be sent.
      Type str
  parse_mode
      Optional. The parse mode to use for text formatting.
      Type str
  caption_entities
      Optional. List of special entities that appear in the caption.
      Type List[telegram.MessageEntity]
  width
      Optional. Video width.
      Type int
  height
      Optional. Video height.
      Type int
  duration
      Optional. Video duration.
      Type int
  supports_streaming
      Optional. Pass True, if the uploaded video is suitable for streaming.
      Type bool
  thumb
      Optional. Thumbnail of the file to send.
      Type telegram.InputFile
  ```
3.2.45 \texttt{telegram.KeyboardButton}

\begin{Verbatim}[commandchars=\{}\end{Verbatim}

```python
class telegram.KeyboardButton(text, request_contact=None, request_location=None, request_poll=None, **_kwargs)
```

Bases: \texttt{telegram.base.TelegramObject}

This object represents one button of the reply keyboard. For simple text buttons \texttt{String} can be used instead of this object to specify text of the button.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their \texttt{text}, \texttt{request_contact}, \texttt{request_location} and \texttt{request_poll} are equal.

\textbf{Note:}

- Optional fields are mutually exclusive.
- \texttt{request_contact} and \texttt{request_location} options will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.
- \texttt{request_poll} option will only work in Telegram versions released after 23 January, 2020. Older clients will receive unsupported message.

\textbf{Parameters}

- \texttt{text} (\texttt{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.
- \texttt{request_contact} (\texttt{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.
- \texttt{request_location} (\texttt{bool}, optional) – If True, the user’s current location will be sent when the button is pressed. Available in private chats only.
- \texttt{request_poll} (\texttt{KeyboardButtonPollType}, optional) – If specified, the user will be asked to create a poll and send it to the bot when the button is pressed. Available in private chats only.

\texttt{text}

Text of the button.

\textbf{Type} \texttt{str}

\texttt{request_contact}

Optional. The user’s phone number will be sent.

\textbf{Type} \texttt{bool}

\texttt{request_location}

Optional. The user’s current location will be sent.

\textbf{Type} \texttt{bool}

\texttt{request_poll}

Optional. If the user should create a poll.

\textbf{Type} \texttt{KeyboardButtonPollType}
3.2.46 telegram.KeyboardButtonPollType

class telegram.KeyboardButtonPollType(type=None, **kwargs)
  Bases: telegram.base.TelegramObject

This object represents type of a poll, which is allowed to be created and sent when the corresponding button is pressed.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `type` is equal.

  `type`
  Optional. If `telegram.Poll.QUIZ` is passed, the user will be allowed to create only polls in the quiz mode. If `telegram.Poll.REGULAR` is passed, only regular polls will be allowed. Otherwise, the user will be allowed to create a poll of any type.

  Type str

3.2.47 telegram.Location

class telegram.Location(longitude, latitude, horizontal_accuracy=None, live_period=None, heading=None, proximity_alert_radius=None, **kwargs)
  Bases: telegram.base.TelegramObject

This object represents a point on the map.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `longitude` and `latitude` are equal.

  Parameters
  
  • `longitude` (float) – Longitude as defined by sender.
  • `latitude` (float) – Latitude as defined by sender.
  • `horizontal_accuracy` (float, optional) – The radius of uncertainty for the location, measured in meters; 0-1500.
  • `live_period` (int, optional) – Time relative to the message sending date, during which the location can be updated, in seconds. For active live locations only.
  • `heading` (int, optional) – The direction in which user is moving, in degrees; 1-360. For active live locations only.
  • `proximity_alert_radius` (int, optional) – Maximum distance for proximity alerts about approaching another chat member, in meters. For sent live locations only.
  • `**kwargs` (dict) – Arbitrary keyword arguments.

  `longitude`
  Longitude as defined by sender.

  Type float

  `latitude`
  Latitude as defined by sender.

  Type float

  `horizontal_accuracy`
  Optional. The radius of uncertainty for the location, measured in meters.

  Type float

  `live_period`
  Optional. Time relative to the message sending date, during which the location can be updated, in seconds. For active live locations only.
3.2.48 telegram.LoginUrl

class telegram.LoginUrl(url, forward_text=None, bot_username=None, request_write_access=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents a parameter of the inline keyboard button used to automatically authorize a user. Serves as a great replacement for the Telegram Login Widget when the user is coming from Telegram. All the user needs to do is tap/click a button and confirm that they want to log in. Telegram apps support these buttons as of version 5.7.

Sample bot: @discussbot

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `url` is equal.

**Note:** You must always check the hash of the received data to verify the authentication and the integrity of the data as described in Checking authorization

**Parameters**

- `url (str)` – An HTTP URL to be opened with user authorization data added to the query string when the button is pressed. If the user refuses to provide authorization data, the original URL without information about the user will be opened. The data added is the same as described in Receiving authorization data

- `forward_text (str, optional)` – New text of the button in forwarded messages.

- `bot_username (str, optional)` – Username of a bot, which will be used for user authorization. See Setting up a bot for more details. If not specified, the current bot’s username will be assumed. The url’s domain must be the same as the domain linked with the bot. See Linking your domain to the bot for more details.

- `request_write_access (bool, optional)` – Pass `True` to request the permission for your bot to send messages to the user.

**url**

An HTTP URL to be opened with user authorization data.

Type `str`

**forward_text**

Optional. New text of the button in forwarded messages.

Type `str`

**bot_username**

Optional. Username of a bot, which will be used for user authorization.

Type `str`
**request_write_access**

Optional. Pass `True` to request the permission for your bot to send messages to the user.

Type `bool`

### 3.2.49 telegram.Message

```python
class telegram.Message
```

Bases: `telegram.base.TelegramObject`

This object represents a message.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `message_id` and `chat` are equal.

**Note:** In Python `from` is a reserved word, use `from_user` instead.

**Parameters**

- `message_id` *(int)* – Unique message identifier inside this chat.
- `from_user` *(telegram.User, optional)* – Sender of the message; empty for messages sent to channels. For backward compatibility, this will contain a fake sender user in non-channel chats, if the message was sent on behalf of a chat.
- `sender_chat` *(telegram.Chat, optional)* – Sender of the message, sent on behalf of a chat. For example, the channel itself for channel posts, the supergroup itself for messages from anonymous group administrators, the linked channel for messages automatically forwarded to the discussion group. For backward compatibility, `from_user` contains a fake sender user in non-channel chats, if the message was sent on behalf of a chat.
- `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 channels or from anonymous administrators, information about the original sender chat.

**forward_from_message_id** (*int*, optional) – For forwarded channel posts, identifier of the original message in the channel.

**forward_sender_name** (*str*, optional) – Sender’s name for messages forwarded from users who disallow adding a link to their account in forwarded messages.

**forward_date** (*datetime.datetime*, optional) – For forwarded messages, date the original message was sent in Unix time. Converted to `datetime.datetime`.

**is_automatic_forward** (*bool*, optional) – True, if the message is a channel post that was automatically forwarded to the connected discussion group.

New in version 13.9.

**reply_to_message** (*telegram.Message*, optional) – For replies, the original message.

**edit_date** (*datetime.datetime*, optional) – Date the message was last edited in Unix time. Converted to `datetime.datetime`.

**has_protected_content** (*bool*, optional) – True, if the message can’t be forwarded.

New in version 13.9.

**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 `parse_entity` and `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 animation, audio, document, photo, video or voice, 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. For backward compatibility, when this field is set, the location field will also be set.

• **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 changed 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.


New in version 13.4.

• **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) – For messages forwarded from channels, signature of the post author if present.

• **author_signature** (str, optional) – Signature of the post author for messages in channels, or the custom title of an anonymous group administrator.

• **passport_data** (telegram.PassportData, optional) – Telegram Passport data.

• **poll** (telegram.Poll, optional) – Message is a native poll, information about the poll.

• **dice** (telegram.Dice, optional) – Message is a dice with random value from 1 to 6.

• **via_bot** (telegram.User, optional) – Message was sent through an inline bot.

• **proximity_alert_triggered** (telegram.ProximityAlertTriggered, optional) – Service message. A user in the chat triggered another user’s proximity alert while sharing Live Location.

• **voice_chat_scheduled** (telegram.VoiceChatScheduled, optional) – Service message: voice chat scheduled.

  New in version 13.5.

• **voice_chat_started** (telegram.VoiceChatStarted, optional) – Service message: voice chat started.

  New in version 13.4.


  New in version 13.4.

• **voice_chat_participants_invited** (telegram.VoiceChatParticipantsInvited, optional) – Service message: new participants invited to a voice chat.

  New in version 13.4.

• **reply_markup** (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message. login_url buttons are represented as ordinary url buttons.

• **bot** (telegram.Bot, optional) – The Bot to use for instance methods.

---

message_id
Unique message identifier inside this chat.

Type: int

from_user
Optional. Sender of the message; empty for messages sent to channels. For backward compatibility, this will contain a fake sender user in non-channel chats, if the message was sent on behalf of a chat.
Type `telegram.User`

**sender_chat**
Optional. Sender of the message, sent on behalf of a chat. For backward compatibility, `from_user` contains a fake sender user in non-channel chats, if the message was sent on behalf of a chat.

Type `telegram.Chat`

**date**
Date the message was sent.

Type `datetime.datetime`

**chat**
Conversation the message belongs to.

Type `telegram.Chat`

**forward_from**
Optional. Sender of the original message.

Type `telegram.User`

**forward_from_chat**
Optional. For messages forwarded from channels or from anonymous administrators, information about the original sender chat.

Type `telegram.Chat`

**forward_from_message_id**
Optional. Identifier of the original message in the channel.

Type `int`

**forward_date**
Optional. Date the original message was sent.

Type `datetime.datetime`

**is_automatic_forward**
Optional. `True`, if the message is a channel post that was automatically forwarded to the connected discussion group.

New in version 13.9.

Type `bool`

**reply_to_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.

Type `telegram.Message`

**edit_date**
Optional. Date the message was last edited.

Type `datetime.datetime`

**has_protected_content**
Optional. `True`, if the message can’t be forwarded.

New in version 13.9.

Type `bool`

**media_group_id**
Optional. The unique identifier of a media message group this message belongs to.

Type `str`

**text**
Optional. The actual UTF-8 text of the message.
Type `str`  

`entities`  
Optional. Special entities like usernames, URLs, bot commands, etc. that appear in the text. See `Message.parse_entity` and `parse_entities` methods for how to use properly.  
Type `List[telegram.MessageEntity]`

`caption_entities`  
Optional. Special entities like usernames, URLs, bot commands, etc. that appear in the caption. See `Message.parse_caption_entity` and `parse_caption_entities` methods for how to use properly.  
Type `List[telegram.MessageEntity]`

`audio`  
Optional. Information about the file.  
Type `telegram.Audio`

`document`  
Optional. Information about the file.  
Type `telegram.Document`

`animation`  
For backward compatibility, when this field is set, the document field will also be set.  
Type `telegram.Animation`

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

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

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

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

`voice`  
Optional. Information about the file.  
Type `telegram.Voice`

`video_note`  
Optional. Information about the video message.  
Type `telegram.VideoNote`

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

`caption`  
Optional. Caption for the document, photo or video, 0-1024 characters.  
Type `str`

`contact`  
Optional. Information about the contact.
location
Optional. Information about the location.
Type telegram.Location

venue
Optional. Information about the venue.
Type telegram.Venue

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

new_chat_title
Optional. A chat title was changed to this value.
Type str

new_chat_photo
Optional. A chat photo was changed to this value.
Type List[telegram.PhotoSize]

delete_chat_photo
Optional. The chat photo was deleted.
Type bool

group_chat_created
Optional. The group has been created.
Type bool

supergroup_chat_created
Optional. The supergroup has been created.
Type bool

channel_chat_created
Optional. The channel has been created.
Type bool

message_auto_delete_timer_changed
New in version 13.4.
Type telegram.MessageAutoDeleteTimerChanged

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

migrate_from_chat_id
Optional. The supergroup has been migrated from a group with the specified identifier.
Type int

pinned_message
Optional. Specified message was pinned.
Type telegram.message

invoice
Optional. Information about the invoice.
Type telegram.Invoice
**successful_payment**
Optional. Information about the payment.

Type `telegram.SuccessfulPayment`

**connected_website**
Optional. The domain name of the website on which the user has logged in.

Type `str`

**forward_signature**
Optional. Signature of the post author for messages forwarded from channels.

Type `str`

**forward_sender_name**
Optional. Sender’s name for messages forwarded from users who disallow adding a link to their account in forwarded messages.

Type `str`

**author_signature**
Optional. Signature of the post author for messages in channels, or the custom title of an anonymous group administrator.

Type `str`

**passport_data**
Optional. Telegram Passport data.

Type `telegram.PassportData`

**poll**
Optional. Message is a native poll, information about the poll.

Type `telegram.Poll`

**dice**
Optional. Message is a dice.

Type `telegram.Dice`

**via_bot**
Optional. Bot through which the message was sent.

Type `telegram.User`

**proximity_alert_triggered**
Optional. Service message. A user in the chat triggered another user’s proximity alert while sharing Live Location.

Type `telegram.ProximityAlertTriggered`

**voice_chat_scheduled**
Optional. Service message: voice chat scheduled.

New in version 13.5.

Type `telegram.VoiceChatScheduled`

**voice_chat_started**
Optional. Service message: voice chat started.

New in version 13.4.

Type `telegram.VoiceChatStarted`

**voice_chat_ended**
Optional. Service message: voice chat ended.

New in version 13.4.
Type `telegram.VoiceChatEnded`

**voice_chat_participants_invited**
Optional. Service message: new participants invited to a voice chat.
New in version 13.4.
Type `telegram.VoiceChatParticipantsInvited`

**reply_markup**
Optional. Inline keyboard attached to the message.
Type `telegram.InlineKeyboardMarkup`

**bot**
Optional. The Bot to use for instance methods.
Type `telegram.Bot`

**property 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.

Changed in version 13.10: Spoiler entities are now formatted as HTML.

**Returns**
Message caption with caption entities formatted as HTML.

**Return type** `str`

**property caption_html_urled**
Creates an HTML-formatted string from the markup entities found in the message’s caption.

Use this if you want to retrieve the message caption with the caption entities formatted as HTML. This also formats `telegram.MessageEntity.URL` as a hyperlink.

Changed in version 13.10: Spoiler entities are now formatted as HTML.

**Returns**
Message caption with caption entities formatted as HTML.

**Return type** `str`

**property caption_markdown**
Creates an Markdown-formatted string from the markup entities found in the message’s caption using `telegram.ParseMode.MARKDOWN`.

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.

**Note:** `telegram.ParseMode.MARKDOWN` is is a legacy mode, retained by Telegram for backward compatibility. You should use `caption_markdown_v2()` instead.

**Returns**
Message caption with caption entities formatted as Markdown.

**Return type** `str`

**Raises** `ValueError` – If the message contains underline, strikethrough, spoiler or nested entities.

**property caption_markdown_urled**
Creates an Markdown-formatted string from the markup entities found in the message’s caption using `telegram.ParseMode.MARKDOWN`.

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.
Note: `telegram.ParseMode.MARKDOWN` is a legacy mode, retained by Telegram for backward compatibility. You should use `caption_markdown_v2_url()` instead.

**Returns** Message caption with caption entities formatted as Markdown.

**Return type** `str`

**Raises** `ValueError` – If the message contains underline, strikethrough, spoiler or nested entities.

**property caption_markdown_v2**

Creates a Markdown-formatted string from the markup entities found in the message’s caption using `telegram.ParseMode.MARKDOWN_V2`.

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.

Changed in version 13.10: Spoiler entities are now formatted as Markdown V2.

**Returns** Message caption with caption entities formatted as Markdown.

**Return type** `str`

**property caption_markdown_v2_url**

Creates a Markdown-formatted string from the markup entities found in the message’s caption using `telegram.ParseMode.MARKDOWN_V2`.

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.

Changed in version 13.10: Spoiler entities are now formatted as Markdown V2.

**Returns** Message caption with caption entities formatted as Markdown.

**Return type** `str`

**property chat_id**

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

**Type** `int`

**copy**(`chat_id`, `caption=None`, `parse_mode=None`, `caption_entities=None`, `disable_notification=None`, `reply_to_message_id=None`, `allow_sending_without_reply=None`, `reply_markup=None`, `timeout=None`, `api_kwargs=None`, `protect_content=None`)  

**copy**

Shortcut for:

```python
bot.copy_message(chat_id=chat_id,
                 from_chat_id=update.effective_message.chat_id,
                 message_id=update.effective_message.message_id,
                 *args,
                 **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.copy_message()`.

**Returns** On success, returns the MessageId of the sent message.

**Return type** `telegram.MessageId`

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

See `telegram.TelegramObject.de_json()`.

**delete**(`timeout=None`, `api_kwargs=None`)  

**delete**

Shortcut for:
bot.delete_message(chat_id=message.chat_id,
             message_id=message.message_id,
             *args,
             **kwargs)

For the documentation of the arguments, please see telegram.Bot.delete_message().

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

**Return type** bool

edit_caption(caption=None, reply_markup=None, timeout=None, parse_mode=None, api_kwargs=None, caption_entities=None)

Shortcut for:

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

For the documentation of the arguments, please see telegram.Bot.
edit_message_caption().

**Note:** You can only edit messages that the bot sent itself (i.e. of the bot.send_* family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

**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_live_location(latitude=None, longitude=None, location=None, reply_markup=None, timeout=None, api_kwargs=None, horizontal_accuracy=None, heading=None, proximity_alert_radius=None)

Shortcut for:

bot.edit_message_live_location(chat_id=message.chat_id,
                          message_id=message.message_id,
                          *args,
                          **kwargs)

For the documentation of the arguments, please see telegram.Bot.
edit_message_live_location().

**Note:** You can only edit messages that the bot sent itself (i.e. of the bot.send_* family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

**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_media(media=None, reply_markup=None, timeout=None, api_kwargs=None)

Shortcut for:
bot.edit_message_media(chat_id=message.chat_id,
message_id=message.message_id,
*args,
**kwargs)

For the documentation of the arguments, please see telegram.Bot.edit_message_media().

**Note:** You can only edit messages that the bot sent itself (i.e. of the bot.send_* family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

**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_reply_markup** *(reply_markup=None, timeout=None, api_kwargs=None)*

Shortcut for:

bot.edit_message_reply_markup(chat_id=message.chat_id,
message_id=message.message_id,
*args,
**kwargs)

For the documentation of the arguments, please see telegram.Bot.edit_message_reply_markup().

**Note:** You can only edit messages that the bot sent itself (i.e. of the bot.send_* family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

**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_text** *(text, parse_mode=None, disable_web_page_preview=None, reply_markup=None, timeout=None, api_kwargs=None, entities=None)*

Shortcut for:

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

For the documentation of the arguments, please see telegram.Bot.edit_message_text().

**Note:** You can only edit messages that the bot sent itself (i.e. of the bot.send_* family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

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

**Return type** telegram.Message
**property effective_attachment**

```python
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**

```python
forward(chat_id, disable_notification=None, timeout=None, api_kwargs=None, protect_content=None)

Shortcut for:
```
```python
bot.forward_message(chat_id=chat_id,
from_chat_id=update.effective_message.chat_id,
message_id=update.effective_message.message_id,
*args,
**kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.forward_message()`.

**Note:** Since the release of Bot API 5.5 it can be impossible to forward messages from some chats. Use the attributes `telegram.Message.has_protected_content` and `telegram.Chat.has_protected_content` to check this.

As a workaround, it is still possible to use `copy()`. However, this behaviour is undocumented and might be changed by Telegram.

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

**Return type** `telegram.Message`

**get_game_high_scores**

```python
get_game_high_scores(user_id, timeout=None, api_kwargs=None)

Shortcut for:
```
```python
bot.get_game_high_scores(chat_id=message.chat_id,
message_id=message.message_id,
*args,
**kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.get_game_high_scores()`.

**Note:** You can only edit messages that the bot sent itself (i.e. of the `bot.send_*` family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

**Returns** List[`telegram.GameHighScore`]

**property link**

Convenience property. If the chat of the message is not a private chat or normal group, returns a t.me link of the message.

**Type** `str`

**parse_caption_entities**

```python
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.

**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`

**Raises** `RuntimeError` – If the message has no caption.

**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

**Raises** RuntimeError – If the message has no text.

### pin

(disable_notification=None, timeout=None, api_kwargs=None)

Shortcut for:

```python
bot.pin_chat_message(chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs)
```

For the documentation of the arguments, please see **telegram.Bot.pin_chat_message()**.

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

**Return type** bool

### reply_animation

(animation, duration=None, width=None, height=None, thumb=None, caption=None, reply_to_message_id=None, reply_markup=None, timeout=20, api_kwargs=None, allow_sending_without_reply=None, caption_entities=None, filename=None, quote=None, protect_content=None)

Shortcut for:

```python
bot.send_animation(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see **telegram.Bot.send_animation()**.

**Parameters** **quote** (bool, optional) – If set to True, the animation 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

(audio, duration=None, performer=None, title=None, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, thumb=None, api_kwargs=None, allow_sending_without_reply=None, caption_entities=None, filename=None, quote=None, protect_content=None)

Shortcut for:

```python
bot.send_audio(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see **telegram.Bot.send_audio()**.

**Parameters** **quote** (bool, optional) – If set to True, the audio is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

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

**Return type** telegram.Message

### reply_chat_action

(action, timeout=None, api_kwargs=None)

Shortcut for:

```python
bot.send_chat_action(update.effective_message.chat_id, *args, **kwargs)
```
For the documentation of the arguments, please see `telegram.Bot.send_chat_action()`.

New in version 13.2.

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

**Return type** `bool`

`reply_contact(phone_number=None, first_name=None, last_name=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, contact=None, vcard=None, allow_sending_without_reply=None, quote=None, protect_content=None)`

Shortcut for:

```python
bot.send_contact(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_contact()`.

**Parameters**

- `quote` (`bool`, optional) – If set to `True`, the contact is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: `True` in group chats and `False` in private chats.

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

**Return type** `telegram.Message`

`reply_copy(from_chat_id, message_id, caption=None, parse_mode=None, caption_entities=None, disable_notification=None, reply_to_message_id=None, allow_sending_without_reply=None, reply_markup=None, timeout=None, api_kwargs=None, quote=None, protect_content=None)`

Shortcut for:

```python
bot.copy_message(chat_id=message.chat.id, from_chat_id=from_chat_id, message_id=message_id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.copy_message()`.

**Parameters**

- `quote` (`bool`, optional) – If set to `True`, the copy 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, returns the MessageId of the sent message.

**Return type** `telegram.MessageId`

`reply_dice(disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, emoji=None, api_kwargs=None, allow_sending_without_reply=None, quote=None, protect_content=None)`

Shortcut for:

```python
bot.send_dice(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_dice()`.

**Parameters**

- `quote` (`bool`, optional) – If set to `True`, the dice 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 (document, filename=None, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, thumb=None, api_kwargs=None, disable_content_type_detection=None, allow_sending_without_reply=None, caption_entities=None, quote=None, protect_content=None)

Shortcut for:

bot.send_document(update.effective_message.chat_id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.send_document().

Parameters quote (bool, optional) – If set to True, the document is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type telegram.Message

reply_game (game_short_name, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, api_kwargs=None, allow_sending_without_reply=None, quote=None, protect_content=None)

Shortcut for:

bot.send_game(update.effective_message.chat_id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.send_game().

Parameters quote (bool, optional) – If set to True, the game 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.

New in version 13.2.

Returns On success, instance representing the message posted.

Return type telegram.Message

reply_html (text, disable_web_page_preview=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, api_kwargs=None, allow_sending_without_reply=None, entities=None, quote=None, protect_content=None)

Shortcut for:

bot.send_message(update.effective_message.chat_id, *args, **kwargs, parse_mode=ParseMode.HTML)

Sends a message with HTML formatting.

For the documentation of the arguments, please see telegram.Bot.send_message().

Parameters 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.

Returns On success, instance representing the message posted.

Return type telegram.Message
reply_invoice(title, description, payload, provider_token, currency, prices, start_parameter=None, 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=None, reply_to_message_id=None, reply_markup=None, provider_data=None, send_phone_number_to_provider=None, send_email_to_provider=None, timeout=None, api_kwargs=None, allow_sending_without_reply=None, quote=None)

Shortcut for:

bot.send_invoice(update.effective_message.chat_id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.send_invoice().

**Warning:** As of API 5.2 start_parameter is an optional argument and therefore the order of the arguments had to be changed. Use keyword arguments to make sure that the arguments are passed correctly.

New in version 13.2.

Changed in version 13.5: As of Bot API 5.2, the parameter start_parameter is optional.

Parameters **quote** (bool, optional) – If set to True, the invoice is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type telegram.Message

reply_location(latitude=None, longitude=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, location=None, live_period=None, api_kwargs=None, horizontal_accuracy=None, heading=None, proximity_alert_radius=None, allow_sending_without_reply=None, quote=None, protect_content=None)

Shortcut for:

bot.send_location(update.effective_message.chat_id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.send_location().

Parameters **quote** (bool, optional) – If set to True, the location is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type telegram.Message

reply_markdown(text, disable_web_page_preview=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, api_kwargs=None, allow_sending_without_reply=None, entities=None, quote=None, protect_content=None)

Shortcut for:

bot.send_message(update.effective_message.chat_id, *args, **kwargs, parse_mode=ParseMode.MARKDOWN)
Sends a message with Markdown version 1 formatting.

For the documentation of the arguments, please see `telegram.Bot.send_message()`.

**Note:** `telegram.ParseMode.MARKDOWN` is a legacy mode, retained by Telegram for backward compatibility. You should use `reply_markdown_v2()` instead.

### Parameters
- **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.

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

### Return type
- `telegram.Message`

### `reply_markdown_v2`

Sends a message with markdown version 2 formatting.

For the documentation of the arguments, please see `telegram.Bot.send_message()`.

**Parameters**
- **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.

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

**Return type**
- `telegram.Message`

### `reply_media_group`

Shortcut for:

```python
bot.send_media_group(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_media_group()`.

**Parameters**
- **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.error.TelegramError`

### `reply_photo`

Shortcut for:

```python
bot.send_photo(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_photo()`.

**Parameters**
- **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.error.TelegramError`
For the documentation of the arguments, please see `telegram.Bot.send_photo()`.

**Parameters**
- `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`

---

```
bot.send_poll(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_poll()`.

**Parameters**
- `quote` (bool, optional) – If set to True, the poll 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`

---

```
bot.send_sticker(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_sticker()`.

**Parameters**
- `quote` (bool, optional) – If set to True, the sticker is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

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

**Return type** `telegram.Message`

---

```
bot.send_message(update.effective_message.chat_id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_message()`.

**Parameters**
- `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.

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

**Return type** `telegram.Message`
reply_venue(latitude=None, longitude=None, title=None, address=None, foursquare_id=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, venue=None, foursquare_type=None, allow_sending_without_reply=None, quote=None, protect_content=None)

Shortcut for:
bot.send_venue(update.effective_message.chat_id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.send_venue().

Parameters
quote (bool, optional) – If set to True, the venue is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type telegram.Message

reply_video(video, duration=None, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, width=None, height=None, parse_mode=None, supports_streaming=None, thumb=None, api_kwargs=None, allow_sending_without_reply=None, caption_entities=None, filename=None, quote=None, protect_content=None)

Shortcut for:
bot.send_video(update.effective_message.chat_id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.send_video().

Parameters
quote (bool, optional) – If set to True, the video is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type telegram.Message

reply_video_note(video_note, duration=None, length=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, thumb=None, api_kwargs=None, allow_sending_without_reply=None, filename=None, quote=None, protect_content=None)

Shortcut for:
bot.send_video_note(update.effective_message.chat_id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.send_video_note().

Parameters
quote (bool, optional) – If set to True, the video note 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(voice, duration=None, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, api_kwargs=None, allow_sending_without_reply=None, caption_entities=None, filename=None, quote=None, protect_content=None)

Shortcut for:
bot.send_voice(update.effective_message.chat_id, *args, **kwargs)

For the documentation of the arguments, please see telegram.Bot.send_voice().
Parameters `quote` (bool, optional) – If set to True, the voice note 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`

### set_game_score

`set_game_score(user_id, score, force=None, disable_edit_message=None, timeout=None, api_kwargs=None)`

Shortcut for:

```python
bot.set_game_score(chat_id=message.chat_id,
                   message_id=message.message_id,
                   *args,
                   **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.set_game_score()`.

**Note:** You can only edit messages that the bot sent itself (i.e. of the `bot.send_*` family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

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

Return type `telegram.Message`

### stop_live_location

`stop_live_location(reply_markup=None, timeout=None, api_kwargs=None)`

Shortcut for:

```python
bot.stop_message_live_location(chat_id=message.chat_id,
                                message_id=message.message_id,
                                *args,
                                **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.stop_message_live_location()`.

**Note:** You can only edit messages that the bot sent itself (i.e. of the `bot.send_*` family of methods) or channel posts, if the bot is an admin in that channel. However, this behaviour is undocumented and might be changed by Telegram.

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

Return type `telegram.Message`

### stop_poll

`stop_poll(reply_markup=None, timeout=None, api_kwargs=None)`

Shortcut for:

```python
bot.stop_poll(chat_id=message.chat_id,
              message_id=message.message_id,
              *args,
              **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.stop_poll()`.

Returns On success, the stopped Poll with the final results is returned.
Return type `telegram.Poll`

**property 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.

Changed in version 13.10: Spoiler entities are now formatted as HTML.

**Returns** Message text with entities formatted as HTML.

**Return type** `str`

**property text_html_urlled**

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.

Changed in version 13.10: Spoiler entities are now formatted as HTML.

**Returns** Message text with entities formatted as HTML.

**Return type** `str`

**property text_markdown**

Creates a Markdown-formatted string from the markup entities found in the message using `telegram.ParseMode.MARKDOWN`.

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.

**Note:** `telegram.ParseMode.MARKDOWN` is a legacy mode, retained by Telegram for backward compatibility. You should use `text_markdown_v2()` instead.

**Returns** Message text with entities formatted as Markdown.

**Return type** `str`

**Raises** `ValueError` – If the message contains underline, strikethrough, spoiler or nested entities.

**property text_markdown_urlled**

Creates a Markdown-formatted string from the markup entities found in the message using `telegram.ParseMode.MARKDOWN`.

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.

**Note:** `telegram.ParseMode.MARKDOWN` is a legacy mode, retained by Telegram for backward compatibility. You should use `text_markdown_v2_urlled()` instead.

**Returns** Message text with entities formatted as Markdown.

**Return type** `str`

**Raises** `ValueError` – If the message contains underline, strikethrough, spoiler or nested entities.

**property text_markdown_v2**

Creates a Markdown-formatted string from the markup entities found in the message using `telegram.ParseMode.MARKDOWN_V2`. 

---

**3.2. telegram package**

243
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.

Changed in version 13.10: Spoiler entities are now formatted as Markdown V2.

Returns Message text with entities formatted as Markdown.

Return type str

property text_markdown_v2_urled

Creates an Markdown-formatted string from the markup entities found in the message using telegram.ParseMode.MARKDOWN_V2.

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.

Changed in version 13.10: Spoiler entities are now formatted as Markdown V2.

Returns Message text with entities formatted as Markdown.

Return type str

to_dict()

See telegram.TelegramObject.to_dict().

unpin(timeout=None, api_kwargs=None)

Shortcut for:

```python
bot.unpin_chat_message(chat_id=message.chat_id,  
message_id=message.message_id,  
*args,  
**kwargs)
```

For the documentation of the arguments, please see telegram.Bot.unpin_chat_message().

Returns On success, True is returned.

Return type bool

### 3.2.50 telegram.MessageAutoDeleteTimerChanged

class telegram.MessageAutoDeleteTimerChanged (message_auto_delete_time,  
**kwargs)

Bases: telegram.base.TelegramObject

This object represents a service message about a change in auto-delete timer settings.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their message_auto_delete_time is equal.

New in version 13.4.

Parameters

- **message_auto_delete_time (int)** – New auto-delete time for messages in the chat.

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

message_auto_delete_time

New auto-delete time for messages in the chat.

Type int
3.2.51 telegram.MessageId

class telegram.MessageId(message_id, **kwargs)
    Bases: telegram.base.TelegramObject

This object represents a unique message identifier. Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their message_id is equal.

    message_id
        Unique message identifier

        Type int

3.2.52 telegram.MessageEntity

class telegram.MessageEntity(type, offset, length, url=None, user=None, language=None, **kwargs)
    Bases: telegram.base.TelegramObject

This object represents one special entity in a text message. For example, hashtags, usernames, URLs, etc. Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their type, offset and length are equal.

Parameters

* type (str) – Type of the entity. Currently, can be mention (@username), hashtag, bot_command, url, email, phone_number, bold (bold text), italic (italic text), strikethrough, spoiler (spoiler message), 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.

* language (str, optional) – For PRE only, the programming language of the entity text.

    type
        Type of the entity.

        Type str

    offset
        Offset in UTF-16 code units to the start of the entity.

        Type int

    length
        Length of the entity in UTF-16 code units.

        Type int

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

        Type str

    user
        Optional. The mentioned user.

        Type telegram.User
language
Optional. Programming language of the entity text.

Type str

ALL_TYPES: ClassVar[List[str]] = ['mention', 'hashtag', 'cashtag', 'phone_number', 'bot_command', 'url', 'email', 'bold', 'italic', 'code', 'pre', 'text_link', 'text_mention', 'underline', 'strikethrough', 'spoiler']

List of all the types

BOLD: ClassVar[str] = 'bold'

BOT_COMMAND: ClassVar[str] = 'bot_command'

CASHTAG: ClassVar[str] = 'cashtag'

CODE: ClassVar[str] = 'code'

EMAIL: ClassVar[str] = 'email'

HASHTAG: ClassVar[str] = 'hashtag'

ITALIC: ClassVar[str] = 'italic'

MENTION: ClassVar[str] = 'mention'

PHONE_NUMBER: ClassVar[str] = 'phone_number'

PRE: ClassVar[str] = 'pre'

SPOILER: ClassVar[str] = 'spoiler'

New in version 13.10.

STRIKETHROUGH: ClassVar[str] = 'strikethrough'

TEXT_LINK: ClassVar[str] = 'text_link'

TEXT_MENTION: ClassVar[str] = 'text_mention'

UNDERLINE: ClassVar[str] = 'underline'

URL: ClassVar[str] = 'url'

classmethod de_json(data, bot)
See telegram.TelegramObject.de_json().
3.2.53 telegram.ParseMode

class telegram.ParseMode

    Bases: object

    This object represents a Telegram Message Parse Modes.

    HTML: ClassVar[str] = 'HTML'
    telegram.constants.PARSEMODE_HTML

    MARKDOWN: ClassVar[str] = 'Markdown'
    telegram.constants.PARSEMODE_MARKDOWN

    Note: MARKDOWN is a legacy mode, retained by Telegram for backward compatibility. You should use MARKDOWN_V2 instead.

    MARKDOWN_V2: ClassVar[str] = 'MarkdownV2'
    telegram.constants.PARSEMODE_MARKDOWN_V2

3.2.54 telegram.PhotoSize

class telegram.PhotoSize(file_id, file_unique_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.

    Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their file_unique_id is equal.

    Parameters

    • file_id (str) – Identifier for this file, which can be used to download or reuse the file.

    • file_unique_id (str) – Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the 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.

    file_id

    Identifier for this file.

    Type str

    file_unique_id

    Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the file.

    Type str

    width

    Photo width.

    Type int

    height

    Photo height.
Type int

file_size
Optional. File size.
Type int

bot
Optional. The Bot to use for instance methods.
Type telegram.Bot

get_file(timeout=None, api_kwargs=None)
Convenience wrapper over telegram.Bot.get_file
For the documentation of the arguments, please see telegram.Bot.get_file().

Returns telegram.File

Raises telegram.error.TelegramError

3.2.55 telegram.Poll
class telegram.Poll(id, question, options, total_voter_count, is_closed, is_anonymous, type, allows_multiple_answers, correct_option_id=None, explanation=None, explanation_entities=None, open_period=None, close_date=None, **kwargs)
Bases: telegram.base.TelegramObject

This object contains information about a poll.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their id is equal.

id
Unique poll identifier.
Type str

question
Poll question, 1-300 characters.
Type str

options
List of poll options.
Type List[PollOption]

total_voter_count
Total number of users that voted in the poll.
Type int

is_closed
True, if the poll is closed.
Type bool

is_anonymous
True, if the poll is anonymous.
Type bool

type
Poll type, currently can be REGULAR or QUIZ.
Type str

allows_multiple_answers
True, if the poll allows multiple answers.
**Type** bool

**correct_option_id**
Optional. Identifier of the correct answer option.

**Type** int

**explanation**
Optional. Text that is shown when a user chooses an incorrect answer or taps on the lamp icon in a quiz-style poll.

**Type** str

**explanation_entities**
Optional. Special entities like usernames, URLs, bot commands, etc. that appear in the explanation.

**Type** List[telegram.MessageEntity]

**open_period**
Optional. Amount of time in seconds the poll will be active after creation.

**Type** int

**close_date**
Optional. Point in time when the poll will be automatically closed.

**Type** datetime.datetime

**Parameters**

- **id** (str) – Unique poll identifier.
- **question** (str) – Poll question, 1-300 characters.
- **options** (List[PollOption]) – List of poll options.
- **is_closed** (bool) – True, if the poll is closed.
- **is_anonymous** (bool) – True, if the poll is anonymous.
- **type** (str) – Poll type, currently can be **REGULAR** or **QUIZ**.
- **allows_multiple_answers** (bool) – True, if the poll allows multiple answers.
- **correct_option_id** (int, optional) – 0-based identifier of the correct answer option. Available only for polls in the quiz mode, which are closed, or was sent (not forwarded) by the bot or to the private chat with the bot.
- **explanation** (str, optional) – Text that is shown when a user chooses an incorrect answer or taps on the lamp icon in a quiz-style poll, 0-200 characters.
- **explanation_entities** (List[telegram.MessageEntity], optional) – Special entities like usernames, URLs, bot commands, etc. that appear in the explanation.
- **open_period** (int, optional) – Amount of time in seconds the poll will be active after creation.
- **close_date** (datetime.datetime, optional) – Point in time (Unix timestamp) when the poll will be automatically closed. Converted to datetime.datetime.

**MAX_OPTION_LENGTH**: ClassVar[int] = 100

**MAX_QUESTION_LENGTH**: ClassVar[int] = 300

**QUIZ**: ClassVar[str] = 'quiz'
**REGULAR**: `ClassVar[str] = 'regular'

telemetry.constants.POLL_REGULAR

classmethod de_json(data, bot)

See `telegram.TelegramObject.de_json()`.

parse_explanation_entities(types=None)

Returns a dict that maps `telegram.MessageEntity` to str. It contains entities from this polls explanation 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 `explanation_entities` attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See `parse_explanation_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_explanation_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`

**Raises**

- `RuntimeError` – If the poll has no explanation.

`to_dict()`

See `telegram.TelegramObject.to_dict()`.

### 3.2.56 `telegram.PollAnswer`

**class** `telegram.PollAnswer(poll_id, user, option_ids, **kwargs)`

**Bases**: `telegram.base.TelegramObject`

This object represents an answer of a user in a non-anonymous poll.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `poll_id`, `user` and `options_ids` are equal.

**poll_id**

Unique poll identifier.

**Type** `str`

**user**

The user, who changed the answer to the poll.
Type `telegram.User`

**option_ids**
Identifiers of answer options, chosen by the user.

Type `List[int]`

Parameters

- **poll_id**(str) – Unique poll identifier.
- **user**(telegram.User) – The user, who changed the answer to the poll.
- **option_ids**(List[int]) – 0-based identifiers of answer options, chosen by the user. May be empty if the user retracted their vote.

classmethod `de_json`(data, bot)

See `telegram.TelegramObject.de_json()`.

### 3.2.57 `telegram.PollOption`

**class** `telegram.PollOption`(text, voter_count, **kwargs)

Bases: `telegram.base.TelegramObject`

This object contains information about one answer option in a poll.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `text` and `voter_count` are equal.

Parameters

- **text**(str) – Option text, 1-100 characters.
- **voter_count**(int) – Number of users that voted for this option.

`text`

Option text, 1-100 characters.

Type `str`

`voter_count`

Number of users that voted for this option.

Type `int`

**MAX_LENGTH**: `ClassVar[int] = 100`

`telegram.constants.MAX_POLL_OPTION_LENGTH`

### 3.2.58 `telegram.ProximityAlertTriggered`

**class** `telegram.ProximityAlertTriggered`(traveler, watcher, distance, **kwargs)

Bases: `telegram.base.TelegramObject`

This object represents the content of a service message, sent whenever a user in the chat triggers a proximity alert set by another user.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `traveler`, `watcher` and `distance` are equal.

Parameters

- **traveler**(telegram.User) – User that triggered the alert
- **watcher**(telegram.User) – User that set the alert
- **distance**(int) – The distance between the users
traveler
User that triggered the alert
Type telegram.User
watcher
User that set the alert
Type telegram.User
distance
The distance between the users
Type int
classmethod de_json (data, bot)
See telegram.TelegramObject.de_json().

3.2.59 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).

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.

Note: User will not be able to summon this keyboard; if you want to hide the keyboard from sight but keep it accessible, use telegram.ReplyKeyboardMarkup.one_time_keyboard.

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 telegram.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.

remove_keyboard
Requests clients to remove the custom keyboard.
Type True

selective
Optional. Use this parameter if you want to remove the keyboard for specific users only.
Type bool
3.2.60 telegram.ReplyKeyboardMarkup

class telegram.ReplyKeyboardMarkup(keyboard, resize_keyboard=False, one_time_keyboard=False, selective=False, input_field_placeholder=None, **kwargs)

Bases: telegram.replymarkup.ReplyMarkup

This object represents a custom keyboard with reply options.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their size of keyboard and all the buttons are equal.

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.

- **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 telegram.Message object.

  2) If the bot’s message is a reply (has reply_to_message_id), sender of the original message.

  Defaults to False.

- **input_field_placeholder** (str, optional) – The placeholder to be shown in the input field when the keyboard is active; 1-64 characters.

  New in version 13.7.

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

keyboard

Array of button rows.

Type List[List[telegram.KeyboardButton | str]]

resize_keyboard

Optional. Requests clients to resize the keyboard.

Type bool

one_time_keyboard

Optional. Requests clients to hide the keyboard as soon as it’s been used.

Type bool

selective

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

**input_field_placeholder**
Optional. The placeholder shown in the input field when the reply is active.
New in version 13.7.
Type `str`

**classmethod from_button**
`button`, `resize_keyboard=False`, `one_time_keyboard=False`, `selective=False`, `input_field_placeholder=None`, `**kwargs`)
Shortcut for:
```
ReplyKeyboardMarkup([[button]], **kwargs)
```

Return a 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.

- **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`.

- **input_field_placeholder** *(str)* – Optional. The placeholder shown in the input field when the reply is active.

  New in version 13.7.

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

**classmethod from_column**
`button_column`, `resize_keyboard=False`, `one_time_keyboard=False`, `selective=False`, `input_field_placeholder=None`, `**kwargs`)
Shortcut for:
```
ReplyKeyboardMarkup([[button] for button in button_column], **kwargs)
```

Return a ReplyKeyboardMarkup from a single column of KeyboardButtons.

**Parameters**


- **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.
- **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.

- **input_field_placeholder** (str) – Optional. The placeholder shown in the input field when the reply is active.

    New in version 13.7.

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

    **classmethod from_row** (button_row, resize_keyboard=False, one_time_keyboard=False, selective=False, input_field_placeholder=None, **kwargs)

    Shortcut for:

    ```python
    ReplyKeyboardMarkup([button_row], **kwargs)
    ```

    Return a 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.

    - **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.

    - **input_field_placeholder** (str) – Optional. The placeholder shown in the input field when the reply is active.

      New in version 13.7.

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

    **to_dict()**

    See `telegram.TelegramObject.to_dict()`.
3.2.61 telegram.ReplyMarkup

class telegram.ReplyMarkup
    Bases: telegram.base.TelegramObject
    Base class for Telegram ReplyMarkup Objects.

3.2.62 telegram.TelegramObject

class telegram.TelegramObject
    Bases: object
    Base class for most Telegram objects.
    classmethod de_json(data, bot)
        Converts JSON data to a Telegram object.
        Parameters
        • data (Dict[str,...]) – The JSON data.
        • bot (telegram.Bot) – The bot associated with this object.
        Returns The Telegram object.
    classmethod de_list(data, bot)
        Converts JSON data to a list of Telegram objects.
        Parameters
        • data (Dict[str,...]) – The JSON data.
        • bot (telegram.Bot) – The bot associated with these objects.
        Returns A list of Telegram objects.
    to_dict()
        Gives representation of object as dict.
        Returns dict
    to_json()
        Gives a JSON representation of object.
        Returns str

3.2.63 telegram.Update

class telegram.Update(update_id, message=None, edited_message=None, channel_post=None, edited_channel_post=None, inline_query=None, chosen_inline_result=None, callback_query=None, shipping_query=None, pre_checkout_query=None, poll=None, poll_answer=None, my_chat_member=None, chat_member=None, chat_join_request=None, **_kwargs)
    Bases: telegram.base.TelegramObject
    This object represents an incoming update.
    Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their update_id is equal.

    Note: At most one of the optional parameters can be present in any given update.
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. If there are no new updates for at least a week, then identifier of the next update will be chosen randomly instead of sequentially.

- **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.

- **poll** (telegram.Poll, optional) – New poll state. Bots receive only updates about stopped polls and polls, which are sent by the bot.

- **poll_answer** (telegram.PollAnswer, optional) – A user changed their answer in a non-anonymous poll. Bots receive new votes only in polls that were sent by the bot itself.

- **my_chat_member** (telegram.ChatMemberUpdated, optional) – The bot’s chat member status was updated in a chat. For private chats, this update is received only when the bot is blocked or unblocked by the user.

  New in version 13.4.

- **chat_member** (telegram.ChatMemberUpdated, optional) – A chat member’s status was updated in a chat. The bot must be an administrator in the chat and must explicitly specify 'chat_member' in the list of 'allowed_updates' to receive these updates (see telegram.Bot.get_updates(), telegram.Bot.set_webhook(), telegram.ext.Updater.start_polling() and telegram.ext.Updater.start_webhook()).

  New in version 13.4.

- **chat_join_request** (telegram.ChatJoinRequest, optional) – A request to join the chat has been sent. The bot must have the telegram.ChatPermissions.can_invite_users administrator right in the chat to receive these updates.

  New in version 13.8.

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

3.2. telegram package 257
update_id
   The update’s unique identifier.
   Type int

message
   Optional. New incoming message.
   Type telegram.Message

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

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

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

inline_query
   Optional. New incoming inline query.
   Type telegram.InlineQuery

chosen_inline_result
   Optional. The result of an inline query that was chosen by a user.
   Type telegram.ChosenInlineResult

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

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

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

poll
   Optional. New poll state. Bots receive only updates about stopped polls and polls, which are sent by the bot.
   Type telegram.Poll

poll_answer
   Optional. A user changed their answer in a non-anonymous poll. Bots receive new votes only in polls that were sent by the bot itself.
   Type telegram.PollAnswer

my_chat_member
   Optional. The bot’s chat member status was updated in a chat. For private chats, this update is received only when the bot is blocked or unblocked by the user.
   New in version 13.4.
   Type telegram.ChatMemberUpdated
chat_member
Optional. A chat member's status was updated in a chat. The bot must be an administrator in the chat and must explicitly specify 'chat_member' in the list of 'allowed_updates' to receive these updates (see telegram.Bot.get_updates(), telegram.Bot.set_webhook(), telegram.ext.Updater.start_polling() and telegram.ext.Updater.start_webhook()).

New in version 13.4.

Type telegram.ChatMemberUpdated

chat_join_request
Optional. A request to join the chat has been sent. The bot must have the 'can_invite_users' administrator right in the chat to receive these updates.

New in version 13.8.

Type telegram.ChatJoinRequest

ALL_TYPES = ['message', 'edited_message', 'channel_post', 'edited_channel_post', 'inline_query', 'chosen_inline_result', ...
'shipping_query', 'pre_checkout_query', 'poll', 'poll_answer', 'my_chat_member', 'chat_member', 'chat_join_request']

telegram.constants.UPDATE_ALL_TYPES

New in version 13.5.

CALLBACK_QUERY = 'callback_query'

telegram.constants.UPDATE_CALLBACK_QUERY

New in version 13.5.

CHANNEL_POST = 'channel_post'

telegram.constants.UPDATE_CHANNEL_POST

New in version 13.5.

CHAT_JOIN_REQUEST = 'chat_join_request'

telegram.constants.UPDATE_CHAT_JOIN_REQUEST

New in version 13.8.

CHAT_MEMBER = 'chat_member'

telegram.constants.UPDATE_CHAT_MEMBER

New in version 13.5.

CHOSEN_INLINE_RESULT = 'chosen_inline_result'

telegram.constants.UPDATE_CHOSEN_INLINE_RESULT

New in version 13.5.

EDITED_CHANNEL_POST = 'edited_channel_post'

telegram.constants.UPDATE_EDITED_CHANNEL_POST

New in version 13.5.

EDITED_MESSAGE = 'edited_message'

telegram.constants.UPDATE_EDITED_MESSAGE

New in version 13.5.

INLINE_QUERY = 'inline_query'

telegram.constants.UPDATE_INLINE_QUERY

New in version 13.5.

MESSAGE = 'message'

telegram.constants.UPDATE_MESSAGE

New in version 13.5.
MY_CHAT_MEMBER = 'my_chat_member'
    telegram.constants.UPDATE_MY_CHAT_MEMBER
    New in version 13.5.

POLL = 'poll'
    telegram.constants.UPDATE_POLL
    New in version 13.5.

POLL_ANSWER = 'poll_answer'
    telegram.constants.UPDATE_POLL_ANSWER
    New in version 13.5.

PRE_CHECKOUT_QUERY = 'pre_checkout_query'
    telegram.constants.UPDATE_PRE_CHECKOUT_QUERY
    New in version 13.5.

SHIPPING_QUERY = 'shipping_query'
    telegram.constants.UPDATE_SHIPPING_QUERY
    New in version 13.5.

classmethod de_json(data, bot)
    See telegram.TelegramObject.de_json().

@property effective_chat
    The chat that this update was sent in, no matter what kind of update this is. Will be None
    for inline_query, chosen_inline_result, callback_query from inline messages,
    shipping_query, pre_checkout_query, poll and poll_answer.
    
    Type telegram.Chat

@property effective_message
    The message included in this update, no matter what kind of update this is. Will be None
    for inline_query, chosen_inline_result, callback_query from inline messages,
    shipping_query, pre_checkout_query, poll, poll_answer, my_chat_member,
    chat_layer as well as chat_join_request in case the bot is missing the telegram.
    ChatPermissions.can_invite_users administrator right in the chat.
    
    Type telegram.Message

@property effective_user
    The user that sent this update, no matter what kind of update this is. Will be None for
    channel_post and poll.
    
    Type telegram.User

3.2.64 telegram.User

class telegram.User
    id, first_name, is_bot, last_name=None, user-
    name=None, language_code=None, can_join_groups=None,
    can_read_all_group_messages=None, supports_inline_queries=None,
    bot=None, **kwargs

    Bases: telegram.base.TelegramObject

    This object represents a Telegram user or bot.
    
    Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if
    their id is equal.
    
    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 bots first name.
• `last_name` *(str, optional)* – User’s or bots last name.
• `username` *(str, optional)* – User’s or bots username.
• `language_code` *(str, optional)* – IETF language tag of the user’s language.
• `can_join_groups` *(str, optional)* – True, if the bot can be invited to groups. Returned only in `telegram.Bot.get_me` requests.
• `can_read_all_group_messages` *(str, optional)* – True, if privacy mode is disabled for the bot. Returned only in `telegram.Bot.get_me` requests.
• `supports_inline_queries` *(str, optional)* – True, if the bot supports inline queries. Returned only in `telegram.Bot.get_me` requests.
• `bot` *(telegram.Bot, optional)* – The Bot to use for instance methods.

`id`
Unique identifier for this user or bot.

    Type  int

`is_bot`
True, if this user is a bot.

    Type  bool

`first_name`
User’s or bot’s first name.

    Type  str

`last_name`
Optional. User’s or bot’s last name.

    Type  str

`username`
Optional. User’s or bot’s username.

    Type  str

`language_code`
Optional. IETF language tag of the user’s language.

    Type  str

`can_join_groups`
Optional. True, if the bot can be invited to groups. Returned only in `telegram.Bot.get_me` requests.

    Type  str

`can_read_all_group_messages`
Optional. True, if privacy mode is disabled for the bot. Returned only in `telegram.Bot.get_me` requests.

    Type  str

`supports_inline_queries`
Optional. True, if the bot supports inline queries. Returned only in `telegram.Bot.get_me` requests.

    Type  str

`bot`
Optional. The Bot to use for instance methods.

    Type  telegram.Bot
approve_join_request (chat_id, timeout=None, api_kwargs=None)

Shortcut for:
```
bot.approve_chat_join_request(user_id=update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.approve_chat_join_request()`.

New in version 13.8.

Returns On success, True is returned.

Return type `bool`

copy_message (chat_id, message_id, caption=None, parse_mode=None, caption_entities=None, disable_notification=None, reply_to_message_id=None, allow_sending_without_reply=None, reply_markup=None, timeout=None, api_kwargs=None, protect_content=None)

Shortcut for:
```
bot.copy_message(from_chat_id=update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.copy_message()`.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

decline_join_request (chat_id, timeout=None, api_kwargs=None)

Shortcut for:
```
bot.decline_chat_join_request(user_id=update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.decline_chat_join_request()`.

New in version 13.8.

Returns On success, True is returned.

Return type `bool`

property full_name

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

Type `str`

get_profile_photos (offset=None, limit=100, timeout=None, api_kwargs=None)

Shortcut for:
```
bot.get_user_profile_photos(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.get_user_profile_photos()`.

property link

Convenience property. If `username` is available, returns a t.me link of the user.

Type `str`

mention_button (name=None)

Shortcut for:
```
InlineKeyboardButton(text=name, url=f"tg://user?id={update.effective_user.id}"")
```
New in version 13.9.

**Parameters**

- **name** *(str)* – The name used as a link for the user. Defaults to `full_name`.

**Returns**

- InlineButton with url set to the user mention

**Return type** `telegram.InlineKeyboardButton`

**mention_html** *(name=None)*

- **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)*

- **name** *(str)* – The name used as a link for the user. Defaults to `full_name`.

**Returns**

- The inline mention for the user as markdown (version 1).

**Return type** `str`

**mention_markdown_v2** *(name=None)*

- **name** *(str)* – The name used as a link for the user. Defaults to `full_name`.

**Returns**

- The inline mention for the user as markdown (version 2).

**Return type** `str`

**property name**

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

**Type** `str`

**pin_message** *(message_id, disable_notification=None, timeout=None, api_kwargs=None)*

- **message_id**
- **disable_notification** *(None)*
- **timeout** *(None)*
- **api_kwargs** *(None)*

**Shortcut for:**

```python
bot.pin_chat_message(chat_id=update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.pin_chat_message()`.

**Returns**

- On success, `True` is returned.

**Return type** `bool`

**send_action** *(action, timeout=None, api_kwargs=None)*

- **action**
- **timeout** *(None)*
- **api_kwargs** *(None)*

**Alias for** `send_chat_action`

**send_animation** *(animation, duration=None, width=None, height=None, thumb=None, caption=None, parse_mode=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, api_kwargs=None, allow_sending_without_reply=None, caption_entities=None, filename=None, protect_content=None)*

**Shortcut for:**

```python
bot.send_animation(update.effective_user.id, *args, **kwargs)
```
send_audio(audio, duration=None, performer=None, title=None, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, thumb=None, api_kwargs=None, allow_sending_without_reply=None, caption_entities=None, filename=None, protect_content=None)

Shortcut for:

bot.send_audio(update.effective_user.id, *args, **kwargs)

For the documentation of the arguments, please see `telegram.Bot.send_audio()`.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

send_chat_action(action, timeout=None, api_kwargs=None)

Shortcut for:

bot.send_chat_action(update.effective_user.id, *args, **kwargs)

For the documentation of the arguments, please see `telegram.Bot.send_chat_action()`.

Returns On success.

Return type `True`

send_contact(phone_number=None, first_name=None, last_name=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, contact=None, vcard=None, api_kwargs=None, allow_sending_without_reply=None, protect_content=None)

Shortcut for:

bot.send_contact(update.effective_user.id, *args, **kwargs)

For the documentation of the arguments, please see `telegram.Bot.send_contact()`.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

send_copy(from_chat_id, message_id, caption=None, parse_mode=None, caption_entities=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, api_kwargs=None, allow_sending_without_reply=None, protect_content=None)

Shortcut for:

bot.copy_message(chat_id=update.effective_user.id, *args, **kwargs)

For the documentation of the arguments, please see `telegram.Bot.copy_message()`.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

send_dice(disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, emoji=None, api_kwargs=None, allow_sending_without_reply=None, protect_content=None)

Shortcut for:

bot.send_dice(update.effective_user.id, *args, **kwargs)

For the documentation of the arguments, please see `telegram.Bot.send_dice()`.
Returns On success, instance representing the message posted.

Return type telegram.Message

send_document (document, filename=None, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, thumb=None, api_kwargs=None, disable_content_type_detection=None, allow_sending_without_reply=None, caption_entities=None, protect_content=None)

Shortcut for:
```
bot.send_document(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see telegram.Bot.send_document().

Returns On success, instance representing the message posted.

Return type telegram.Message

send_game (game_short_name, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, api_kwargs=None, allow_sending_without_reply=None, protect_content=None)

Shortcut for:
```
bot.send_game(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see telegram.Bot.send_game().

Returns On success, instance representing the message posted.

Return type telegram.Message

send_invoice (title, description, payload, provider_token, currency, prices, start_parameter=None, 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=None, reply_to_message_id=None, reply_markup=None, provider_data=None, send_phone_number_to_provider=None, send_email_to_provider=None, timeout=None, api_kwargs=None, max_tip_amount=None, suggested_tip_amounts=None, protect_content=None)

Shortcut for:
```
bot.send_invoice(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see telegram.Bot.send_invoice().

**Warning:** As of API 5.2 start_parameter is an optional argument and therefore the order of the arguments had to be changed. Use keyword arguments to make sure that the arguments are passed correctly.

Changed in version 13.5: As of Bot API 5.2, the parameter start_parameter is optional.

Returns On success, instance representing the message posted.

Return type telegram.Message

send_location (latitude=None, longitude=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, location=None, live_period=None, api_kwargs=None, horizontal_accuracy=None, heading=None, proximity_alert_radius=None, allow_sending_without_reply=None, protect_content=None)

Shortcut for:
bot.send_location(update.effective_user.id, *args, **kwargs)

For the documentation of the arguments, please see `telegram.Bot.send_location()`.

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

**Return type** `telegram.Message`

**send_media_group** (`media`, `disable_notification=None`, `reply_to_message_id=None`, `timeout=20`, `api_kwargs=None`, `allow_sending_without_reply=None`, `protect_content=None`)

Shortcut for:

bot.send_media_group(update.effective_user.id, *args, **kwargs)

For the documentation of the arguments, please see `telegram.Bot.send_media_group()`.

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

**Return type** `List[telegram.Message]`

**send_message** (`text`, `parse_mode=None`, `disable_web_page_preview=None`, `disable_notification=None`, `reply_to_message_id=None`, `reply_markup=None`, `timeout=None`, `api_kwargs=None`, `allow_sending_without_reply=None`, `entities=None`, `protect_content=None`)

Shortcut for:

bot.send_message(update.effective_user.id, *args, **kwargs)

For the documentation of the arguments, please see `telegram.Bot.send_message()`.

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

**Return type** `telegram.Message`

**send_photo** (`photo`, `caption=None`, `disable_notification=None`, `reply_to_message_id=None`, `reply_markup=None`, `timeout=20`, `parse_mode=None`, `api_kwargs=None`, `allow_sending_without_reply=None`, `caption_entities=None`, `filename=None`, `protect_content=None`)

Shortcut for:

bot.send_photo(update.effective_user.id, *args, **kwargs)

For the documentation of the arguments, please see `telegram.Bot.send_photo()`.

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

**Return type** `telegram.Message`

**send_poll** (`question`, `options`, `is_anonymous=True`, `type='regular'`, `allows_multiple_answers=False`, `correct_option_id=None`, `is_closed=None`, `disable_notification=None`, `reply_to_message_id=None`, `reply_markup=None`, `timeout=None`, `explanation=None`, `explanation_parse_mode=None`, `open_period=None`, `close_date=None`, `api_kwargs=None`, `allow_sending_without_reply=None`, `explanation_entities=None`, `protect_content=None`)

Shortcut for:

bot.send_poll(update.effective_user.id, *args, **kwargs)

For the documentation of the arguments, please see `telegram.Bot.send_poll()`.

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

**Return type** `telegram.Message`
send_sticker(sticker, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, api_kwargs=None, allow_sending_without_reply=None, protect_content=None)

Shortcut for:

```python
bot.send_sticker(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_sticker()`.

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

**Return type** `telegram.Message`

send_venue(latitude=None, longitude=None, title=None, address=None, foursquare_id=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=None, venue=None, foursquare_type=None, google_place_id=None, google_place_type=None, allow_sending_without_reply=None, protect_content=None)

Shortcut for:

```python
bot.send_venue(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_venue()`.

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

**Return type** `telegram.Message`

send_video(video, duration=None, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, width=None, height=None, parse_mode=None, supports_streaming=None, thumb=None, api_kwargs=None, allow_sending_without_reply=None, caption_entities=None, filename=None, protect_content=None)

Shortcut for:

```python
bot.send_video(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_video()`.

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

**Return type** `telegram.Message`

send_video_note(video_note, duration=None, length=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, thumb=None, api_kwargs=None, allow_sending_without_reply=None, filename=None, protect_content=None)

Shortcut for:

```python
bot.send_video_note(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_video_note()`.

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

**Return type** `telegram.Message`

send_voice(voice, duration=None, caption=None, disable_notification=None, reply_to_message_id=None, reply_markup=None, timeout=20, parse_mode=None, api_kwargs=None, allow_sending_without_reply=None, caption_entities=None, filename=None, protect_content=None)

Shortcut for:

```python
bot.send_voice(update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.send_voice()`.
**unpin_all_messages** *(timeout=None, api_kwargs=None)*

Shortcut for:

```python
bot.unpin_all_chat_messages(chat_id=update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.unpin_all_chat_messages()`.

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

**Return type** `bool`

**unpin_message** *(timeout=None, api_kwargs=None, message_id=None)*

Shortcut for:

```python
bot.unpin_chat_message(chat_id=update.effective_user.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.unpin_chat_message()`.

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

**Return type** `bool`

### 3.2.65 telegram.UserProfilePhotos

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

**Bases:** `telegram.base.TelegramObject`

This object represents a user’s profile pictures.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `total_count` and `photos` are equal.

**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).

**total_count**

Total number of profile pictures.

**Type** `int`

**photos**

Requested profile pictures.

**Type** `List[List[telegram.PhotoSize]]`

**classmethod de_json**(data, bot)

See `telegram.Bot.de_json()`.

**to_dict**

See `telegram.Bot.de_json()`.
3.2.66 telegram.Venue

class telegram.Venue

Bases: telegram.base.TelegramObject

This object represents a venue.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their location and title are equal.

Note: Foursquare details and Google Place details are mutually exclusive. However, this behaviour is undocumented and might be changed by Telegram.

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”.)
- **google_place_id** *(str, optional)* – Google Places identifier of the venue.
- **google_place_type** *(str, optional)* – Google Places type of the venue. (See supported types.)
- **_kwargs** *(dict)* – Arbitrary keyword arguments.

location

Venue location.

Type telegram.Location

title

Name of the venue.

Type str

address

Address of the venue.

Type str

foursquare_id

Optional. Foursquare identifier of the venue.

Type str

foursquare_type

Optional. Foursquare type of the venue.

Type str

google_place_id

Optional. Google Places identifier of the venue.

Type str

google_place_type

Optional. Google Places type of the venue.

Type str
classmethod de_json(data, bot)
    See telegram.TelegramObject.de_json().

3.2.67 telegram.Video

class telegram.Video(file_id, file_unique_id, width, height, duration, thumb=None, mime_type=None, file_size=None, bot=None, file_name=None, **kwargs)
    Bases: telegram.base.TelegramObject

This object represents a video file.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their file_unique_id is equal.

Parameters

- **file_id**(str) – Identifier for this file, which can be used to download or reuse the file.
- **file_unique_id**(str) – Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the 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.
- **file_name**(str, optional) – Original filename as defined by sender.
- **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.

**file_id**
    Identifier for this file.

    Type: str

**file_unique_id**
    Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the file.

    Type: str

**width**
    Video width as defined by sender.

    Type: int

**height**
    Video height as defined by sender.

    Type: int

**duration**
    Duration of the video in seconds as defined by sender.

    Type: int

**thumb**
    Optional. Video thumbnail.
Type `telegram.PhotoSize`

**file_name**
Optional. Original filename as defined by sender.
Type `str`

**mime_type**
Optional. Mime type of a file as defined by sender.
Type `str`

**file_size**
Optional. File size.
Type `int`

**bot**
Optional. The Bot to use for instance methods.
Type `telegram.Bot`

**classmethod de_json**(data, bot)
See `telegram.TelegramObject.de_json()`.

**get_file**(timeout=None, api_kwargs=None)
 Convenience wrapper over `telegram.Bot.get_file`
For the documentation of the arguments, please see `telegram.Bot.get_file()`.
Returns `telegram.File`

Raises `telegram.error.TelegramError` –

### 3.2.68 telegram.VideoNote

**class telegram.VideoNote**(file_id, file_unique_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).

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `file_unique_id` is equal.

**Parameters**

- **file_id**(str) – Identifier for this file, which can be used to download or reuse the file.
- **file_unique_id**(str) – Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the file.
- **length**(int) – Video width and height (diameter of the video message) 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.

**file_id**
Identifier for this file.
Type `str`
**file_unique_id**

Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the file.

*Type str*

**length**

Video width and height as defined by sender.

*Type int*

**duration**

Duration of the video in seconds as defined by sender.

*Type int*

**thumb**

Optional. Video thumbnail.

*Type telegram.PhotoSize*

**file_size**

Optional. File size.

*Type int*

**bot**

Optional. The Bot to use for instance methods.

*Type telegram.Bot*

**classmethod de_json**(data, bot)

See `telegram.TelegramObject.de_json()`.

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

Convenience wrapper over `telegram.Bot.get_file`

For the documentation of the arguments, please see `telegram.Bot.get_file()`.

*Returns* telegram.File

*Raises* telegram.error.TelegramError

### 3.2.69 telegram.Voice

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

*Bases*: telegram.base.TelegramObject

This object represents a voice note.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `file_unique_id` is equal.

**Parameters**

- **file_id**(str) – Identifier for this file, which can be used to download or reuse the file.
- **file_unique_id**(str) – Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the 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.

file_id

Identifier for this file.

Type: str

file_unique_id

Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the file.

Type: str

duration

Duration of the audio in seconds as defined by sender.

Type: int

mime_type

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

Type: str

file_size

Optional. File size.

Type: int

bot

Optional. The Bot to use for instance methods.

Type: telegram.Bot

class telegram.VoiceChatStarted(**kwargs)

Bases: telegram.base.TelegramObject

This object represents a service message about a voice chat started in the chat. Currently holds no information.

New in version 13.4.

Parameters

- **duration** (int) – Voice chat duration in seconds.

class telegram.VoiceChatEnded(duration, **kwargs)

Bases: telegram.base.TelegramObject

This object represents a service message about a voice chat ended in the chat.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their **duration** are equal.

New in version 13.4.
**kwargs (dict) – Arbitrary keyword arguments.

duration
   Voice chat duration in seconds.
   Type int

### 3.2.72 telegram.VoiceChatScheduled

class telegram.VoiceChatScheduled (start_date, **_kwargs)  
Bases: telegram.base.TelegramObject

This object represents a service message about a voice chat scheduled in the chat.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their start_date are equal.

**Parameters**

- **start_date** (datetime.datetime) – Point in time (Unix timestamp) when the voice chat is supposed to be started by a chat administrator

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

**start_date**
   Point in time (Unix timestamp) when the voice chat is supposed to be started by a chat administrator
   Type datetime.datetime

**classmethod de_json**(data, bot)  
See telegram.TelegramObject.de_json().

to_dict()
   See telegram.TelegramObject.to_dict().

### 3.2.73 telegram.VoiceChatParticipantsInvited

class telegram.VoiceChatParticipantsInvited (users, **_kwargs)  
Bases: telegram.base.TelegramObject

This object represents a service message about new members invited to a voice chat.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their users are equal.

New in version 13.4.

**Parameters**

- **users** (List[telegram.User]) – New members that were invited to the voice chat.

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

users
   New members that were invited to the voice chat.
   Type List[telegram.User]

**classmethod de_json**(data, bot)  
See telegram.TelegramObject.de_json().

to_dict()
   See telegram.TelegramObject.to_dict().
### 3.2.74 `telegram.WebhookInfo`

```python
class telegram.WebhookInfo:
    url, has_custom_certificate, pending_update_count,
    last_error_date, last_error_message, max_connections, ip_address, **kwargs

Bases: telegram.base.TelegramObject
```

This object represents a Telegram WebhookInfo. Contains information about the current status of a webhook.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `url`, `has_custom_certificate`, `pending_update_count`, `ip_address`, `last_error_date`, `last_error_message`, `max_connections` and `allowed_updates` are equal.

**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.
- `ip_address (str, optional)` – Currently used webhook IP address.
- `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, except `telegram.Update.chat_member`.

**url**

Webhook URL.

Type `str`

**has_custom_certificate**

If a custom certificate was provided for webhook.

Type `bool`

**pending_update_count**

Number of updates awaiting delivery.

Type `int`

**ip_address**

Optional. Currently used webhook IP address.

Type `str`

**last_error_date**

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

Type `int`

**last_error_message**

Optional. Error message in human-readable format.

Type `str`
max_connections
Optional. Maximum allowed number of simultaneous HTTPS connections.
Type int

allowed_updates
Optional. A list of update types the bot is subscribed to. Defaults to all update types, except
telegram.Update.chat_member.
Type List[str]

3.2.75 Stickers

telegram.Sticker
class telegram.Sticker(file_id, file_unique_id, width, height, is_animated, is_video,
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.
Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if
their file_unique_id is equal.

Note: As of v13.11 is_video is a required argument and therefore the order of the arguments had to be
changed. Use keyword arguments to make sure that the arguments are passed correctly.

Parameters

- file_id (str) – Identifier for this file, which can be used to download or reuse the file.
- file_unique_id (str) – Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the file.
- width (int) – Sticker width.
- height (int) – Sticker height.
- is_animated (bool) – True, if the sticker is animated.
- is_video (bool) – True, if the sticker is a video sticker.
  New in version 13.11.
- 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 (telegram.Bot, optional) – The Bot to use for instance methods.
- (obj (**kwargs) – dict): Arbitrary keyword arguments.

file_id
Identifier for this file.
Type str
file_unique_id
    Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t
    be used to download or reuse the file.
    
    Type str

width
    Sticker width.
    
    Type int

height
    Sticker height.
    
    Type int

is_animated
    True, if the sticker is animated.
    
    Type bool

is_video
    True, if the sticker is a video sticker.
    
    New in version 13.11.
    
    Type bool

thumb
    Optional. Sticker thumbnail in the .webp or .jpg format.
    
    Type telegram.PhotoSize

emoji
    Optional. Emoji associated with the sticker.
    
    Type str

set_name
    Optional. Name of the sticker set to which the sticker belongs.
    
    Type str

mask_position
    Optional. For mask stickers, the position where the mask should be placed.
    
    Type telegram.MaskPosition

file_size
    Optional. File size.
    
    Type int

bot
    Optional. The Bot to use for instance methods.
    
    Type telegram.Bot

classmethod de_json(data, bot)
    See telegram.TelegramObject.de_json().

get_file(timeout=None, api_kwargs=None)
    Convenience wrapper over telegram.Bot.get_file
    
    For the documentation of the arguments, please see telegram.Bot.get_file().
    
    Returns telegram.File
    
    Raises telegram.error.TelegramError –
`telegram.StickerSet` documentation:

```python
class telegram.StickerSet(name, title, is_animated, contains_masks, stickers, is_video, thumb=None, **kwargs)
```

This object represents a sticker set.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `name` is equal.

**Note:** As of v13.11 `is_video` is a required argument and therefore the order of the arguments had to be changed. Use keyword arguments to make sure that the arguments are passed correctly.

**Parameters**

- `name` (str) – Sticker set name.
- `title` (str) – Sticker set title.
- `is_animated` (bool) – True, if the sticker set contains animated stickers.
- `is_video` (bool) – True, if the sticker set contains video stickers.
  
  New in version 13.11.
- `contains_masks` (bool) – True, if the sticker set contains masks.
- `stickers` (List[telegram.Sticker]) – List of all set stickers.
- `thumb` (telegram.PhotoSize, optional) – Sticker set thumbnail in the .WEBP, .TGS, or .WEBM format.

**name**

Sticker set name.

**Type** str

**title**

Sticker set title.

**Type** str

**is_animated**

True, if the sticker set contains animated stickers.

**Type** bool

**is_video**

True, if the sticker set contains video stickers.

New in version 13.11.

**Type** bool

**contains_masks**

True, if the sticker set contains masks.

**Type** bool

**stickers**

List of all set stickers.

**Type** List[telegram.Sticker]

**thumb**

Optional. Sticker set thumbnail in the .WEBP, .TGS or .WEBM format.

**Type** telegram.PhotoSize
classmethod de_json(data, bot)
    See telegram.TelegramObject.de_json().

to_dict()
    See telegram.TelegramObject.to_dict().

telegram.MaskPosition

class telegram.MaskPosition (point, x_shift, y_shift, scale, **_kwargs)
    Bases: telegram.base.TelegramObject
    This object describes the position on faces where a mask should be placed by default.

    Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their point, x_shift, y_shift and, scale are equal.

    point
        The part of the face relative to which the mask should be placed. One of 'forehead', 'eyes', 'mouth', or 'chin'.
        Type str

    x_shift
        Shift by X-axis measured in widths of the mask scaled to the face size, from left to right.
        Type float

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

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

    Note: 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. One of 'forehead', 'eyes', 'mouth', or 'chin'.
- **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: ClassVar[str] = 'chin'
    telegram.constants.STICKER_CHIN

EYES: ClassVar[str] = 'eyes'
    telegram.constants.STICKER_EYES

FOREHEAD: ClassVar[str] = 'forehead'
    telegram.constants.STICKER_FOREHEAD
MOUTH: ClassVar[str] = 'mouth'

```
from telegram.constants import STICKER_MOUTH
```

```python
classmethod de_json(data, bot)
    See telegram.TelegramObject.de_json().
```

## 3.2.76 Inline Mode

**telegram.InlineQuery**

class telegram.InlineQuery(id, from_user, query, offset, location=None, chat_type=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.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their id is equal.

**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.
- **query (str)** – Text of the query (up to 256 characters).
- **offset (str)** – Offset of the results to be returned, can be controlled by the bot.
- **chat_type (str, optional)** – Type of the chat, from which the inline query was sent. Can be either `telegram.Chat.SENDER` for a private chat with the inline query sender, `telegram.Chat.PRIVATE`, `telegram.Chat.GROUP`, `telegram.Chat.SUPERGROUP` or `telegram.Chat.CHANNEL`. The chat type should be always known for requests sent from official clients and most third-party clients, unless the request was sent from a secret chat. New in version 13.5.
- **location (telegram.Location, optional)** – Sender location, only for bots that request user location.
- ****kwargs (dict)** – Arbitrary keyword arguments.

**id**

Unique identifier for this query.

Type `str`

**from_user**

Sender.

Type `telegram.User`

**query**

Text of the query (up to 256 characters).

Type `str`

**offset**

Offset of the results to be returned, can be controlled by the bot.
Type `str`

**location**
Optional. Sender location, only for bots that request user location.

Type `telegram.Location`

**chat_type**
Type of the chat, from which the inline query was sent.

New in version 13.5.

Type `str`, optional

**MAX_RESULTS**: ClassVar[int] = 50

`telegram.constants.MAX_INLINE_QUERY_RESULTS`

New in version 13.2.

`answer` (results, cache_time=300, is_personal=None, next_offset=None, switch_pm_text=None, switch_pm_parameter=None, timeout=None, current_offset=None, api_kwargs=None, auto_pagination=False)

Shortcut for:

```python
bot.answer_inline_query(update.inline_query.id, *args, current_offset=self.offset if auto_pagination else None, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.answer_inline_query()`.

**Parameters**

`auto_pagination` (bool, optional) – If set to True, offset will be passed as current_offset to `telegram.Bot.answer_inline_query()`.

Defaults to False.

**Raises** `TypeError` - If both current_offset and auto_pagination are supplied.

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

See `telegram.TelegramObject.de_json()`.

---

telegram.InlineQueryResult

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

Bases: `telegram.base.TelegramObject`

Baseclass for the InlineQueryResult* classes.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their id is equal.

**Note:** All URLs passed in inline query results will be available to end users and therefore must be assumed to be public.

**Parameters**

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

**type**
Type of the result.
Type `str`

**id**

Unique identifier for this result, 1-64 Bytes.

Type `str`

**to_dict()**

See `telegram.TelegramObject.to_dict()`.

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.

**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.

**type**

‘article’.

Type `str`

**id**

Unique identifier for this result, 1-64 Bytes.

Type `str`

**title**

Title of the result.

Type `str`

**input_message_content**

Content of the message to be sent.

Type `telegram.InputMessageContent`

**reply_markup**

Optional. Inline keyboard attached to the message.
Type `telegram.ReplyMarkup`  

url  
Optional. URL of the result.  
Type `str`

hide_url  
Optional. Pass `True`, if you don’t want the URL to be shown in the message.  
Type `bool`

description  
Optional. Short description of the result.  
Type `str`

thumb_url  
Optional. Url of the thumbnail for the result.  
Type `str`

thumb_width  
Optional. Thumbnail width.  
Type `int`

thumb_height  
Optional. Thumbnail height.  
Type `int`

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, caption_entities=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.

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) – Performer.
- `audio_duration` (`str`, optional) – Audio duration in seconds.
- `caption` (`str`, optional) – Caption, 0-1024 characters after entities parsing.
- `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.
- `caption_entities` (`List[telegram.MessageEntity]`, optional) – List of special entities that appear in the caption, which can be specified instead of `parse_mode`.
- `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.

**type**

Type: str

**id**

Unique identifier for this result, 1-64 bytes.

Type: str

**audio_url**

A valid URL for the audio file.

Type: str

**title**

Title.

Type: str

**performer**

Optional. Performer.

Type: str

**audio_duration**

Optional. Audio duration in seconds.

Type: str

**caption**

Optional. Caption, 0-1024 characters after entities parsing.

Type: str

**parse_mode**

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

Type: str

**caption_entities**

Optional. List of special entities that appear in the caption, which can be specified instead of parse_mode.

Type: List[telegram.MessageEntity]

**reply_markup**

Optional. Inline keyboard attached to the message.

Type: telegram.InlineKeyboardMarkup

**input_message_content**

Optional. Content of the message to be sent instead of the audio.

Type: telegram.InputMessageContent
telegram.InlineQueryResultCachedAudio

```python
class telegram.InlineQueryResultCachedAudio(id, audio_file_id, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, caption_entities=None, **kwargs)
```

Bases: `telegram.inline.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.

**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 after entities parsing.
- **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.
- **caption_entities (List[telegram.MessageEntity], optional)** – List of special entities that appear in the caption, which can be specified instead of `parse_mode`.
- **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.

**Type**

- `type` ‘audio’.
  
  Type `str`

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

- `audio_file_id`
  
  A valid file identifier for the audio file.
  
  Type `str`

- `caption`
  
  Optional. Caption, 0-1024 characters after entities parsing.
  
  Type `str`

- `parse_mode`
  
  Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
  
  Type `str`

- `caption_entities`
  
  Optional. List of special entities that appear in the caption, which can be specified instead of `parse_mode`.
  
  Type `List[telegram.MessageEntity]`
reply_markup
Optional. Inline keyboard attached to the message.

Type telegram.InlineKeyboardMarkup

input_message_content
Optional. Content of the message to be sent instead of the audio.

Type telegram.InputMessageContent

telegram.InlineQueryResultCachedDocument

class telegram.InlineQueryResultCachedDocument
(id, title, document_file_id, description=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, caption_entities=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.

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 of the document to be sent, 0-1024 characters after entities parsing.

• **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.

• **caption_entities**(List[telegram.MessageEntity], optional) – List of special entities that appear in the caption, which can be specified instead of parse_mode.

• **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.

type
‘document’.

Type str

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

Type str

title
Title for the result.

Type str
document_file_id
A valid file identifier for the file.

Type str
description
Optional. Short description of the result.

Type str
caption
Optional. Caption of the document to be sent, 0-1024 characters after entities parsing.

Type str
parse_mode
Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.

Type str
caption_entities
Optional. List of special entities that appear in the caption, which can be specified instead of parse_mode.

Type List[telegram.MessageEntity]
reply_markup
Optional. Inline keyboard attached to the message.

Type telegram.InlineKeyboardMarkup
input_message_content
Optional. Content of the message to be sent instead of the file.

Type telegram.InputMessageContent

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

Parameters

- **id**(str) – Unique identifier for this result, 1-64 bytes.
- **gif_file_id**(str) – A valid file identifier for the GIF file.
- **title**(str, optional) – Title for the result.caption (str, optional):
- **caption**(str, optional) – Caption of the GIF file to be sent, 0-1024 characters after entities parsing.
- **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.
• **caption_entities** (List[telegram.MessageEntity], optional) – List of special entities that appear in the caption, which can be specified instead of parse_mode.

• **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.

type
  ‘gif’.
  Type str

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

gif_file_id
  A valid file identifier for the GIF file.
  Type str

title
  Optional. Title for the result.
  Type str

caption
  Optional. Caption of the GIF file to be sent, 0-1024 characters after entities parsing.
  Type str

parse_mode
  Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
  Type str

caption_entities
  Optional. List of special entities that appear in the caption, which can be specified instead of parse_mode.
  Type List[telegram.MessageEntity]

reply_markup
  Optional. Inline keyboard attached to the message.
  Type telegram.InlineKeyboardMarkup

input_message_content
  Optional. Content of the message to be sent instead of the gif.
  Type telegram.InputMessageContent
class telegram.InlineQueryResultCachedMpeg4Gif(id, mpeg4_file_id, title=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, caption_entities=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.

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 of the MPEG-4 file to be sent, 0-1024 characters after entities parsing.
- **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.
- **caption_entities** *(List[telegram.MessageEntity], optional)* – List of special entities that appear in the caption, which can be specified instead of parse_mode.
- **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.

type 'mpeg4_gif'.

**id**

Unique identifier for this result, 1-64 bytes.

**mpeg4_file_id**

A valid file identifier for the MP4 file.

**title**

Optional. Title for the result.

**caption**

Optional. Caption of the MPEG-4 file to be sent, 0-1024 characters after entities parsing.

**parse_mode**

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text
or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

*Type*  
`str`

**caption_entities**

Optional. List of special entities that appear in the caption, which can be specified instead of `parse_mode`.

*Type*  
`List[telegram.MessageEntity]`

**reply_markup**

Optional. Inline keyboard attached to the message.

*Type*  
`telegram.InlineKeyboardMarkup`

**input_message_content**

Optional. Content of the message to be sent instead of the MPEG-4 file.

*Type*  
`telegram.InputMessageContent`

---

**telegram.InlineQueryResultCachedPhoto**

```python
class telegram.InlineQueryResultCachedPhoto(id, photo_file_id, title=None, description=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, caption_entities=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.

**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 of the photo to be sent, 0-1024 characters after entities parsing.
- **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.
- **caption_entities** *(List[telegram.MessageEntity], optional)* – List of special entities that appear in the caption, which can be specified instead of `parse_mode`.
- **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.

*Type*  
`str`
id
    Unique identifier for this result, 1-64 bytes.
    Type str

photo_file_id
    A valid file identifier of the photo.
    Type str

title
    Optional. Title for the result.
    Type str
description
    Optional. Short description of the result.
    Type str
caption
    Optional. Caption of the photo to be sent, 0-1024 characters after entities parsing.
    Type str

parse_mode
    Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
    Type str
caption_entities
    Optional. List of special entities that appear in the caption, which can be specified instead of parse_mode.
    Type List[telegram.MessageEntity]

reply_markup
    Optional. Inline keyboard attached to the message.
    Type telegram.InlineKeyboardMarkup

input_message_content
    Optional. Content of the message to be sent instead of the photo.
    Type telegram.InputMessageContent

**class telegram.InlineQueryResultCachedSticker**

    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.

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.

Parameters

- **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.

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

```python
type
'sticker'.
Type str

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

sticker_file_id
A valid file identifier of the sticker.
Type str

reply_markup
Optional. Inline keyboard attached to the message.
Type telegram.InlineKeyboardMarkup

input_message_content
Optional. Content of the message to be sent instead of the sticker.
Type telegram.InputMessageContent
```

telegram.InlineQueryResultCachedVideo

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

**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 of the video to be sent, 0-1024 characters after entities parsing.

• **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.

• **caption_entities** *(List[telegram.MessageEntity], optional)* – List of special entities that appear in the caption, which can be specified instead of **parse_mode**.

• **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.

type
    ‘video’.  

    Type  str

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

    Type  str

video_file_id
    A valid file identifier for the video file.  

    Type  str

title
    Title for the result.  

    Type  str

description
    Optional. Short description of the result.  

    Type  str

caption
    Optional. Caption of the video to be sent, 0-1024 characters after entities parsing.  

    Type  str

parse_mode
    Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.  

    Type  str

caption_entities
    Optional. List of special entities that appear in the caption, which can be specified instead of parse_mode.  

    Type  List[telegram.MessageEntity]

reply_markup
    Optional. Inline keyboard attached to the message.  

    Type  telegram.InlineKeyboardMarkup

input_message_content
    Optional. Content of the message to be sent instead of the video.  

    Type  telegram.InputMessageContent

`telegram.InlineQueryResultCachedVoice`

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

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 after entities parsing.
• **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.
• **caption_entities** *(List[telegram.MessageEntity], optional)* – List of special entities that appear in the caption, which can be specified instead of **parse_mode**.
• **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 message.
• ****kwargs** *(dict)* – Arbitrary keyword arguments.

```python
type = 'voice'.

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

voice_file_id
    A valid file identifier for the voice message.

title
    Voice message title.

caption
    Optional. Caption, 0-1024 characters after entities parsing.

caption_entities
    Optional. List of special entities that appear in the caption, which can be specified instead of **parse_mode**.

reply_markup
    Optional. Inline keyboard attached to the message.

input_message_content
    Optional. Content of the message to be sent instead of the voice message.
```


**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.

**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.

**type**

*contact*.

**Type** *str*

**id**

Unique identifier for this result, 1-64 bytes.

**Type** *str*

**phone_number**

Contact’s phone number.

**Type** *str*

**first_name**

Contact’s first name.

**Type** *str*

**last_name**

Optional. Contact’s last name.

**Type** *str*

**vcard**

Optional. Additional data about the contact in the form of a vCard, 0-2048 bytes.

**Type** *str*
reply_markup
Optional. Inline keyboard attached to the message.
Type telegram.InlineKeyboardMarkup

input_message_content
Optional. Content of the message to be sent instead of the contact.
Type telegram.InputMessageContent

thumb_url
Optional. Url of the thumbnail for the result.
Type str
thumb_width
Optional. Thumbnail width.
Type int
gthumb_height
Optional. Thumbnail height.
Type int

telegram.InlineQueryResultDocument
class telegram.InlineQueryResultDocument (id, document_url, title, mime_type, caption=None, description=None, reply_markup=None, input_message_content=None, thumb_url=None, thumb_width=None, thumb_height=None, parse_mode=None, caption_entities=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.

Parameters

- **id** (str) – Unique identifier for this result, 1-64 bytes.
- **title** (str) – Title for the result.
- **caption** (str, optional) – Caption of the document to be sent, 0-1024 characters after entities parsing.
- **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.
- **caption_entities** (List[telegram.MessageEntity], optional) – List of special entities that appear in the caption, which can be specified instead of parse_mode.
- **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.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Unique identifier for this result, 1-64 bytes.</td>
</tr>
<tr>
<td>str</td>
<td>Title for the result.</td>
</tr>
<tr>
<td>str</td>
<td>Optional. Caption of the document to be sent, 0-1024 characters after entities parsing.</td>
</tr>
<tr>
<td>str</td>
<td>Mime type of the content of the file, either “application/pdf” or “application/zip”.</td>
</tr>
<tr>
<td>str</td>
<td>Optional. Short description of the result.</td>
</tr>
<tr>
<td>str</td>
<td>Optional. Inline keyboard attached to the message.</td>
</tr>
<tr>
<td>str</td>
<td>A valid URL for the file.</td>
</tr>
<tr>
<td>List</td>
<td>Optional. List of special entities that appear in the caption, which can be specified instead of parse_mode.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>document_url</th>
<th>A valid URL for the file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>mime_type</td>
<td>Mime type of the content of the file, either “application/pdf” or “application/zip”.</td>
</tr>
<tr>
<td>description</td>
<td>Optional. Short description of the result.</td>
</tr>
<tr>
<td>parse_mode</td>
<td>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.</td>
</tr>
<tr>
<td>caption_entities</td>
<td>Optional. List of special entities that appear in the caption, which can be specified instead of parse_mode.</td>
</tr>
<tr>
<td>id</td>
<td>Unique identifier for this result, 1-64 bytes.</td>
</tr>
<tr>
<td>title</td>
<td>Title for the result.</td>
</tr>
<tr>
<td>type</td>
<td>‘document’.</td>
</tr>
<tr>
<td>caption</td>
<td>Optional. Caption of the document to be sent, 0-1024 characters after entities parsing.</td>
</tr>
<tr>
<td>parse_mode</td>
<td>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.</td>
</tr>
<tr>
<td>caption_entities</td>
<td>Optional. List of special entities that appear in the caption, which can be specified instead of parse_mode.</td>
</tr>
<tr>
<td>document_url</td>
<td>A valid URL for the file.</td>
</tr>
<tr>
<td>mime_type</td>
<td>Mime type of the content of the file, either “application/pdf” or “application/zip”.</td>
</tr>
<tr>
<td>description</td>
<td>Optional. Short description of the result.</td>
</tr>
<tr>
<td>reply_markup</td>
<td>Optional. Inline keyboard attached to the message.</td>
</tr>
<tr>
<td>input_message_content</td>
<td>Optional. Content of the message to be sent instead of the file.</td>
</tr>
<tr>
<td>thumb_url</td>
<td>Optional. URL of the thumbnail (jpeg only) for the file.</td>
</tr>
</tbody>
</table>
Type  str

**thumb_width**  
Optional. Thumbnail width.

Type  int

**thumb_height**  
Optional. Thumbnail height.

Type  int

game_short_name


type  str

id


game_short_name


title=None  
caption=None  
reply_markup=None  
input_message_content=None  
gif_duration=None  
parse_mode=None  
thumb_mime_type=None  
caption_entities=None  
**kwargs


telegram.InlineQueryResultGame


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

Bases:  telegram.inline.inlinequeryresult.InlineQueryResult

Represents a telegram.Game.

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.


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,  
thumb_mime_type=None,  
caption_entities=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.

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 (JPEG or GIF) or animated (MPEG4) thumbnail for the result.
• `thumb_mime_type (str, optional) – MIME type of the thumbnail, must be one of 'image/jpeg', 'image/gif', or 'video/mp4'. Defaults to 'image/jpeg'.
• `title (str, optional) – Title for the result.
• `caption (str, optional) – Caption of the GIF file to be sent, 0-1024 characters after entities parsing.
• `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.
• `caption_entities (List[telegram.MessageEntity], optional) – List of special entities that appear in the caption, which can be specified instead of `parse_mode`.
• `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 animation.
• **kwargs (dict) – Arbitrary keyword arguments.

**type**

Type `str`

**id**

Unique identifier for this result, 1-64 bytes.

Type `str`

**gif_url**

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

Type `str`

**gif_width**

Optional. Width of the GIF.

Type `int`

**gif_height**

Optional. Height of the GIF.

Type `int`

**gif_duration**

Optional. Duration of the GIF.

Type `int`

**thumb_url**

URL of the static (JPEG or GIF) or animated (MPEG4) thumbnail for the result.

Type `str`
thumb_mime_type
Optional. MIME type of the thumbnail.
Type str

title
Optional. Title for the result.
Type str
caption
Optional. Caption of the GIF file to be sent, 0-1024 characters after entities parsing.
Type str
parse_mode
Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
Type str
caption_entities
Optional. List of special entities that appear in the caption, which can be specified instead of parse_mode.
Type List[telegram.MessageEntity]
reply_markup
Optional. Inline keyboard attached to the message.
Type telegram.InlineKeyboardMarkup
input_message_content
Optional. Content of the message to be sent instead of the GIF animation.
Type telegram.InputMessageContent

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, horizontal_accuracy=None, heading=None, proximity_alert_radius=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.

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.
- **horizontal_accuracy**(float, optional) – The radius of uncertainty for the location, measured in meters; 0-1500.
- **live_period**(int, optional) – Period in seconds for which the location can be updated, should be between 60 and 86400.
• **heading** (int, optional) – For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.

• **proximity_alert_radius** (int, optional) – For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and 100000 if specified.

• **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.

**type**

Type **str**

**id**

Unique identifier for this result, 1-64 bytes.

Type **str**

**latitude**

Location latitude in degrees.

Type **float**

**longitude**

Location longitude in degrees.

Type **float**

**title**

Location title.

Type **str**

**horizontal_accuracy**

Optional. The radius of uncertainty for the location, measured in meters.

Type **float**

**live_period**

Optional. Period in seconds for which the location can be updated, should be between 60 and 86400.

Type **int**

**heading**

Optional. For live locations, a direction in which the user is moving, in degrees.

Type **int**

**proximity_alert_radius**

Optional. For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters.

Type **int**

**reply_markup**

Optional. Inline keyboard attached to the message.

Type **telegram.InlineKeyboardMarkup**
**input_message_content**

Optional. Content of the message to be sent instead of the location.

Type `telegram.InputMessageContent`

**thumb_url**

Optional. Url of the thumbnail for the result.

Type `str`

**thumb_width**

Optional. Thumbnail width.

Type `int`

**thumb_height**

Optional. Thumbnail height.

Type `int`

**telegram.InlineQueryResultMpeg4Gif**

class `telegram.InlineQueryResultMpeg4Gif`(*id, mpeg4_url, thumb_url, mpeg4_width=None, mpeg4_height=None, title=None, caption=None, reply_markup=None, input_message_content=None, mpeg4_duration=None, parse_mode=None, thumb_mime_type=None, caption_entities=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.

**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.
- **thumb_mime_type** (str) – Optional. MIME type of the thumbnail, must be one of 'image/jpeg', 'image/gif', or 'video/mp4'. Defaults to 'image/jpeg'.
- **title** (str, optional) – Title for the result.
- **caption** (str, optional) – Caption of the MPEG-4 file to be sent, 0-1024 characters after entities parsing.
- **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.
- **caption_entities** (List[`telegram.MessageEntity`], optional) – List of special entities that appear in the caption, which can be specified instead of `parse_mode`. 
• **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 animation.

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

```python
py

• **type**
  ```py
  'mpeg4_gif'.
  ```
  Type int

• **id**
  Unique identifier for this result, 1-64 bytes.
  Type str

• **mpeg4_url**
  A valid URL for the MP4 file. File size must not exceed 1MB.
  Type str

• **mpeg4_width**
  Optional. Video width.
  Type int

• **mpeg4_height**
  Optional. Video height.
  Type int

• **mpeg4_duration**
  Optional. Video duration.
  Type int

• **thumb_url**
  URL of the static (JPEG or GIF) or animated (MPEG4) thumbnail for the result.
  Type str

• **thumb_mime_type**
  Optional. MIME type of the thumbnail.
  Type str

• **title**
  Optional. Title for the result.
  Type str

• **caption**
  Optional. Caption of the MPEG-4 file to be sent, 0-1024 characters after entities parsing.
  Type str

• **parse_mode**
  Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.
  Type str

• **caption_entities**
  Optional. List of special entities that appear in the caption, which can be specified instead of **parse_mode**.
  Type List[`telegram.MessageEntity`]
```
reply_markup
Optional. Inline keyboard attached to the message.
Type `telegram.InlineKeyboardMarkup`

input_message_content
Optional. Content of the message to be sent instead of the video animation.
Type `telegram.InputMessageContent`

telegram.InlineQueryResultPhoto
class `telegram.InlineQueryResultPhoto` *(id, photo_url, thumb_url, photo_width=\text{None}, photo_height=\text{None}, title=\text{None}, description=\text{None}, caption=\text{None}, reply_markup=\text{None}, input_message_content=\text{None}, parse_mode=\text{None}, caption_entities=\text{None}, \text{**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.

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 of the photo to be sent, 0-1024 characters after entities parsing.
- **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.
- **caption_entities**(List[telegram.MessageEntity], optional) – List of special entities that appear in the caption, which can be specified instead of `parse_mode`.
- **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.
- **\text{**kwargs}**(dict) – Arbitrary keyword arguments.

**type**

- ’photo’.

**id**

Unique identifier for this result, 1-64 bytes.

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

thumb_url
URL of the thumbnail for the photo.
    Type  str

photo_width
Optional. Width of the photo.
    Type  int

photo_height
Optional. Height of the photo.
    Type  int

title
Optional. Title for the result.
    Type  str

description
Optional. Short description of the result.
    Type  str

caption
Optional. Caption of the photo to be sent, 0-1024 characters after entities parsing.
    Type  str

parse_mode
Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
    Type  str

caption_entities
Optional. List of special entities that appear in the caption, which can be specified instead of parse_mode.
    Type  List[telegram.MessageEntity]

reply_markup
Optional. Inline keyboard attached to the message.
    Type  telegram.InlineKeyboardMarkup

input_message_content
Optional. Content of the message to be sent instead of the photo.
    Type  telegram.InputMessageContent
### telegram.InlineQueryResultVenue

**class telegram.InlineQueryResultVenue**

```python
class telegram.InlineQueryResultVenue(id, latitude, longitude, title, address, foursquare_id=None, foursquare_type=None, reply_markup=None, input_message_content=None, thumb_url=None, thumb_width=None, thumb_height=None, google_place_id=None, google_place_type=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.

**Note:** Foursquare details and Google Pace details are mutually exclusive. However, this behaviour is undocumented and might be changed by Telegram.

**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”).
- **google_place_id (str, optional)** – Google Places identifier of the venue.
- **google_place_type (str, optional)** – Google Places type of the venue. (See supported types.)
- **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.

**type**

`'venue'`.

**id**

Unique identifier for this result, 1-64 Bytes.

**latitude**

Latitude of the venue location in degrees.
**longitude**
Longitude of the venue location in degrees.

**title**
Title of the venue.

**address**
Address of the venue.

**foursquare_id**
Optional. Foursquare identifier of the venue if known.

**foursquare_type**
Optional. Foursquare type of the venue, if known.

**google_place_id**
Optional. Google Places identifier of the venue.

**google_place_type**
Optional. Google Places type of the venue.

**reply_markup**
Optional. Inline keyboard attached to the message.

**input_message_content**
Optional. Content of the message to be sent instead of the venue.

**thumb_url**
Optional. Url of the thumbnail for the result.

**thumb_width**
Optional. Thumbnail width.

**thumb_height**
Optional. Thumbnail height.
class telegram.InlineQueryResultVideo(id, video_url, mime_type, title, caption=None, video_width=None, video_height=None, video_duration=None, description=None, reply_markup=None, input_message_content=None, parse_mode=None, caption_entities=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.

**Note:** If an InlineQueryResultVideo message contains an embedded video (e.g., YouTube), you must replace its content using `input_message_content`.

### 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 after entities parsing.
- **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.
- **caption_entities (List[telegram.MessageEntity], optional)** – List of special entities that appear in the caption, which can be specified instead of `parse_mode`.
- **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. This field is required if InlineQueryResultVideo is used to send an HTML-page as a result (e.g., a YouTube video).
- ****kwargs (dict)** – Arbitrary keyword arguments.

**type**

`'video'`

**id**

Unique identifier for this result, 1-64 bytes.
**video_url**
A valid URL for the embedded video player or video file.

*Type* `str`

**mime_type**
Mime type of the content of video, “text/html” or “video/mp4”.

*Type* `str`

**thumb_url**
URL of the thumbnail (jpeg only) for the video.

*Type* `str`

**title**
Title for the result.

*Type* `str`

**caption**
Optional. Caption of the video to be sent, 0-1024 characters after entities parsing.

*Type* `str`

**parse_mode**
Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in `telegram.ParseMode` for the available modes.

*Type* `str`

**caption_entities**
Optional. List of special entities that appear in the caption, which can be specified instead of `parse_mode`.

*Type* `List[telegram.MessageEntity]`

**video_width**
Optional. Video width.

*Type* `int`

**video_height**
Optional. Video height.

*Type* `int`

**video_duration**
Optional. Video duration in seconds.

*Type* `int`

**description**
Optional. Short description of the result.

*Type* `str`

**reply_markup**
Optional. Inline keyboard attached to the message.

*Type* `telegram.InlineKeyboardMarkup`

**input_message_content**
Optional. Content of the message to be sent instead of the video. This field is required if Inline-QueryResultVideo is used to send an HTML-page as a result (e.g., a YouTube video).

*Type* `telegram.InputMessageContent`
**telegram.InlineQueryResultVoice**

class telegram.InlineQueryResultVoice(id, voice_url, title, voice_duration=None, caption=None, reply_markup=None, input_message_content=None, parse_mode=None, caption_entities=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.

**Parameters**

- **id** (str) – Unique identifier for this result, 1-64 bytes.
- **voice_url** (str) – A valid URL for the voice recording.
- **title** (str) – Recording title.
- **caption** (str, optional) – Caption, 0-1024 characters after entities parsing.
- **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.
- **caption_entities** (List[telegram.MessageEntity], optional) – List of special entities that appear in the caption, which can be specified instead of parse_mode.
- **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 recording.
- ****kwargs (dict) – Arbitrary keyword arguments.

**type**

'voice'.

Type **str**

**id**

Unique identifier for this result, 1-64 bytes.

Type **str**

**voice_url**

A valid URL for the voice recording.

Type **str**

**title**

Recording title.

Type **str**

**caption**

Optional. Caption, 0-1024 characters after entities parsing.

Type **str**

**parse_mode**

Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in the media caption. See the constants in telegram.ParseMode for the available modes.
Type `str`

caption_entities
Optional. List of special entities that appear in the caption, which can be specified instead of `parse_mode`.
Type `List[telegram.MessageEntity]`

voice_duration
Optional. Recording duration in seconds.
Type `int`

reply_markup
Optional. Inline keyboard attached to the message.
Type `telegram.InlineKeyboardMarkup`

input_message_content
Optional. Content of the message to be sent instead of the voice recording.
Type `telegram.InputMessageContent`

```python
from telegram import InputMessageContent
```

telegram.InputMessageContent

class telegram.InputMessageContent
Bases: telegram.base.TelegramObject

Base class for Telegram InputMessageContent Objects.


```python
from telegram import InlineKeyboardMarkup
```

telegram.InputTextMessageContent

class telegram.InputTextMessageContent
Bases: telegram.inline.inputmessagecontent.InputMessageContent

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `message_text` is equal.

Parameters

- `message_text` *(str)* – Text of the message to be sent, 1-4096 characters after entities parsing. 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. See the constants in `telegram.ParseMode` for the available modes.

- `entities` *(List[telegram.MessageEntity], optional)* – List of special entities that appear in the caption, which can be specified instead of `parse_mode`.

- `disable_web_page_preview` *(bool, optional)* – Disables link previews for links in the sent message.

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

```python
from telegram import ParseMode
```

**message_text**
Text of the message to be sent, 1-4096 characters after entities parsing.
Type `str`

**parse_mode**
Optional. Send Markdown or HTML, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot’s message. See the constants in `telegram.ParseMode` for the available modes.

Type `str`

**entities**
Optional. List of special entities that appear in the caption, which can be specified instead of `parse_mode`.

Type `List[telegram.MessageEntity]`

**disable_web_page_preview**
Optional. Disables link previews for links in the sent message.

Type `bool`

**to_dict()**
See `telegram.TelegramObject.to_dict()`.

---

**telegram.InputLocationMessageContent**

```python
class telegram.InputLocationMessageContent (latitude, longitude, live_period=None,
                                           horizontal_accuracy=None, heading=None,
                                           proximity_alert_radius=None,
                                           **_kwargs)

Bases: telegram.inline.inputmessagecontent.InputMessageContent
```

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `latitude` and `longitude` are equal.

**Parameters**

- **latitude** (float) – Latitude of the location in degrees.
- **longitude** (float) – Longitude of the location in degrees.
- **horizontal_accuracy** (float, optional) – The radius of uncertainty for the location, measured in meters; 0-1500.
- **live_period** (int, optional) – Period in seconds for which the location can be updated, should be between 60 and 86400.
- **heading** (int, optional) – For live locations, a direction in which the user is moving, in degrees. Must be between 1 and 360 if specified.
- **proximity_alert_radius** (int, optional) – For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters. Must be between 1 and 100000 if specified.
- **_kwargs** (dict) – Arbitrary keyword arguments.

**latitude**
Latitude of the location in degrees.

Type `float`

**longitude**
Longitude of the location in degrees.

Type `float`

**horizontal_accuracy**
Optional. The radius of uncertainty for the location, measured in meters.
Type `float`

**live_period**
Optional. Period in seconds for which the location can be updated.
Type `int`

**heading**
Optional. For live locations, a direction in which the user is moving, in degrees.
Type `int`

**proximity_alert_radius**
Optional. For live locations, a maximum distance for proximity alerts about approaching another chat member, in meters.
Type `int`

### `telegram.InputVenueMessageContent`

**class** `telegram.InputVenueMessageContent` *(latitude, longitude, title, address, foursquare_id=None, foursquare_type=None, google_place_id=None, google_place_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.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `latitude`, `longitude` and `title` are equal.

**Note:** Foursquare details and Google Places details are mutually exclusive. However, this behavior is undocumented and might be changed by Telegram.

**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”.)
- **google_place_id** `(str, optional)` – Google Places identifier of the venue.
- **google_place_type** `(str, optional)` – Google Places type of the venue. (See supported types.)
- ****kwargs** `(dict)` – Arbitrary keyword arguments.

**latitude**
Latitude of the location in degrees.
Type `float`

**longitude**
Longitude of the location in degrees.
Type `float`
title
    Name of the venue.
    Type str

address
    Address of the venue.
    Type str

def foursquare_id
    Optional. Foursquare identifier of the venue, if known.
    Type str

def foursquare_type
    Optional. Foursquare type of the venue, if known.
    Type str

def google_place_id
    Optional. Google Places identifier of the venue.
    Type str

def google_place_type
    Optional. Google Places type of the venue.
    Type str

class telegram.InputContactMessageContent
    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.

    Bases: telegram.inline.inputmessagecontent.InputMessageContent

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

    Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their phone_number is equal.

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.

phone_number
    Contact’s phone number.
    Type str

first_name
    Contact’s first name.
    Type str

last_name
    Optional. Contact’s last name.
    Type str
Optional. Additional data about the contact in the form of a vCard, 0-2048 bytes.

Type str

telegram.InputInvoiceMessageContent
class telegram.InputInvoiceMessageContent (title, description, payload, provider_token, currency, prices, max_tip_amount=None, suggested_tip_amounts=None, provider_data=None, photo_url=None, photo_size=None, photo_width=None, photo_height=None, need_phone_number=None, need_email=None, need_shipping_address=None, send_phone_number_to_provider=None, send_email_to_provider=None, is_flexible=None, **kwargs)

Bases: telegram.inline.inputmessagecontent.InputMessageContent

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

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their title, description, payload, provider_token, currency and prices are equal.

New in version 13.5.

Parameters

* title (str) – Product name, 1-32 characters

* description (str) – Product description, 1-255 characters

* 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) – Payment provider token, obtained via @Botfather.

* currency (str) – Three-letter ISO 4217 currency code, see more on currencies

* prices (List[telegram.LabeledPrice]) – Price breakdown, a JSON-serialized list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.)

* max_tip_amount (int, optional) – The maximum accepted amount for tips in the smallest units of the currency (integer, not float/double). For example, for a maximum tip of US$ 1.45 pass max_tip_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). Defaults to 0.

* suggested_tip_amounts (List[int], optional) – A JSON-serialized array of suggested amounts of tip in the smallest units of the currency (integer, not float/double). At most 4 suggested tip amounts can be specified. The suggested tip amounts must be positive, passed in a strictly increased order and must not exceed max_tip_amount.

* provider_data (str, optional) – A JSON-serialized object for data about the invoice, which will be shared with the payment provider. A detailed description of the required fields should be provided by the payment provider.

* 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 (int, 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 address 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.
• ****kwargs** (dict) – Arbitrary keyword arguments.

*title*
  Product name, 1-32 characters
  Type str

*description*
  Product description, 1-255 characters
  Type str

*payload*
  Bot-defined invoice payload, 1-128 bytes. This will not be displayed to the user, use for your internal processes.
  Type str

*provider_token*
  Payment provider token, obtained via @Botfather.
  Type str

*currency*
  Three-letter ISO 4217 currency code, see more on currencies
  Type str

*prices*
  Price breakdown, a JSON-serialized list of components.
  Type List[telegram.LabeledPrice]

*max_tip_amount*
  Optional. The maximum accepted amount for tips in the smallest units of the currency (integer, not float/double).
  Type int

*suggested_tip_amounts*
  Optional. A JSON-serialized array of suggested amounts of tip in the smallest units of the currency (integer, not float/double).
  Type List[int]
provider_data
    Optional. A JSON-serialized object for data about the invoice, which will be shared with the payment provider.
    Type str

photo_url
    Optional. URL of the product photo for the invoice.
    Type str

photo_size
    Optional. Photo size.
    Type int

photo_width
    Optional. Photo width.
    Type int

photo_height
    Optional. Photo height.
    Type int

need_name
    Optional. Pass True, if you require the user’s full name to complete the order.
    Type bool

need_phone_number
    Optional. Pass True, if you require the user’s phone number to complete the order
    Type bool

need_email
    Optional. Pass True, if you require the user’s email address to complete the order.
    Type bool

need_shipping_address
    Optional. Pass True, if you require the user’s shipping address to complete the order
    Type bool

send_phone_number_to_provider
    Optional. Pass True, if user’s phone number should be sent to provider.
    Type bool

send_email_to_provider
    Optional. Pass True, if user’s email address should be sent to provider.
    Type bool

is_flexible
    Optional. Pass True, if the final price depends on the shipping method.
    Type bool

classmethod de_json(data, bot)
    See telegram.TelegramObject.de_json().

to_dict()
    See telegram.TelegramObject.to_dict().
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.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their result_id is equal.

Note:

• In Python from is a reserved word, use from_user instead.
• It is necessary to enable inline feedback via @Botfather in order to receive these objects in updates.

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.

result_id

The unique identifier for the result that was chosen.

Type str

from_user

The user that chose the result.

Type telegram.User

location

Optional. Sender location.

Type telegram.Location

inline_message_id

Optional. Identifier of the sent inline message.

Type str

query

The query that was used to obtain the result.

Type str

classmethod de_json(data, bot)

See telegram.TelegramObject.de_json().
3.2.77 Payments

telegram.LabeledPrice

class telegram.LabeledPrice(label, amount, **kwargs)
Bases: telegram.base.TelegramObject

This object represents a portion of the price for goods or services. Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their label and amount are equal.

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.

label
Portion label.

Type str

amount
Price of the product in the smallest units of the currency.

Type int

telegram.Invoice

class telegram.Invoice(title, description, start_parameter, currency, total_amount, **kwargs)
Bases: telegram.base.TelegramObject

This object contains basic information about an invoice. Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their title, description, start_parameter, currency and total_amount are equal.

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. 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.

title
Product name.

Type str

description
Product description.
Type `str`

**start_parameter**
Unique bot deep-linking parameter.
Type `str`

**currency**
Three-letter ISO 4217 currency code.
Type `str`

**total_amount**
Total price in the smallest units of the currency.
Type `int`

**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.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `country_code`, `state`, `city`, `street_line1`, `street_line2` and `post_code` are equal.

**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.

**country_code**
ISO 3166-1 alpha-2 country code.
Type `str`

**state**
State, if applicable.
Type `str`

**city**
City.
Type `str`

**street_line1**
First line for the address.
Type `str`

**street_line2**
Second line for the address.
Type `str`

**post_code**
Address post code.
Type `str`

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. Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `name`, `phone_number`, `email` and `shipping_address` are equal.

    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.

    name
    Optional. User name.
    Type `str`

    phone_number
    Optional. User’s phone number.
    Type `str`

    email
    Optional. User email.
    Type `str`

    shipping_address
    Optional. User shipping address.
    Type telegram.ShippingAddress

classmethod de_json(data, bot)
    See telegram.TelegramObject.de_json().

telegram.ShippingOption

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

    This object represents one shipping option. Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `id` is equal.

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

title
Option title.
    Type  str

prices
List of price portions.
    Type  List[telegram.LabeledPrice]

to_dict()
See telegram.TelegramObject.to_dict().

** telegram.SuccessfulPayment**

class telegram.SuccessfulPayment(currency, total_amount, invoice_payload, telegram_payment_charge_id, provider_payment_charge_id, shipping_option_id=None, order_info=None, **kwargs)
Bases: telegram.base.TelegramObject
This object contains basic information about a successful payment.
Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their telegram_payment_charge_id and provider_payment_charge_id are equal.

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.

currency
    Three-letter ISO 4217 currency code.
    Type  str

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

invoice_payload
    Bot specified invoice payload.
    Type  str

shipping_option_id
    Optional. Identifier of the shipping option chosen by the user.
Type str

order_info
Optional. Order info provided by the user.
Type telegram.OrderInfo

telegram_payment_charge_id
Telegram payment identifier.
Type str

provider_payment_charge_id
Provider payment identifier.
Type str

classmethod de_json(data, bot)
See telegram.TelegramObject.de_json().

telegram.ShippingQuery

class telegram.ShippingQuery (id, from_user, invoice_payload, shipping_address, bot=None, **kwargs)
Bases: telegram.base.TelegramObject
This object contains information about an incoming shipping query.
Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their id is equal.

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.

id
Unique query identifier.
Type str

from_user
User who sent the query.
Type telegram.User

invoice_payload
Bot specified invoice payload.
Type str

shipping_address
User specified shipping address.
Type telegram.ShippingAddress
bot
Optional. The Bot to use for instance methods.

Type `telegram.Bot`

`answer` (*ok*, `shipping_options=None`, `error_message=None`, `timeout=None`, `api_kwargs=None`)

Shortcut for:

```
bot.answer_shipping_query(update.shipping_query.id, *args, **kwargs)
```

For the documentation of the arguments, please see `telegram.Bot.answer_shipping_query()`.

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

See `telegram.TelegramObject.de_json()`.

### telegram.PreCheckoutQuery

**class** `telegram.PreCheckoutQuery` (*id*, `from_user`, `currency`, `total_amount`, `invoice_payload`, `shipping_option_id=None`, `order_info=None`, `bot=None`, **kwargs)

**Bases:** `telegram.base.TelegramObject`

This object contains information about an incoming pre-checkout query.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `id` is equal.

**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 (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.

`id`

Unique query identifier.

Type `str`

`from_user`

User who sent the query.

Type `telegram.User`
currency
Three-letter ISO 4217 currency code.

Type str

total_amount
Total price in the smallest units of the currency.

Type int

invoice_payload
Bot specified invoice payload.

Type str

shipping_option_id
Optional. Identifier of the shipping option chosen by the user.

Type str

order_info
Optional. Order info provided by the user.

Type telegram.OrderInfo

bot
Optional. The Bot to use for instance methods.

Type telegram.Bot

answer( ok, error_message=None, timeout=None, api_kwargs=None)
Shortcut for:

```python
bot.answer_pre_checkout_query(update.pre_checkout_query.id, *args, **kwargs)
```

For the documentation of the arguments, please see telegram.Bot.

answer_pre_checkout_query().

classmethod de_json(data, bot)
See telegram.TelegramObject.de_json().

### 3.2.78 Games

telegram.Game

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

This object represents a game. Use BotFather to create and edit games, their short names will act as unique identifiers.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their title, description and photo are equal.

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 `telegram.Bot.set_game_score()`, or manually edited using `telegram.Bot.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.

**title**

Title of the game.

Type str

description

Description of the game.

Type str

photo

Photo that will be displayed in the game message in chats.

Type List[telegram.PhotoSize]

text

Optional. Brief description of the game or high scores included in the game message. Can be automatically edited to include current high scores for the game when the bot calls `telegram.Bot.set_game_score()`, or manually edited using `telegram.Bot.edit_message_text()`.

Type str

text_entities

Optional. Special entities that appear in text, such as usernames, URLs, bot commands, etc.

Type List[telegram.MessageEntity]

animation

Optional. Animation that will be displayed in the game message in chats. Upload via BotFather.

Type telegram.Animation

classmethod `de_json`(data, bot)

See `telegram.TelegramObject.de_json()`.

parse_text_entities(types=None)

Returns a dict that maps `telegram.MessageEntity` to str. It contains entities from this message filtered by their type attribute as the key, and the text that each entity belongs to as the value of the dict.

**Note:** This method should always be used instead of the text_entities attribute, since it calculates the correct substring from the message text based on UTF-16 codepoints. See parse_text_entity for more info.

**Parameters**

**types** (List[str], optional) – List of MessageEntity types as strings. If the type attribute of an entity is contained in this list, it will be returned. Defaults to `telegram.MessageEntity.ALL_TYPES`.

**Returns**

A dictionary of entities mapped to the text that belongs to them, calculated based on UTF-16 codepoints.

**Return type**

Dict[telegram.MessageEntity, str]
parse_text_entity(entity)
Returns the text from a given telegram.MessageEntity.

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

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

Returns The text of the given entity.
Return type str

Raises RuntimeError – If this game has no text.

to_dict()
See telegram.TelegramObject.to_dict().

telegram.Callbackgame
class telegram.CallbackGame
    Bases: telegram.base.TelegramObject

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

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.

    Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their position, user and score are equal.

    Parameters

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

    position
    Position in high score table for the game.

    Type int

    user
    User.

    Type telegram.User

    score
    Score.

    Type int

classmethod de_json(data, bot)
    See telegram.TelegramObject.de_json().
3.2.79 Passport

`telegram.PassportElementError`

class `telegram.PassportElementError(source, type, message, **kwargs)`
   Bases: `telegram.base.TelegramObject`
   Baseclass for the PassportElementError* classes.
   This object represents an error in the Telegram Passport element which was submitted that should be re-
   solved by the user.
   Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if
   their `source` and `type` are equal.

   **Parameters**

   * `source` (str) – Error source.
   * `type` (str) – The section of the user’s Telegram Passport which has the error.
   * `**kwargs` (dict) – Arbitrary keyword arguments.

   `source`
      Error source.
      Type str

   `type`
      The section of the user’s Telegram Passport which has the error.
      Type str

   `message`
      Error message.
      Type str

`telegram.PassportElementErrorFile`

class `telegram.PassportElementErrorFile(type, file_hash, message, **kwargs)`
   Bases: `telegram.passport.passportelementerrors.PassportElementError`
   Represents an issue with a document scan. The error is considered resolved when the file with the document
   scan changes.
   Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if
   their `source`, `type`, `file_hash`, and `message` are equal.

   **Parameters**

   * `type` (str) – The section of the user’s Telegram Passport which has the is-
     sue, 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.

   `type`
      The section of the user’s Telegram Passport which has the issue, one of "utility_bill", 
      "bank_statement", "rental_agreement", "passport_registration", 
      "temporary_registration".
      Type str
**telegram.PassportElementErrorFiles**

```python
class telegram.PassportElementErrorFiles (type, file_hashes, message, **kwargs)
```

Represents an issue with a list of scans. The error is considered resolved when the list of files with the document scans changes.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their source, type, file_hashes, and message are equal.

**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.

**type**
The section of the user’s Telegram Passport which has the issue, one of "utility_bill", "bank_statement", "rental_agreement", "passport_registration", "temporary_registration".

**file_hashes**
List of base64-encoded file hashes.

**message**
Error message.

---

**telegram.PassportElementErrorReverseSide**

```python
class telegram.PassportElementErrorReverseSide (type, file_hash, message, **kwargs)
```

Represents an issue with the reverse side of a document. The error is considered resolved when the file with the reverse side of the document changes.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their source, type, file_hash, and message are equal.

**Parameters**

- **type** *(str)* – The section of the user’s Telegram Passport which has the issue, one of "driver_license", "identity_card".
• **file_hash** *(str)* – Base64-encoded hash of the file with the reverse side of the document.
• **message** *(str)* – Error message.
• **kwargs** *(dict)* – Arbitrary keyword arguments.

**type**
The section of the user’s Telegram Passport which has the issue, one of "driver_license", "identity_card".

Type str

**file_hash**
Base64-encoded hash of the file with the reverse side of the document.

Type str

**message**
Error message.

Type str

### telegram.PassportElementErrorFrontSide

**class** telegram.PassportElementErrorFrontSide *(type, file_hash, message, **kwargs)*

Bases: telegrampassport.passporelementerrors.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.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their source, type, file_hash, and message are equal.

**Parameters**

• **type** *(str)* – The section of the user’s Telegram Passport which has the issue, one of "passport", "driver_license", "identity_card", "internal_passport".
• **file_hash** *(str)* – Base64-encoded hash of the file with the front side of the document.
• **message** *(str)* – Error message.
• **kwargs** *(dict)* – Arbitrary keyword arguments.

**type**
The section of the user’s Telegram Passport which has the issue, one of "passport", "driver_license", "identity_card", "internal_passport".

Type str

**file_hash**
Base64-encoded hash of the file with the front side of the document.

Type str

**message**
Error message.

Type str
telegram.PassportElementErrorDataField

class telegram.PassportElementErrorDataField(type, field_name, data_hash, message, **kwargs)

Bases: telegram.passport.passportelementerrors.PassportElementError

Represents an issue in one of the data fields that was provided by the user. The error is considered resolved when the field’s value changes.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `source`, `type`, `field_name`, `data_hash` and `message` are equal.

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.

```

type

The section of the user’s Telegram Passport which has the error, one of "personal_details", "passport", "driver_license", "identity_card", "internal_passport", "address".

Type str

field_name

Name of the data field which has the error.

Type str

data_hash

Base64-encoded data hash.

Type str

message

Error message.

Type str
```

telegram.PassportElementErrorSelfie

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.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `source`, `type`, `file_hash`, and `message` are equal.

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.

**type**

The section of the user’s Telegram Passport which has the issue, one of "passport", "driver_license", "identity_card", "internal_passport".

Type str

**file_hash**

Base64-encoded hash of the file with the selfie.

Type str

**message**

Error message.

Type str

**telegram.PassportElementErrorTranslationFile**

class telegram.PassportElementErrorTranslationFile(type, file_hash, message, **kwargs)

Bases: telegrampassport.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.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their source, type, file_hash, and message are equal.

**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.

**type**

Type of element of the user’s Telegram Passport which has the issue, one of "passport", "driver_license", "identity_card", "internal_passport", "utility_bill", "bank_statement", "rental_agreement", "passport_registration", "temporary_registration".

Type str

**file_hash**

Base64-encoded hash of the file.

Type str

**message**

Error message.

Type str
class telegram.PassportElementErrorTranslationFiles

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 changes.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their source, type, file_hashes, and message are equal.

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.

type

Type of element of the user’s Telegram Passport which has the issue, one of "passport", "driver_license", "identity_card", "internal_passport", "utility_bill", "bank_statement", "rental_agreement", "passport_registration", "temporary_registration".

Type str

type

List of base64-encoded file hashes.

Type List[str]

message

Error message.

Type str

class telegram.PassportElementErrorUnspecified

Bases: telegram.passport.passportelementerrors.PassportElementError

Represents an issue in an unspecified place. The error is considered resolved when new data is added.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their source, type, element_hash, and message are equal.

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.

type

Type of element of the user’s Telegram Passport which has the issue.

Type str
element_hash
    Base64-encoded element hash.
    
    Type str

message
    Error message.
    
    Type str

telegram.Credentials

class telegram.Credentials(secure_data, nonce, bot=None, **kwargs)
    Bases: telegram.base.TelegramObject

    secure_data
        Credentials for encrypted data
        
        Type telegram.SecureData

    nonce
        Bot-specified nonce
        
        Type str

classmethod de_json(data, bot)
    See telegram.TelegramObject.de_json().

telegram.DataCredentials

class telegram.DataCredentials(data_hash, secret, **kwargs)
    Bases: telegram.passport.credentials._CredentialsBase

    These credentials can be used to decrypt encrypted data from the data field in EncryptedPassportData.

    Parameters
        • data_hash (str) – Checksum of encrypted data
        • secret (str) – Secret of encrypted data

    hash
        Checksum of encrypted data
        
        Type str

    secret
        Secret of encrypted data
        
        Type str

to_dict()
    See telegram.TelegramObject.to_dict().
class telegram.SecureData (personal_details=None, passport=None, internal_passport=None, driver_license=None, identity_card=None, address=None, utility_bill=None, bank_statement=None, rental_agreement=None, passport_registration=None, temporary_registration=None, bot=None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents the credentials that were used to decrypt the encrypted data. All fields are optional and depend on fields that were requested.

personal_details
Credentials for encrypted personal details.

Type telegram.SecureValue, optional

passport
Credentials for encrypted passport.

Type telegram.SecureValue, optional

internal_passport
Credentials for encrypted internal passport.

Type telegram.SecureValue, optional

driver_license
Credentials for encrypted driver license.

Type telegram.SecureValue, optional

identity_card
Credentials for encrypted ID card

Type telegram.SecureValue, optional

address
Credentials for encrypted residential address.

Type telegram.SecureValue, optional

utility_bill
Credentials for encrypted utility bill.

Type telegram.SecureValue, optional

bank_statement
Credentials for encrypted bank statement.

Type telegram.SecureValue, optional

rental_agreement
Credentials for encrypted rental agreement.

Type telegram.SecureValue, optional

passport_registration
Credentials for encrypted registration from internal passport.

Type telegram.SecureValue, optional

temporary_registration
Credentials for encrypted temporary registration.

Type telegram.SecureValue, optional

classmethod de_json (data, bot)
See telegram.TelegramObject.de_json().
**telegram.SecureValue**

```python
class telegram.SecureValue(
    data=None, front_side=None, reverse_side=None, selfie=None,
    files=None, translation=None, bot=None, **kwargs)
```

**Bases:** `telegram.base.TelegramObject`

This object represents the credentials that were used to decrypt the encrypted value. All fields are optional and depend on the type of field.

**data**

Credentials for encrypted Telegram Passport data. Available for “personal_details”, “passport”, “driver_license”, “identity_card”, “identity_passport” and “address” types.

*Type* `telegram.DataCredentials`, optional

**front_side**

Credentials for encrypted document’s front side. Available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

*Type* `telegram.FileCredentials`, optional

**reverse_side**

Credentials for encrypted document’s reverse side. Available for “driver_license” and “identity_card”.

*Type* `telegram.FileCredentials`, optional

**selfie**

Credentials for encrypted selfie of the user with a document. Can be available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

*Type* `telegram.FileCredentials`, optional

**translation**


*Type* `List[telegram.FileCredentials]`, optional

**files**


*Type* `List[telegram.FileCredentials]`, optional

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

See `telegram.TelegramObject.de_json()`.

```python
to_dict()
```

See `telegram.TelegramObject.to_dict()`.

**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**

Checksum of encrypted file
```python

type str

secret
    Secret of encrypted file
    Type str

to_dict()
    See telegram.TelegramObject.to_dict().
```

**telegram.IdDocumentData**

class telegram.IdDocumentData(document_no, expiry_date, bot=None, **kwargs)
    Bases: telegram.base.TelegramObject
    This object represents the data of an identity document.

document_no
    Document number.
    Type str

expiry_date
    Optional. Date of expiry, in DD.MM.YYYY format.
    Type str

**telegram.PersonalDetails**

class telegram.PersonalDetails(first_name, last_name, birth_date, gender, country_code, residence_country_code, first_name_native=None, last_name_native=None, middle_name=None, middle_name_native=None, bot=None, **kwargs)
    Bases: telegram.base.TelegramObject
    This object represents personal details.

first_name
    First Name.
    Type str

middle_name
    Optional. First Name.
    Type str

last_name
    Last Name.
    Type str

birth_date
    Date of birth in DD.MM.YYYY format.
    Type str

gender
    Gender, male or female.
    Type str

country_code
    Citizenship (ISO 3166-1 alpha-2 country code).
    Type str
```
residence_country_code
Country of residence (ISO 3166-1 alpha-2 country code).

Type str

first_name_native
First Name in the language of the user’s country of residence.

Type str

middle_name_native
Optional. Middle Name in the language of the user’s country of residence.

Type str

last_name_native
Last Name in the language of the user’s country of residence.

Type str

telegram.ResidentialAddress

class telegram.ResidentialAddress(street_line1, street_line2, city, state, country_code,
post_code, bot=None, **kwargs)
Bases: telegram.base.TelegramObject
This object represents a residential address.

street_line1
First line for the address.

Type str

street_line2
Optional. Second line for the address.

Type str

city
City.

Type str

state
Optional. State.

Type str

country_code
ISO 3166-1 alpha-2 country code.

Type str

post_code
Address post code.

Type str
**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.

**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.

**Parameters**

- **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** (telegram.Bot, optional) – The Bot to use for instance methods.

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

**data**

Array with encrypted information about documents and other Telegram Passport elements that was shared with the bot.

**Type** List[telegram.EncryptedPassportElement]

**credentials**

Encrypted credentials.

**Type** telegram.EncryptedCredentials

**bot**

The Bot to use for instance methods.

**Type** telegram.Bot, optional

**classmethod de_json**(data, bot)

See telegram.TelegramObject.de_json().

**property 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.TelgramDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

**Type** telegram.Credentials

**property decrypted_data**

Lazily decrypt and return information about documents and other Telegram Passport elements which were shared with the bot.

**Raises** telegram.TelgramDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

**Type** List[telegram.EncryptedPassportElement]

**to_dict()**

See telegram.TelegramObject.to_dict().
telegram.PassportFile

class telegram.PassportFile(file_id, file_unique_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.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their file_unique_id is equal.

Parameters

- **file_id**(str) – Identifier for this file, which can be used to download or reuse the file.
- **file_unique_id**(str) – Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the 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.

file_id

Identifier for this file.

Type str

file_unique_id

Unique identifier for this file, which is supposed to be the same over time and for different bots. Can’t be used to download or reuse the file.

Type str

file_size

File size.

Type int

file_date

Unix time when the file was uploaded.

Type int

bot

Optional. The Bot to use for instance methods.

Type telegram.Bot

classmethod de_json_decrypted(data, bot, credentials)

Variant of telegram.TelegramObject.de_json() that also takes into account passport credentials.

Parameters

- **data**(Dict[str,...]) – The JSON data.
- **bot**(telegram.Bot) – The bot associated with this object.
- **credentials**(telegram.FileCredentials) – The credentials

Returns

Return type telegram.PassportFile
classmethod de_list_decrypted(data, bot, credentials)

Variant of `telegram.TelegramObject.de_list()` that also takes into account passport credentials.

**Parameters**

- `data` (Dict[str,...]) – The JSON data.
- `bot` (telegram.Bot) – The bot associated with these objects.
- `credentials` (telegram.FileCredentials) – The credentials

**Returns**

**Return type** List[telegram.PassportFile]

get_file(timeout=None, api_kwargs=None)


For the documentation of the arguments, please see `telegram.Bot.get_file()`.

**Returns** `telegram.File`

**Raises** `telegram.error.TelegramError`

**telegram.EncryptedPassportElement**

class `telegram.EncryptedPassportElement` (type, data=None, phone_number=None, email=None, files=None, front_side=None, reverse_side=None, selfie=None, translation=None, hash=None, bot=None, credentials=None, **_kwargs)

Bases: telegram.base.TelegramObject

Contains information about documents or other Telegram Passport elements shared with the bot by the user. The data has been automatically decrypted by python-telegram-bot.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `type, data, phone_number, email, files, front_side, reverse_side and selfie` are equal.

**Note:** This object is decrypted only when originating from `telegram.PassportData.decrypted_data`.

**Parameters**

- `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.

**type**


  Type str

**data**

Optional. Decrypted or encrypted data, available for “personal_details”, “passport”, “driver_license”, “identity_card”, “identity_passport” and “address” types.

  Type telegram.PersonalDetails | telegram.IdDocument | telegram.ResidentialAddress | str

**phone_number**

Optional. User’s verified phone number, available only for “phone_number” type.

  Type str

**email**

Optional. User’s verified email address, available only for “email” type.

  Type str

**files**

Optional. Array of encrypted/decrypted files with documents provided by the user, available for “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration” and “temporary_registration” types.

  Type List[telegram.PassportFile]

**front_side**

Optional. Encrypted/decrypted file with the front side of the document, provided by the user. Available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

  Type telegram.PassportFile
Optional. Encrypted/decrypted file with the reverse side of the document, provided by the user. Available for “driver_license” and “identity_card”.

Type `telegram.PassportFile`

Optional. Encrypted/decrypted file with the selfie of the user holding a document, provided by the user; available for “passport”, “driver_license”, “identity_card” and “internal_passport”.

Type `telegram.PassportFile`


Type `List[telegram.PassportFile]`

Base64-encoded element hash for using in `telegram.PassportElementErrorUnspecified`.

Type `str`

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

See `telegram.TelegramObject.de_json()`.

Variant of `telegram.TelegramObject.de_json()` that also takes into account passport credentials.

Parameters

- `data` (Dict[str, ...]) – The JSON data.
- `bot` (`telegram.Bot`) – The bot associated with this object.
- `credentials` (`telegram.FileCredentials`) – The credentials

Returns

Return type `telegram.EncryptedPassportElement`

See `telegram.TelegramObject.to_dict()`.

Contains data required for decrypting and authenticating EncryptedPassportElement. See the Telegram Passport Documentation for a complete description of the data decryption and authentication processes.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their `data`, `hash` and `secret` are equal.

Note: This object is decrypted only when originating from `telegram.PassportData.decrypted_credentials`. 
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.

**data**
Decrypted data with unique user’s nonce, data hashes and secrets used for EncryptedPassportElement decryption and authentication or base64 encrypted data.

*Type* telegram.Credentials or str

**hash**
Base64-encoded data hash for data authentication.

*Type* str

**secret**
Decrypted or encrypted secret used for decryption.

*Type* str

**property decrypted_data**
Lazily decrypt and return credentials data. This object also contains the user specified nonce as decrypted_data.nonce.

*Raises* telegram.TelegramDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

*Type* telegram.Credentials

**property decrypted_secret**
Lazily decrypt and return secret.

*Raises* telegram.TelegramDecryptionError – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

*Type* str

### 3.2.80** utils

**telegram.utils.helpers Module**

This module contains helper functions.

**telegram.utils.helpers.DEFAULT_20**: telegram.utils.helpers.DefaultValue = 20

Default 20

*Type* DefaultValue

**telegram.utils.helpers.DEFAULT_FALSE**: telegram.utils.helpers.DefaultValue = False

Default False

*Type* DefaultValue

**telegram.utils.helpers.DEFAULT_NONE**: telegram.utils.helpers.DefaultValue = None

Default None

*Type* DefaultValue
class telegram.utils.helpers.DefaultValue(value=None)

Bases: Generic[telegram.utils.helpers.DVType]

Wrapper for immutable default arguments that allows to check, if the default value was set explicitly. Usage:

```python
DefaultOne = DefaultValue(1)
def f(arg=DefaultOne):
    if arg is DefaultOne:
        print('arg is the default')
        arg = arg.value
    else:
        print('arg was set explicitly')
        print(f'arg = {str(arg)!r}')
```

This yields:

```python
>>> f()
'arg' is the default
'arg' = 1
>>> f(1)
'arg' was set explicitly
'arg' = 1
>>> f(2)
'arg' was set explicitly
'arg' = 2
```

Also allows to evaluate truthiness:

```python
default = DefaultValue(value)
if default:
    ...
```

is equivalent to:

```python
default = DefaultValue(value)
if value:
    ...
```

repr(DefaultValue(value)) returns repr(value) and str(DefaultValue(value)) returns f'DefaultValue({value})'.

**Parameters**

- **value (obj)** – The value of the default argument

**value**

The value of the default argument

**Type**

obj

**static get_value (obj)**

Shortcut for:

```python
return obj.value if isinstance(obj, DefaultValue) else obj
```

**Parameters**

- **obj (object)** – The object to process

**Returns**

The value

**Return type**

Same type as input, or the value of the input

telegram.utils.helpers.create_deep_linked_url(bot_username, payload=None, group=False)

Creates a deep-linked URL for this bot_username with the specified payload. See https://core.telegram.org/bots#deep-linking to learn more.

The payload may consist of the following characters: A-Z, a-z, 0-9, _, -
Note: Works well in conjunction with `CommandHandler("start", callback, filters = Filters.regex('payload'))`

Examples

```python
create_deep_linked_url(bot.get_me().username, "some-params")
```

Parameters

- `bot_username` (str) – The username to link to
- `payload` (str, optional) – Parameters to encode in the created URL
- `group` (bool, optional) – If True the user is prompted to select a group to add the bot to. If False, opens a one-on-one conversation with the bot. Defaults to False.

Returns An URL to start the bot with specific parameters

Return type str

telegram.utils.helpers.decode_conversations_from_json(json_string)
Helper method to decode a conversations dict (that uses tuples as keys) from a JSON-string created with `encode_conversations_to_json()`.

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

Returns The conversations dict after decoding

Return type dict

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

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

Returns The user/chat_data defaultdict after decoding

Return type dict

telegram.utils.helpers.effective_message_type(entity)
Extracts the type of message as a string identifier from a `telegram.Message` or a `telegram.Update`.

Parameters `entity` (`telegram.Update` | `telegram.Message`) – The update or message to extract from.

Returns One of `Message.MESSAGE_TYPES`

Return type str

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

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

Returns The JSON-serialized conversations dict

Return type str

telegram.utils.helpers.escape_markdown(text, version=1, entity_type=None)
Helper function to escape telegram markup symbols.

Parameters

- `text` (str) – The text.
• **version** (int | str) – Use to specify the version of telegrams Markdown. Either 1 or 2. Defaults to 1.

• **entity_type** (str, optional) – For the entity types PRE, CODE and the link part of TEXT_LINKS, only certain characters need to be escaped in MarkdownV2. See the official API documentation for details. Only valid in combination with version=2, will be ignored else.

`telegram.utils.helpers.from_timestamp(unixtime, tzinfo=<UTC>)`

Converts an (integer) unix timestamp to a timezone aware datetime object. None s are left alone (i.e. from_timestamp(\(\text{None}\)) is None).

**Parameters**

• **unixtime** (int) – Integer POSIX timestamp.

• **tzinfo** (datetime.tzinfo, optional) – The timezone to which the timestamp is to be converted to. Defaults to UTC.

**Returns** Timezone aware equivalent datetime.datetime value if unixtime is not None; else None.

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

Returns the signal name of the given signal number.

`telegram.utils.helpers.is_local_file(obj)`

Checks if a given string is a file on local system.

**Parameters** **obj** (str) – The string to check.

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

**Parameters**

• **user_id** (int) – The user’s id which you want to mention.

• **name** (str) – The name the mention is showing.

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

**Return type** str

`telegram.utils.helpers.mention_markdown(user_id, name, version=1)`

**Parameters**

• **user_id** (int) – The user’s id which you want to mention.

• **name** (str) – The name the mention is showing.

• **version** (int | str) – Use to specify the version of Telegram’s Markdown. Either 1 or 2. Defaults to 1.

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

**Return type** str

`telegram.utils.helpers.parse_file_input(file_input, tg_type=None, attach=None, file-name=None)`

Parses input for sending files:

• For string input, if the input is an absolute path of a local file, adds the file:// prefix. If the input is a relative path of a local file, computes the absolute path and adds the file:// prefix. Returns the input unchanged, otherwise.

• `pathlib.Path` objects are treated the same way as strings.

• For IO and bytes input, returns an `telegram.InputFile`.

• If `tg_type` is specified and the input is of that type, returns the file_id attribute.

**Parameters**
• `file_input` (str | bytes | filelike object | Telegram media object) – The input to parse.

• `tg_type` (type, optional) – The Telegram media type the input can be. E.g. `telegram.Animation`.

• `attach` (bool, optional) – Whether this file should be send as one file or is part of a collection of files. Only relevant in case an `telegram.InputFile` is returned.

• `filename` (str, optional) – The filename. Only relevant in case an `telegram.InputFile` is returned.

Returns: The parsed input or the untouched `file_input`, in case it’s no valid file input.

Return type: str | `telegram.InputFile` | object

`telegram.utils.helpers.to_float_timestamp(time_object, reference_timestamp=None, tzinfo=None)`

Converts a given time object to a float POSIX timestamp. Used to convert different time specifications to a common format. The time object can be relative (i.e. indicate a time increment, or a time of day) or absolute. Object objects from the `datetime` module that are timezone-naive will be assumed to be in UTC, if `bot` is not passed or `bot.defaults` is None.

Parameters:

• `time_object` (int | float | `datetime.timedelta` | `datetime.datetime` | `datetime.time`) – Time value to convert. The semantics of this parameter will depend on its type:
  - int or float will be interpreted as “seconds from reference_t”
  - `datetime.timedelta` will be interpreted as “time increment from reference_t”
  - `datetime.datetime` will be interpreted as an absolute date/time value
  - `datetime.time` will be interpreted as a specific time of day

• `reference_timestamp` (float, optional) – POSIX timestamp that indicates the absolute time from which relative calculations are to be performed (e.g. when `t` is given as an int, indicating “seconds from reference_t”). Defaults to now (the time at which this function is called).

If `t` is given as an absolute representation of date & time (i.e. a `datetime.datetime` object), `reference_timestamp` is not relevant and so its value should be None. If this is not the case, a `ValueError` will be raised.

• `tzinfo` (pytz.BaseTzInfo, optional) – If `t` is a naive object from the `datetime` module, it will be interpreted as this timezone. Defaults to `pytz.utc`.

Note: Only to be used by `telegram.ext`.

Returns: The return value depends on the type of argument `t`. If `t` is given as a time increment (i.e. as an int, float or `datetime.timedelta`), then the return value will be `reference_t + t`.

Else if it is given as an absolute date/time value (i.e. a `datetime.datetime` object), the equivalent value as a POSIX timestamp will be returned.

Finally, if it is a time of the day without date (i.e. a `datetime.time` object), the return value is the nearest future occurrence of that time of day.

Return type: float | None

Raises
• **TypeError** – If `t`’s type is not one of those described above.

• **ValueError** – If `t` is a `datetime.datetime` and `reference_timestamp` is not `None`.

```python
telegram.utils.helpers.to_timestamp(dt_obj, reference_timestamp=None, tzinfo=None)
```

Wrapper over `to_float_timestamp()` which returns an integer (the float value truncated down to the nearest integer).

See the documentation for `to_float_timestamp()` for more details.

**telegram.utils.promise.Promise**

class `telegram.utils.promise.Promise`

Shortcut for `telegram.ext.utils.promise.Promise`.

Depreciated since version 13.2: Use `telegram.ext.utils.promise.Promise` instead.

**telegram.utils.request.Request**

class `telegram.utils.request.Request`

```python
(transport=Transport, con_pool_size=1, proxy_url=None, url=None, urlib3_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`.
- **urlib3_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. Defaults to `5.0`.
- **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. Defaults to `5.0`.

**property con_pool_size**

The size of the connection pool used.

```python
download(url, filename, timeout=None)
```

Download a file by its URL.

**Parameters**

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

```python
post(url, data, timeout=None)
```

Request an URL.
Parameters

- **url** *(str)* – The web location we want to retrieve.
- **data** *(Dict[str, str|int], optional)* – A dict of key/value pairs.
- **timeout** *(int|float, optional)* – If 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).

```
stop()
```

Performs cleanup on shutdown.

**telegram.utils.types Module**

This module contains custom typing aliases.

**telegram.utils.types.DVInput**

Generic type for bot method parameters which can have defaults. DVInput[type] is the same as Union[DefaultValue, type].

alias of Union[DefaultValue, DVType, DVT]

**telegram.utils.types.FileInput**

Valid input for passing files to Telegram. Either a file id as string, a file like object, a local file path as string, pathlib.Path or the file contents as bytes.

alias of Union[str, bytes, IO, InputFile, pathlib.Path]

**telegram.utils.types.FileLike**

Either an open file handler or a telegram.InputFile.

alias of Union[IO, InputFile]

**telegram.utils.types.JSONDict**

Dictionary containing response from Telegram or data to send to the API.

alias of Dict[str, Any]

**telegram.utils.types.ODVInput**

Generic type for bot method parameters which can have defaults. ODVInput[type] is the same as Optional[Union[DefaultValue, type]].

alias of Optional[Union[DefaultValue, DVType, DVT]]

**telegram.utils.types.SLT**

Single instance or list/tuple of instances.

alias of Union[RT, List[RT], Tuple[RT, ...]]
3.3 Changelog

3.3.1 Changelog

Version 13.11

Released 2022-02-02

This is the technical changelog for version 13.11. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes:

• Full Support for Bot API 5.7 (#2881)

Version 13.10

Released 2022-01-03

This is the technical changelog for version 13.10. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes:

• Full Support for API 5.6 (#2835)

Minor Changes & Doc fixes:

• Update Copyright to 2022 (#2836)
• Update Documentation of BotCommand (#2820)

Version 13.9

Released 2021-12-11

This is the technical changelog for version 13.9. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes:

• Full Support for Api 5.5 (#2809)

Minor Changes

• Adjust Automated Locking of Inactive Issues (#2775)

Version 13.8.1

Released 2021-11-08

This is the technical changelog for version 13.8.1. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Doc fixes:

• Add ChatJoinRequest(Handler) to Docs (#2771)
Version 13.8

Released 2021-11-08

This is the technical changelog for version 13.8. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes:

• Full support for API 5.4 (#2767)

Minor changes, CI improvements, Doc fixes and Type hinting:

• Create Issue Template Forms (#2689)
• Fix camelCase Functions in ExtBot (#2659)
• Fix Empty Captions not Being Passed by Bot.copy_message (#2651)
• Fix Setting Thumbs When Uploading A Single File (#2583)
• Fix Bug in BasePersistence.insert/replace_bot for Objects with __dict__ not in __slots__ (#2603)

Version 13.7

Released 2021-07-01

This is the technical changelog for version 13.7. More elaborate release notes can be found in the news channel @pythontelegrambotchannel.

Major Changes:

• Full support for Bot API 5.3 (#2572)

Bug Fixes:

• Fix Bug in BasePersistence.insert/replace_bot for Objects with __dict__ in their slots (#2561)
• Remove Incorrect Warning About Defaults and ExtBot (#2553)

Minor changes, CI improvements, Doc fixes and Type hinting:

• Type Hinting Fixes (#2552)
• Doc Fixes (#2551)
• Improve Deprecation Warning for __slots__ (#2574)
• Stabilize CI (#2575)
• Fix Coverage Configuration (#2571)
• Better Exception-Handling for BasePersistence.replace/insert_bot (#2564)
• Remove Deprecated pass_args from Deeplinking Example (#2550)
Version 13.6

Released 2021-06-06

New Features:

- Arbitrary `callback_data` (#1844)
- Add `ContextTypes` & `BasePersistence.refresh_user/chat/bot_data` (#2262)
- Add `Filters.attachment` (#2528)
- Add `pattern Argument` to `ChosenInlineResultHandler` (#2517)

Major Changes:

- Add `slots` (#2345)

Minor changes, CI improvements, Doc fixes and Type hinting:

- Doc Fixes (#2495, #2510)
- Add `max_connections Parameter` to `Updater.start_webhook` (#2547)
- Fix for `Promise.done_callback` (#2544)
- Improve Code Quality (#2536, #2454)
- Increase Test Coverage of `CallbackQueryHandler` (#2520)
- Stabilize CI (#2522, #2537, #2541)
- Fix `send_phone_number_to_provider argument` for `Bot.send_invoice` (#2527)
- Handle Classes as Input for `BasePersistence.replace/insert_bot` (#2523)
- Bump Tornado Version and Remove Workaround from #2067 (#2494)

Version 13.5

Released 2021-04-30

Major Changes:

- Full support of Bot API 5.2 (#2489).

Note: The `start_parameter` argument of `Bot.send_invoice` and the corresponding shortcuts is now optional, so the order of parameters had to be changed. Make sure to update your method calls accordingly.

- Update `ChatActions`, Deprecating `ChatAction.RECORD_AUDIO` and `ChatAction.UPLOAD_AUDIO` (#2460)

New Features:

- Convenience Utilities & Example for Handling `ChatMemberUpdated` (#2490)
- `Filters.forwarded_from` (#2446)

Minor changes, CI improvements, Doc fixes and Type hinting:

- Improve Timeouts in `ConversationHandler` (#2417)
- Stabilize CI (#2480)
- Doc Fixes (#2437)
- Improve Type Hints of Data Filters (#2456)
- Add Two `UserWarnings` (#2464)

3.3. Changelog
• Improve Code Quality (#2450)
• Update Fallback Test-Bots (#2451)
• Improve Examples (#2441, #2448)

Version 13.4.1

Released 2021-03-14

Hot fix release:
• Fixed a bug in setup.py (#2431)

Version 13.4

Released 2021-03-14

Major Changes:
• Full support of Bot API 5.1 (#2424)

Minor changes, CI improvements, doc fixes and type hinting:
• Improve Updater.set_webhook (#2419)
• Doc Fixes (#2404)
• Type Hinting Fixes (#2425)
• Update pre-commit Settings (#2415)
• Fix Logging for Vendored urllib3 (#2427)
• Stabilize Tests (#2409)

Version 13.3

Released 2021-02-19

Major Changes:
• Make cryptography Dependency Optional & Refactor Some Tests (#2386, #2370)
• Deprecate MessageQueue (#2393)

Bug Fixes:
• Refactor Defaults Integration (#2363)
• Add Missing telegram.SecureValue to init and Docs (#2398)

Minor changes:
• Doc Fixes (#2359)
Version 13.2

Released 2021-02-02

Major Changes:

- Introduce python-telegram-bot-raw (#2324)
- Explicit Signatures for Shortcuts (#2240)

New Features:

- Add Missing Shortcuts to Message (#2330)
- Rich Comparison for Bot (#2320)
- Add run_async Parameter to ConversationHandler (#2292)
- Add New Shortcuts to Chat (#2291)
- Add New Constant MAX_ANSWER_CALLBACK_QUERY_TEXT_LENGTH (#2282)
- Allow Passing Custom Filename For All Media (#2249)
- Handle Bytes as File Input (#2233)

Bug Fixes:

- Fix Escaping in Nested Entities in Message Properties (#2312)
- Adjust Calling of Dispatcher.update_persistence (#2285)
- Add quote kwarg to Message.reply_copy (#2232)
- ConversationHandler: Docs & edited_channel_post behavior (#2339)

Minor changes, CI improvements, doc fixes and type hinting:

- Doc Fixes (#2253, #2225)
- Reduce Usage of typing.Any (#2321)
- Extend Deeplinking Example (#2235)
- Add pyupgrade to pre-commit Hooks (#2301)
- Add PR Template (#2299)
- Drop Nightly Tests & Update Badges (#2323)
- Update Copyright (#2289, #2287)
- Change Order of Class DocStrings (#2256)
- Add macOS to Test Matrix (#2266)
- Start Using Versioning Directives in Docs (#2252)
- Improve Annotations & Docs of Handlers (#2243)

Version 13.1

Released 2020-11-29

Major Changes:

- Full support of Bot API 5.0 (#2181, #2186, #2190, #2189, #2183, #2184, #2188, #2185, #2192, #2196, #2193, #2223, #2199, #2187, #2147, #2205)

New Features:

- Add Defaults.run_async (#2210)
- Improve and Expand CallbackQuery Shortcuts (#2172)
• Add XOR Filters and make Filters.name a Property (#2179)
• Add Filters.document.file_extension (#2169)
• Add Filters.caption_regex(#2163)
• Add Filters.chat_type (#2128)
• Handle Non-Binary File Input (#2202)

Bug Fixes:
• Improve Handling of Custom Objects in BasePersistence.insert/replace_bot (#2151)
• Fix bugs in replace/insert_bot (#2218)

Minor changes, CI improvements, doc fixes and type hinting:
• Improve Type hinting (#2204, #2118, #2167, #2136)
• Doc Fixes & Extensions (#2201, #2161)
• Use F-Strings Where Possible (#2222)
• Rename kwargs to _kwargs where possible (#2182)
• Comply with PEP561 (#2168)
• Improve Code Quality (#2131)
• Switch Code Formatting to Black (#2122, #2159, #2158)
• Update Wheel Settings (#2142)
• Update timerbot.py to v13.0 (#2149)
• Overhaul Constants (#2137)
• Add Python 3.9 to Test Matrix (#2132)
• Switch Codecov to GitHub Action (#2127)
• Specify Required pytz Version (#2121)

Version 13.0

Released 2020-10-07

For a detailed guide on how to migrate from v12 to v13, see this wiki page.

Major Changes:
• Deprecate old-style callbacks, i.e. set use_context=True by default (#2050)
• Refactor Handling of Message VS Update Filters (#2032)
• Deprecate Message.default_quote (#1965)
• Refactor persistence of Bot instances (#1994)
• Refactor JobQueue (#1981)
• Refactor handling of kwargs in Bot methods (#1924)
• Refactor Dispatcher.run_async, deprecating the @run_async decorator (#2051)

New Features:
• Type Hinting (#1920)
• Automatic Pagination for answer_inline_query (#2072)
• Defaults.tzinfo (#2042)
• Extend rich comparison of objects (#1724)
• Add Filters.via_bot (#2009)
• Add missing shortcuts (#2043)
• Allow DispatcherHandlerStop in ConversationHandler (#2059)
• Make Errors picklable (#2106)

Minor changes, CI improvements, doc fixes or bug fixes:
• Fix Webhook not working on Windows with Python 3.8+ (#2067)
• Fix setting thumbs with send_media_group (#2093)
• Make MessageHandler filter for Filters.update first (#2085)
• Fix PicklePersistence.flush() with only bot_data (#2017)
• Add test for clean argument of Updater.start_polling/webhook (#2002)
• Doc fixes, refinements and additions (#2005, #2008, #2089, #2094, #2090)
• CI fixes (#2018, #2061)
• Refine pollbot.py example (#2047)
• Refine Filters in examples (#2027)
• Rename echobot examples (#2025)
• Use Lock-Bot to lock old threads (#2048, #2052, #2049, #2053)

Version 12.8

Released 2020-06-22

Major Changes:
• Remove Python 2 support (#1715)
• Bot API 4.9 support (#1980)
• IDs/Username of Filters.user and Filters.chat can now be updated (#1757)

Minor changes, CI improvements, doc fixes or bug fixes:
• Update contribution guide and stale bot (#1937)
• Remove NullHandlers (#1913)
• Improve and expand examples (#1943, #1995, #1983, #1997)
• Doc fixes (#1940, #1962)
• Add User.send_poll() shortcut (#1968)
• Ignore private attributes in TelegramObject.to_dict() (#1989)
• Stabilize CI (#2000)
Version 12.7

Released 2020-05-02

Major Changes:

- Bot API 4.8 support. Note: The Dice object now has a second positional argument emoji. This is relevant, if you instantiate Dice objects manually. (#1917)

- Added tzinfo argument to helpers.from_timestamp. It now returns an timezone aware object. This is relevant for Message.(date,forward_date,edit_date), Poll.close_date and ChatMember.until_date (#1621)

New Features:

- New method run_monthly for the JobQueue (#1705)

- Job.next_t now gives the datetime of the jobs next execution (#1685)

Minor changes, CI improvements, doc fixes or bug fixes:

- Stabilize CI (#1919, #1931)

- Use ABCs @abstractmethod instead of raising NotImplementedError for Handler, BasePersistence and BaseFilter (#1905)

- Doc fixes (#1914, #1902, #1910)

Version 12.6.1

Released 2020-04-11

Bug fixes:

- Fix serialization of reply_markup in media messages (#1889)

Version 12.6

Released 2020-04-10

Major Changes:

- Bot API 4.7 support. Note: In Bot.create_new_sticker_set and Bot.add_sticker_to_set, the order of the parameters had be changed, as the png_sticker parameter is now optional. (#1858)

Minor changes, CI improvements or bug fixes:

- Add tests for switch_inline_query(_current_chat) with empty string (#1635)

- Doc fixes (#1854, #1874, #1884)

- Update issue templates (#1880)

- Favor concrete types over “Iterable” (#1882)

- Pass last valid CallbackContext to TIMEOUT handlers of ConversationHandler (#1826)

- Tweak handling of persistence and update persistence after job calls (#1827)

- Use checkout@v2 for GitHub actions (#1887)
Version 12.5.1

Released 2020-03-30

Minor changes, doc fixes or bug fixes:
- Add missing docs for PollHandler and PollAnswerHandler (#1853)
- Fix wording in Filters docs (#1855)
- Reorder tests to make them more stable (#1835)
- Make ConversationHandler attributes immutable (#1756)
- Make PrefixHandler attributes command and prefix editable (#1636)
- Fix UTC as default tzinfo for Job (#1696)

Version 12.5

Released 2020-03-29

New Features:
- Bot.link gives the t.me link of the bot (#1770)

Major Changes:
- Bot API 4.5 and 4.6 support. (#1508, #1723)

Minor changes, CI improvements or bug fixes:
- Remove legacy CI files (#1783, #1791)
- Update pre-commit config file (#1787)
- Remove builtin names (#1792)
- CI improvements (#1808, #1848)
- Support Python 3.8 (#1614, #1824)
- Use stale bot for auto closing stale issues (#1820, #1829, #1840)
- Doc fixes (#1778, #1818)
- Fix typo in edit_message_media (#1779)
- In examples, answer CallbackQueries and use edit_message_text shortcut (#1721)
- Revert accidental change in vendored urllib3 (#1775)

Version 12.4.2

Released 2020-02-10

Bug Fixes
- Pass correct parse_mode to InlineResults if bot.defaults is None (#1763)
- Make sure PP can read files that dont have bot_data (#1760)
Version 12.4.1

Released 2020-02-08

This is a quick release for #1744 which was accidently left out of v12.4.0 though mentioned in the release notes.

Version 12.4.0

Released 2020-02-08

New features:

- Set default values for arguments appearing repeatedly. We also have a wiki page for the new defaults. (#1490)
- Store data in `CallbackContext.bot_data` to access it in every callback. Also persists. (#1325)
- `Filters.poll` allows only messages containing a poll (#1673)

Major changes:

- `Filters.text` now accepts messages that start with a slash, because `CommandHandler` checks for `MessageEntity.BOT_COMMAND` since v12. This might lead to your MessageHandlers receiving more updates than before (#1680).
- `Filters.command` now checks for `MessageEntity.BOT_COMMAND` instead of just a leading slash. Also by `Filters.command(False)` you can now filters for messages containing a command anywhere in the text (#1744).

Minor changes, CI improvements or bug fixes:

- Add `dispatcher` argument to `Updater` to allow passing a customized `Dispatcher` (#1484)
- Add missing names for `Filters` (#1632)
- Documentation fixes (#1624, #1647, #1669, #1703, #1718, #1734, #1740, #1642, #1739, #1746)
- CI improvements (#1716, #1731, #1738, #1748, #1749, #1750, #1752)
- Fix spelling issue for `encode_conversations_to_json` (#1661)
- Remove double assignement of `Dispatcher.job_queue` (#1698)
- Expose dispatcher as property for `CallbackContext` (#1684)
- Fix `None` check in `JobQueue._put()` (#1707)
- Log datetimes correctly in `JobQueue` (#1714)
- Fix false `Message.link` creation for private groups (#1741)
- Add option `--with-upstream-urllib3` to `setup.py` to allow using non-vendored version (#1725)
- Fix persistence for nested `ConversationHandlers` (#1679)
- Improve handling of non-decodable server responses (#1623)
- Fix download for files without `file_path` (#1591)
- `test_webhook_invalid_posts` is now considered flaky and retried on failure (#1758)
Version 12.3.0

Released 2020-01-11

New features:

- Filters.caption allows only messages with caption (#1631).
- Filter for exact messages/captions with new capability of Filters.text and Filters.caption. Especially useful in combination with ReplyKeyboardMarkup. (#1631).

Major changes:

- Fix inconsistent handling of naive datetimes (#1506).

Minor changes, CI improvements or bug fixes:

- Documentation fixes (#1558, #1569, #1579, #1572, #1566, #1577, #1656).
- Add mutex protection on ConversationHandler (#1533).
- Add MAX_PHOTOSIZE_UPLOAD constant (#1560).
- Add args and kwargs to Message.forward() (#1574).
- Transfer to GitHub Actions CI (#1555, #1556, #1605, #1606, #1607, #1612, #1615, #1645).
- Fix deprecation warning with Py3.8 by vendored urllib3 (#1618).
- Simplify assignments for optional arguments (#1600)
- Allow private groups for Message.link (#1619).
- Fix wrong signature call for ConversationHandler.TIMEOUT handlers (#1653).

Version 12.2.0

Released 2019-10-14

New features:

- Nested ConversationHandlers (#1512).

Minor changes, CI improvements or bug fixes:

- Fix CI failures due to non-backward compat attrs dependency (#1540).
- travis.yml: TEST_OFFICIAL removed from allowed_failures.
- Fix typos in examples (#1537).
- Fix Bot.to_dict to use proper first_name (#1525).
- Refactor test_commandhandler.py (#1408).
- Add Python 3.8 (RC version) to Travis testing matrix (#1543).
- test_bot.py: Add to_dict test (#1544).
- Flake config moved into setup.cfg (#1546).
Version 12.1.1

Released 2019-09-18

Hot fix release
Fixed regression in the vendored urllib3 (#1517).

Version 12.1.0

Released 2019-09-13

Major changes:
- Bot API 4.4 support (#1464, #1510)
- Add `get_file` method to Animation & ChatPhoto. Add, `get_small_file` & `get_big_file` methods to ChatPhoto (#1489)
- Tools for deep linking (#1049)

Minor changes and/or bug fixes:
- Documentation fixes (#1500, #1499)
- Improved examples (#1502)

Version 12.0.0

Released 2019-08-29

Well... This felt like decades. But here we are with a new release.
Expect minor releases soon (mainly complete Bot API 4.4 support)

Major and/or breaking 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
- Bot API 4.1, 4.2, 4.3 support
- Python 3.4 is no longer supported
- Error Handler now handles all types of exceptions (#1485)
- Return UTC from from_timestamp() (#1485)

See the wiki page at https://git.io/fxJuV for a detailed guide on how to migrate from version 11 to version 12.
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.

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

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 `"!"`).

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!
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)

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)
- Documentation fixes (#1348, #1397, #1436)
- Merged filters short-circuit (#1350)
- Fix webhook listen with tornado (#1383)
- Call `task_done()` on update queue after update processing finished (#1428)
- Fix `send_location()` - latitude may be 0 (#1437)
- Make `MessageEntity` objects comparable (#1465)
- Add prefix to thread names (#1358)
Buf fixes since v12.0.0b1

- Fix setting bot on ShippingQuery (#1355)
- Fix _trigger_timeout() missing 1 required positional argument: ‘job’ (#1367)
- Add missing message.text check in PrefixHandler check_update (#1375)
- Make updates persist even on DispatcherHandlerStop (#1463)
- Dispatcher force updating persistence object’s chat data attribute(#1462)

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.
- test_official is now executed in a different job

Version 11.1.0

Released 2018-09-01

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

Version 11.0.0

Released 2018-08-29

Fully support Bot API version 4.0! (also some bugfixes :))

Telegram Passport (#1174):

- Add full support for telegram passport.
  - New bot method: set_passport_data_errors
  - New filter: Filters.passport_data
  - Field passport_data field on Message
  - PassportData can be easily decrypted.
  - PassportFiles are automatically decrypted if originating from decrypted PassportData.
• See new passportbot.py example for details on how to use, or go to our telegram passport wiki page for more info
• NOTE: Passport decryption requires new dependency cryptography.

Inputfile rework (#1184):
• Change how Inputfile is handled internally
• This allows support for specifying the thumbnails of photos and videos using the thumb= argument in the different send_ methods.
• Also allows Bot.send_media_group to actually finally send more than one media.
• Add thumb to Audio, Video and Videonote
• Add Bot.edit_message_media together with InputMediaAnimation, InputMediaAudio, and inputMediaDocument.

Other Bot API 4.0 changes:
• Add 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.sendAnimation, 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)

**Version 10.1.0**

Released 2018-05-02

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)
Version 10.0.2

Released 2018-04-17

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.

Version 10.0.1

Released 2018-03-05

Fixes:

- Fix conversationhandler timeout (PR #1032)
- Add missing docs utils (PR #912)

Version 10.0.0

Released 2018-03-02

Non backward compatabile 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 downloaded 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)

Version 9.0.0

Released 2017-12-08

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.

Version 8.1.1

Released 2017-10-15

• Fix CommandHandler crashing on single character messages (PR #873).
Version 8.1.0

Released 2017-10-14

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 unittest improvements. - Documentation fixes.

Version 8.0.0

Released 2017-09-01

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.

Version 7.0.1

Released 2017-07-28

- Fix TypeError exception in RegexHandler (PR #751).
- Small documentation fix (PR #749).
Version 7.0.0

Released 2017-07-25

- 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.

Pre-version 7.0

2017-06-18

Released 6.1.0

- Fully support Bot API 3.0
- Add more fine-grained filters for status updates
- Bug fixes and other improvements

2017-05-29

Released 6.0.3

- Faulty PyPI release

2017-05-29

Released 6.0.2

- Avoid confusion with user’s urllib3 by renaming vendored urllib3 to ptb.urllib3

2017-05-19

Released 6.0.1

- Add support for User.language_code
- Fix Message.text_html and Message.text_markdown for messages with emoji

2017-05-19

Released 6.0.0

- Add support for Bot API 2.3.1
- Add support for deleteMessage API method
• New, simpler API for JobQueue - https://github.com/python-telegram-bot/python-telegram-bot/pull/484
• Download files into file-like objects - https://github.com/python-telegram-bot/python-telegram-bot/pull/459
• Use vendor urllib3 to address issues with timeouts - The default timeout for messages is now 5 seconds. For sending media, the default timeout is now 20 seconds.
• String attributes that are not set are now None by default, instead of empty strings
• Add text_markdown and text_html properties to Message - https://github.com/python-telegram-bot/python-telegram-bot/pull/507
• Add support for Socks5 proxy - https://github.com/python-telegram-bot/python-telegram-bot/pull/518
• Add support for filters in CommandHandler - https://github.com/python-telegram-bot/python-telegram-bot/pull/536
• Add the ability to invert (not) filters - https://github.com/python-telegram-bot/python-telegram-bot/pull/552
• Add Filters.group and Filters.private
• Compatibility with GAE via urllib3.contrib package - https://github.com/python-telegram-bot/python-telegram-bot/pull/583
• Add equality rich comparison operators to telegram objects - https://github.com/python-telegram-bot/python-telegram-bot/pull/604
• Several bugfixes and other improvements
• Remove some deprecated code

2017-04-17

Released 5.3.1
• Hotfix release due to bug introduced by urllib3 version 1.21

2016-12-11

Released 5.3
• Implement API changes of November 21st (Bot API 2.3)
• JobQueue now supports datetime.timedelta in addition to seconds
• JobQueue now supports running jobs only on certain days
• New Filters.reply filter
• Bugfix for Message.edit_reply_markup
• Other bugfixes

2016-10-25

Released 5.2
• Implement API changes of October 3rd (games update)
• Add Message.edit_* methods
• Filters for the MessageHandler can now be combined using bitwise operators (∧ and |)
• Add a way to save user- and chat-related data temporarily
• Other bugfixes and improvements

2016-09-24

Released 5.1
• Drop Python 2.6 support
• Deprecate telegram.Emoji
• Use ujson if available
• Add instance methods to Message, Chat, User, InlineQuery and CallbackQuery
• RegEx filtering for CallbackQueryHandler and InlineQueryHandler
• New MessageHandler filters: forwarded and entity
• Add Message.get_entity to correctly handle UTF-16 codepoints and MessageEntity offsets
• Fix bug in ConversationHandler when first handler ends the conversation
• Allow multiple Dispatcher instances
• Add ChatMigrated Exception
• Properly split and handle arguments in CommandHandler

2016-07-15

Released 5.0
• Rework JobQueue
• Introduce ConversationHandler
• Introduce telegram.constants - https://github.com/python-telegram-bot/python-telegram-bot/pull/342

2016-07-12

Released 4.3.4
• Fix proxy support with urllib3 when proxy requires auth

2016-07-08

Released 4.3.3
• Fix proxy support with urllib3

2016-07-04

Released 4.3.2
• Fix: Use timeout parameter in all API methods

2016-06-29

Released 4.3.1
• Update wrong requirement: urllib3>=1.10

2016-06-28

Released 4.3
• Use urllib3.PoolManager for connection re-use
• Rewrite run_async decorator to re-use threads
• New requirements: urllib3 and certifi

2016-06-10

Released 4.2.1
• Fix CallbackQuery.to_dict() bug (thanks to @jlmadurga)
• Fix editMessageText exception when receiving a CallbackQuery

2016-05-28

Released 4.2
• Implement Bot API 2.1
• Move botan module to telegram.contrib
• New exception type: BadRequest

2016-05-22

Released 4.1.2

• Fix MessageEntity decoding with Bot API 2.1 changes

2016-05-16

Released 4.1.1

• Fix deprecation warning in Dispatcher

2016-05-15

Released 4.1

• Implement API changes from May 6, 2016
• Fix bug when start_polling with clean=True
• Methods now have snake_case equivalent, for example telegram.Bot.send_message is the same as telegram.Bot.sendMessage

2016-05-01

Released 4.0.3

• Add missing attribute location to InlineQuery

2016-04-29

Released 4.0.2

• Bugfixes
  • KeyboardReplyMarkup now accepts str again

2016-04-27

Released 4.0.1

• Implement Bot API 2.0
• Almost complete recode of Dispatcher
• Please read the Transition Guide to 4.0
• Changes from 4.0rc1
  • The syntax of filters for MessageHandler (upper/lower cases)
  • Handler groups are now identified by int only, and ordered

• Note: v4.0 has been skipped due to a PyPI accident

2016-04-22

Released 4.0rc1

• Implement Bot API 2.0
• Almost complete recode of Dispatcher
• Please read the 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.sendDocument(..., filename='')
• Fix Sticker as InputFile
• Send JSON requests over urlencoded post data
• Markdown support for Bot.sendMessage(..., parse_mode=ParseMode.MARKDOWN)
• Refactor of TelegramError class (no more handling IOError or URLError)

2015-09-05

Released 2.8.2
• Fix regression on Telegram ReplyMarkup
• Add certificate to is_inputfile method
2015-09-05
Released 2.8.1
- Fix regression on Telegram objects with thumb properties

2015-09-04
Released 2.8
- TelegramError when chat_id is empty for send* methods
- setWebhook now supports sending self-signed certificate
- Huge redesign of existing Telegram classes
- Added support for PyPy
- Added docstring for existing classes

2015-08-19
Released 2.7.1
- Fixed JSON serialization for message

2015-08-17
Released 2.7
- Added support for Voice object and sendVoice method
- Due backward compatibility performer or/and title will be required for sendAudio
- Fixed JSON serialization when forwarded message

2015-08-15
Released 2.6.1
- Fixed parsing image header issue on < Python 2.7.3

2015-08-14
Released 2.6.0
- Depreciation of require_authentication and clearCredentials methods
- Giving AUTHORS the proper credits for their contribution for this project
- Message.date and Message.forward_date are now datetime objects

2015-08-12
Released 2.5.3
- telegram.Bot now supports to be unpickled

2015-08-11
Released 2.5.2
- New changes from Telegram Bot API have been applied
- telegram.Bot now supports to be pickled
- Return empty str instead None when message.text is empty

2015-08-10
Released 2.5.1
- Moved from GPLv2 to LGPLv3

2015-08-09
Released 2.5
• Fixes logging calls in API

2015-08-08

Released 2.4
• Fixes Emoji class for Python 3
• PEP8 improvements

2015-08-08

Released 2.3
• Fixes ForceReply class
• Remove logging.basicConfig from library

2015-07-25

Released 2.2
• Allows debug=True when initializing telegram.Bot

2015-07-20

Released 2.1
• Fix to_dict for Document and Video

2015-07-19

Released 2.0
• Fixes bugs
• Improves __str__ over to_json()
• Creates abstract class TelegramObject

2015-07-15

Released 1.9
• Python 3 officially supported
• PEP8 improvements

2015-07-12

Released 1.8
• Fixes crash when replying an unicode text message (special thanks to JRoot3D)

2015-07-11

Released 1.7
• Fixes crash when username is not defined on chat (special thanks to JRoot3D)

2015-07-10

Released 1.6
• Improvements for GAE support

2015-07-10

Released 1.5
• Fixes randomly unicode issues when using InputFile

2015-07-10

Released 1.4
• requests lib is no longer required
• Google App Engine (GAE) is supported

2015-07-10

Published 1.3

• Added support to setWebhook (special thanks to macrojames)

2015-07-09

Published 1.2

• CustomKeyboard classes now available
• Emojis available
• PEP8 improvements

2015-07-08

Published 1.1

• PyPi package now available

2015-07-08

Published 1.0

• Initial checkin of python-telegram-bot
t

telegram.constants, 193
telegram.error, 204
telegram.ext.filters, 47

telegram.ext.utils.types, 92
telegram.utils.helpers, 344
telegram.utils.types, 350
Symbols

__call__() (telegram.ext.DelayQueue method), 24
__call__() (telegram.ext.MessageQueue method), 22
__init__() (telegram.ext.DelayQueue method), 24
__init__() (telegram.ext.MessageQueue method), 22
__weakref__ (telegram.ext.MessageQueue attribute), 22

A

add_bot_ids() (telegram.ext.filters.Filters.via_bot method), 61
add_chat_ids() (telegram.ext.filters.Filters.chat method), 49
add_chat_ids() (telegram.ext.filters.Filters.forwarded_from method), 54
add_chat_ids() (telegram.ext.filters.Filters.sender_chat method), 57
add_done_callback() (telegram.ext.utils.promise.Promise method), 92
add_error_handler() (telegram.ext.Dispatcher method), 11
add_handler() (telegram.ext.Dispatcher method), 11
add_sticker_to_set() (telegram.Bot method), 96
add_user_ids() (telegram.ext.filters.Filters.user method), 60
add_usernames() (telegram.ext.filters.Filters.chat method), 49
add_usernames() (telegram.ext.filters.Filters.forwarded_from method), 54
add_usernames() (telegram.ext.filters.Filters.sender_chat method), 57
add_usernames() (telegram.ext.filters.Filters.user method), 60
add_usernames() (telegram.ext.filters.Filters.via_bot method), 62
address (telegram.ChatLocation attribute), 178
address (telegram.InlineQueryResultVenue attribute), 307
address (telegram.InputVenueMessageContent attribute), 314
address (telegram.SecureData attribute), 335
addStickerToSet() (telegram.Bot method), 96
ADMINISTRATOR (telegram.ChatMember attribute), 183
add (telegram.ext.filters.Filters attribute), 48
ALL_CHAT_ADMINISTRATORS (telegram.BotCommandScope attribute), 154
ALL_EMOJI (telegram.Dice attribute), 202
ALL_GROUP_CHATS (telegram.BotCommandScope attribute), 154
ALL_PRIVATE_CHATS (telegram.BotCommandScope attribute), 154
ALL_TYPES (telegram.MessageEntity attribute), 246
ALWAYS (telegram.Update attribute), 259
allow_empty (telegram.ext.filters.Filters.chat attribute), 49
allow_empty (telegram.ext.filters.Filters.forwarded_from attribute), 54
allow_empty (telegram.ext.filters.Filters.sender_chat attribute), 57
allow_empty (telegram.ext.filters.Filters.user attribute), 60
allow_empty (telegram.ext.filters.Filters.via_bot attribute), 61
allow_reentry() (telegram.ext.ConversationHandler property), 41
allow_sending_without_reply() (telegram.Defaults property), 25
allowed_updates (telegram.WebhookInfo attribute), 276
allows_multiple_answers (telegram.Poll attribute), 248
amount (telegram.LabeledPrice attribute), 319
Animation (class in telegram), 93
animation (telegram.ext.filters.Filters attribute), 48
animation (telegram.Game attribute), 326
<table>
<thead>
<tr>
<th>Function/Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>animation</td>
<td>(telegram.Message attribute), 226</td>
</tr>
<tr>
<td>ANONYMOUS_ADMIN_ID</td>
<td>(in module telegram.constants), 194</td>
</tr>
<tr>
<td>answer()</td>
<td>(telegram.CallbackQuery method), 158</td>
</tr>
<tr>
<td>answer()</td>
<td>(telegram.InlineQuery method), 281</td>
</tr>
<tr>
<td>answer()</td>
<td>(telegram.PreCheckoutQuery method), 325</td>
</tr>
<tr>
<td>answer()</td>
<td>(telegram.ShippingQuery method), 324</td>
</tr>
<tr>
<td>answer_callback_query()</td>
<td>(telegram.Bot method), 98</td>
</tr>
<tr>
<td>answer_inline_query()</td>
<td>(telegram.Bot method), 98</td>
</tr>
<tr>
<td>answer_pre_checkout_query()</td>
<td>(telegram.Bot method), 99</td>
</tr>
<tr>
<td>answer_shipping_query()</td>
<td>(telegram.Bot method), 100</td>
</tr>
<tr>
<td>answerCallbackQuery()</td>
<td>(telegram.Bot method), 97</td>
</tr>
<tr>
<td>answerInlineQuery()</td>
<td>(telegram.Bot method), 97</td>
</tr>
<tr>
<td>answerPreCheckoutQuery()</td>
<td>(telegram.Bot method), 97</td>
</tr>
<tr>
<td>answerShippingQuery()</td>
<td>(telegram.Bot method), 97</td>
</tr>
<tr>
<td>ANY_CHAT_MEMBER</td>
<td>(telegram.ext.ChatMemberHandler attribute), 34</td>
</tr>
<tr>
<td>apk</td>
<td>(telegram.ext.filters.Filters attribute), 52</td>
</tr>
<tr>
<td>application</td>
<td>(telegram.ext.filters.Filters attribute), 51</td>
</tr>
<tr>
<td>approve()</td>
<td>(telegram.ChatJoinRequest method), 178</td>
</tr>
<tr>
<td>approve_chat_join_request()</td>
<td>(telegram.Bot method), 100</td>
</tr>
<tr>
<td>approve_join_request()</td>
<td>(telegram.Chat method), 165</td>
</tr>
<tr>
<td>approve_join_request()</td>
<td>(telegram.User method), 261</td>
</tr>
<tr>
<td>approveChatJoinRequest()</td>
<td>(telegram.Bot method), 100</td>
</tr>
<tr>
<td>arbitrary_callback_data()</td>
<td>(telegram.ext.ExtBot attribute), 5</td>
</tr>
<tr>
<td>args</td>
<td>(telegram.ext.CallbackContext attribute), 14</td>
</tr>
<tr>
<td>args</td>
<td>(telegram.ext.utils.promise.Promise attribute), 91</td>
</tr>
<tr>
<td>async_args</td>
<td>(telegram.ext.CallbackContext attribute), 14</td>
</tr>
<tr>
<td>async_kwargs</td>
<td>(telegram.ext.CallbackContext attribute), 14</td>
</tr>
<tr>
<td>attach</td>
<td>(telegram.InputFile attribute), 210</td>
</tr>
<tr>
<td>attachment</td>
<td>(telegram.ext.filters.Filters attribute), 48</td>
</tr>
<tr>
<td>Audio</td>
<td>(class in telegram), 94</td>
</tr>
<tr>
<td>audio</td>
<td>(telegram.ext.Filters attribute), 48, 51</td>
</tr>
<tr>
<td>audio</td>
<td>(telegram.Message attribute), 226</td>
</tr>
<tr>
<td>audio_duration</td>
<td>(telegram.InlineQueryResultAudio attribute), 284</td>
</tr>
<tr>
<td>audio_file_id</td>
<td>(telegram.InlineQueryResultCachedAudio attribute), 285</td>
</tr>
<tr>
<td>audio_url</td>
<td>(telegram.InlineQueryResultAudio attribute), 284</td>
</tr>
<tr>
<td>author_signature</td>
<td>(telegram.Message attribute), 228</td>
</tr>
<tr>
<td>B</td>
<td>BadRequest, 204</td>
</tr>
<tr>
<td>ban_chat()</td>
<td>(telegram.Chat method), 165</td>
</tr>
<tr>
<td>ban_chat_member()</td>
<td>(telegram.Bot method), 101</td>
</tr>
<tr>
<td>ban_chat_sender_chat()</td>
<td>(telegram.Bot method), 101</td>
</tr>
<tr>
<td>ban_member()</td>
<td>(telegram.Chat method), 165</td>
</tr>
<tr>
<td>ban_sender_chat()</td>
<td>(telegram.Chat method), 165</td>
</tr>
<tr>
<td>banChatMember()</td>
<td>(telegram.Bot method), 101</td>
</tr>
<tr>
<td>banChatSenderChat()</td>
<td>(telegram.Bot method), 101</td>
</tr>
<tr>
<td>bank_statement</td>
<td>(telegram.SecureData attribute), 335</td>
</tr>
<tr>
<td>BaseFilter</td>
<td>(class in telegram.ext.filters), 47</td>
</tr>
<tr>
<td>BasePersistence</td>
<td>(class in telegram.ext), 79</td>
</tr>
<tr>
<td>BASKETBALL</td>
<td>(telegram.Dice attribute), 202</td>
</tr>
<tr>
<td>basketball</td>
<td>(telegram.ext.filters.Filters attribute), 51</td>
</tr>
<tr>
<td>BD</td>
<td>(in module telegram.ext.utils.types), 92</td>
</tr>
<tr>
<td>big_file_id</td>
<td>(telegram.ChatPhoto attribute), 192</td>
</tr>
<tr>
<td>big_file_unique_id</td>
<td>(telegram.ChatPhoto attribute), 192</td>
</tr>
<tr>
<td>bio</td>
<td>(telegram.Chat attribute), 163</td>
</tr>
<tr>
<td>bio</td>
<td>(telegram.ChatJoinRequest attribute), 177</td>
</tr>
<tr>
<td>birth_date</td>
<td>(telegram.PersonalDetails attribute), 337</td>
</tr>
<tr>
<td>BOLD</td>
<td>(telegram.MessageEntity attribute), 246</td>
</tr>
<tr>
<td>Bot</td>
<td>(class in telegram), 96</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.Animation attribute), 94</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.Audio attribute), 95</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.CallbackQuery attribute), 158</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.Document attribute), 203</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.EncryptedPassportElement attribute), 343</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.ext.CallbackDataCache attribute), 89</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.ext.Dispatcher attribute), 10</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.ext.JobQueue attribute), 18</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.ext.Updater attribute), 7</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.Message attribute), 229</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.PassportData attribute), 339</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.PassportFile attribute), 340</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.PhotoSize attribute), 248</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.PreCheckoutQuery attribute), 325</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.ShippingQuery attribute), 323</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.Sticker attribute), 277</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.User attribute), 261</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.Video attribute), 271</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.VideoNote attribute), 272</td>
</tr>
<tr>
<td>bot</td>
<td>(telegram.Voice attribute), 273</td>
</tr>
<tr>
<td>bot()</td>
<td>(telegram.Bot property), 102</td>
</tr>
<tr>
<td>bot()</td>
<td>(telegram.ext.CallbackContext property), 14</td>
</tr>
<tr>
<td>BOT_API_VERSION</td>
<td>(in module telegram.constants), 193</td>
</tr>
</tbody>
</table>
`BotCommandScope` (in module `telegram.constants`), 200
`BotCommandScopeAllChats` (in module `telegram.constants`), 200
`BotCommandScopeAllPrivateChats` (in module `telegram.constants`), 200
`BotCommandScopeChat` (in module `telegram.constants`), 200
`BotCommandScopeChatAdministrators` (in module `telegram.constants`), 200
`BotCommandScopeChatMember` (in module `telegram.constants`), 200
`BotCommandScopeDefault` (in module `telegram.constants`), 200
`bot_data` (in module `telegram.ext.Dispatcher`), 10
`bot_data()` (in module `telegram.ext.CallbackContext`), 15
`bot_data()` (in module `telegram.ext.DictPersistence`), 86
`bot_data_json()` (in module `telegram.ext.DictPersistence`), 86
`bot_ids` (in module `telegram.ext.filters.Filters`), 61
`bot_ids()` (in module `telegram.ext.filters.Filters`), 62
`bot_username` (in module `telegram.LoginUrl`), 220
`BotCommand` (in module `telegram`), 153
`BotCommandScope` (in module `telegram`), 153
`BotCommandScopeAllChatAdministrators` (in module `telegram`), 155
`BotCommandScopeAllGroupChats` (in module `telegram`), 155
`BotCommandScopeAllPrivateChats` (in module `telegram`), 155
`BotCommandScopeChat` (in module `telegram`), 155
`BotCommandScopeChatAdministrators` (in module `telegram`), 156
`BotCommandScopeChatMember` (in module `telegram`), 156
`BotCommandScopeDefault` (in module `telegram`), 154
`BOWLING` (in module `telegram.Dice`), 202
`bowling` (in module `telegram.ext.filters.Filters`), 51
`burst_limit` (in module `telegram.ext.DelayQueue`), 23

`callback` (in module `telegram.ext.CommandHandler`), 38
`callback` (in module `telegram.ext.InlineQueryHandler`), 27
`callback` (in module `telegram.ext.Job`), 43
`callback` (in module `telegram.ext.MessageHandler`), 17
`callback` (in module `telegram.ext.PollAnswerHandler`), 46
`callback` (in module `telegram.ext.PollHandler`), 64
`callback` (in module `telegram.ext.PreCheckoutQueryHandler`), 66
`callback` (in module `telegram.ext.ShippingQueryHandler`), 67
`callback` (in module `telegram.ext.PrefixHandler`), 70
`callback` (in module `telegram.ext.RegexHandler`), 72
`callback` (in module `telegram.ext.ShippingQueryHandler`), 75
`callback` (in module `telegram.ext.StringCommandHandler`), 76
`callback` (in module `telegram.ext.TypeHandler`), 78
`callback_data` (in module `telegram.ext.InvalidCallbackData`), 91
`callback_data` (in module `telegram.InlineKeyboardButton`), 208
`callback_data()` (in module `telegram.ext.DictPersistence`), 87
`callback_data_cache` (in module `telegram.ext.ExtBot`), 5
`callback_data_json()` (in module `telegram.ext.DictPersistence`), 87
`callback_game` (in module `telegram.InlineKeyboardButton`), 208

`CALLBACK_QUERY` (in module `telegram.Update`), 259
`callback_query` (in module `telegram.Update`), 258
`CallbackContext` (in module `telegram.ext`, 14
`CallbackDataCache` (in module `telegram.ext`, 89
`CallbackGame` (in module `telegram.ext`, 327
`CallbackQuery` (in module `telegram.ext`, 157
`CallbackQueryHandler` (in module `telegram.ext`, 29
`can_add_web_page_previews` (in module `telegram.ChatMember`), 183
`can_add_web_page_previews` (in module `telegram.ChatMemberRestricted`), 188
`can_add_web_page_previews` (in module `telegram.ChatPermissions`), 191
`can_be_edited` (in module `telegram.ChatMember`), 181
`can_be_edited` (in module `telegram.ChatMemberAdministrator`), 185
`can_change_info` (in module `telegram.ChatMember`), 182
`can_change_info` (in module `telegram.ChatMemberAdministrator`),
can_change_info (telegram.ChatMemberRestricted attribute), 186
can_change_info (telegram.ChatPermissions attribute), 187
can_change_info (telegram.ChatMember attribute), 182
can_delete_messages (telegram.ChatMemberAdministrator attribute), 185
can_delete_messages (telegram.ChatMemberRestricted attribute), 187
can_delete_messages (telegram.ChatMember attribute), 182
can_edit_messages (telegram.ChatMemberAdministrator attribute), 185
can_edit_messages (telegram.ChatMember attribute), 182
can_join_groups (telegram.User attribute), 261
can_join_groups () (telegram.Bot property), 102
can_manage_chat (telegram.ChatMember attribute), 182
can_manage_chat (telegram.ChatMemberAdministrator attribute), 185
can_manage_chat (telegram.ChatMemberRestricted attribute), 187
can_manage_voice_chats (telegram.ChatMemberAdministrator attribute), 185
can_manage_voice_chats (telegram.ChatMember attribute), 181
can_manage_voice_chats (telegram.ChatMemberAdministrator attribute), 185
can_pin_messages (telegram.ChatMember attribute), 182
can_pin_messages (telegram.ChatMemberAdministrator attribute), 185
can_pin_messages (telegram.ChatMemberRestricted attribute), 187
can_pin_messages (telegram.ChatPermissions attribute), 181
can_post_messages (telegram.ChatMember attribute), 182
can_post_messages (telegram.ChatMemberAdministrator attribute), 185
can_restrict_members (telegram.ChatMember attribute), 182
can_restrict_members (telegram.ChatMemberAdministrator attribute), 185
can_restrict_members (telegram.ChatMemberRestricted attribute), 187
can_read_all_group_messages (telegram.User attribute), 261
can_read_all_group_messages () (telegram.Bot property), 102
can_send_media_messages (telegram.ChatMember attribute), 182
can_send_media_messages (telegram.ChatMemberRestricted attribute), 187
can_send_media_messages (telegram.ChatMember attribute), 182
can_send_messages (telegram.ChatMember attribute), 182
can_send_messages (telegram.ChatMemberRestricted attribute), 187
can_send_messages (telegram.ChatPermissions attribute), 191
can_send_other_messages (telegram.ChatMember attribute), 183
can_send_other_messages (telegram.ChatMemberRestricted attribute), 188
can_send_other_messages (telegram.ChatPermissions attribute), 191
can_send_polls (telegram.ChatMember attribute), 183
can_send_polls (telegram.ChatMemberRestricted attribute), 187
can_send_polls (telegram.ChatPermissions attribute), 191
can_send_polls (telegram.ChatMember attribute), 181
can_send_polls (telegram.ChatPermissions attribute), 191
can_send_polls (telegram.ChatMember attribute), 181
can_set_sticker_set (telegram.Chat attribute), 164
caption (telegram.ext.filters.Filters attribute), 48
caption (telegram.InlineQueryResultAudio attribute), 284
caption (telegram.InlineQueryResultCachedAudio attribute), 285
caption (telegram.InlineQueryResultCachedDocument attribute), 287
caption (telegram.InlineQueryResultCachedGif attribute), 288
caption (telegram.InlineQueryResultCachedMpeg4Gif attribute), 289
caption (telegram.InlineQueryResultCachedPhoto attribute), 291
caption (telegram.InlineQueryResultCachedVideo attribute), 293
caption (telegram.InlineQueryResultCachedVoice attribute), 294
caption (telegram.InlineQueryResultDocument attribute), 297
caption (telegram.InlineQueryResultGif attribute), 384
create join request
(create_new_sticker_set(),
create_deep_linked_url(),
copyMessage(),
correct_option_id(),
country_code(),
country_code(),
country_code(),
country_code(),
create_chat_invite_link(),
country_code()
createNewStickerSet()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
country_code()
Index 389

de_json () (telegram.PollAnswer class method), 251

de_json () (telegram.PreCheckoutQuery class method), 325

de_json () (telegram.ProximityAlertTriggered class method), 252

de_json () (telegram.SecureData class method), 335

de_json () (telegram.SecureValue class method), 336

de_json () (telegram.ShippingQuery class method), 324

de_json () (telegram.Sticker class method), 277

de_json () (telegram.StickerSet class method), 278

de_json () (telegram.SuccessfulPayment class method), 323

de_json () (telegram.TelegramObject class method), 256

de_json () (telegram.Update class method), 260

de_json () (telegram.UserProfilePhotos class method), 268

de_json () (telegram.Venue class method), 269

de_json () (telegram.Video class method), 271

de_json () (telegram.VideoNote class method), 272

de_json () (telegram.VoiceChatParticipantsInvited class method), 274

de_json () (telegram.VoiceChatScheduled class method), 274

de_json_decrypted () (telegram.EncryptedPassportElement class method), 343

de_json_decrypted () (telegram.PassportFile class method), 340

de_list () (telegram.TelegramObject class method), 256

de_list_decrypted () (telegram.PassportFile class method), 340

delete () (telegram.ChatJoinRequest method), 178

delete_chat_join_request () (telegram.Bot method), 105

delete_join_request () (telegram.Chat method), 166

delete_join_request () (telegram.User method), 262

declineChatJoinRequest () (telegram.Bot method), 105

decode_conversations_from_json () (in module telegram.utils.helpers), 346

decode_user_chat_data_from_json () (in module telegram.utils.helpers), 346

decrypted_credentials () (telegram.PassportData property), 339

decrypted_data () (telegram.EncryptedCredentials property), 344

decrypted_data () (telegram.PassportData property), 339

decrypted_secret () (telegram.EncryptedCredentials property), 344

DEFAULT (telegram.BotCommandScope attribute), 154

DEFAULT_20 (in module telegram.utils.helpers), 344

DEFAULT_FALSE (in module telegram.utils.helpers), 344

DEFAULT_NONE (in module telegram.utils.helpers), 344

Defaults (class in telegram.ext), 25

DefaultValue (class in telegram.utils.helpers), 344

DelayQueue (class in telegram.ext), 23
delete () (telegram.Message method), 230
delete_chat_photo (telegram.ext.filters.Filters attribute), 58

delete_chat_photo (telegram.Message attribute), 227
delete_chat_photo () (telegram.Bot method), 106
delete_chat_sticker_set () (telegram.Bot method), 106
delete_message () (telegram.Bot method), 106
delete_message () (telegram.CallbackQuery method), 159

delete_my_commands () (telegram.Bot method), 107
delete_sticker_from_set () (telegram.Bot method), 107
delete_webhook () (telegram.Bot method), 108
deleteChatPhoto () (telegram.Bot method), 106
deleteChatStickerSet () (telegram.Bot method), 106
deleteMessage () (telegram.Bot method), 106
deleteMyCommands () (telegram.Bot method), 106
deleteStickerFromSet () (telegram.Bot method), 106
deleteWebhook () (telegram.Bot method), 106
description (telegram.BotCommand attribute), 153
description (telegram.Chat attribute), 164
description (telegram.Game attribute), 326
description (telegram.InlineQueryResultArticle attribute), 283

description (telegram.InlineQueryResultCachedDocument attribute), 287

description (telegram.InlineQueryResultCachedPhoto attribute), 291

description (telegram.InlineQueryResultCachedVideo attribute), 293

description (telegram.InlineQueryResultDocument attribute), 297

description (telegram.InlineQueryResultPhoto attribute), 305
description (telegram.InlineQueryResultVideo attribute), 309

description (telegram.InlineQuery
edited_channel_post (telegram.Update attribute), 258
edited_message (telegram.ext.filters.Filters attribute), 59
EDITED_MESSAGE (telegram.Update attribute), 259
edited_message (telegram.Update attribute), 258
edited_updates (telegram.ext.MessageHandler attribute), 46
eEditMessageCaption () (telegram.Bot method), 108
eEditMessageLiveLocation () (telegram.Bot method), 108
eEditMessageMedia () (telegram.Bot method), 108
eEditMessageReplyMarkup () (telegram.Bot method), 108
effective_attachment () (telegram.Message property), 232
effective_chat () (telegram.Update property), 269
effective_message () (telegram.Update property), 260
effective_message_type () (in module telegram.utils.helpers), 346
effective_user () (telegram.Update property), 260
element_hash (telegram.PassportElementErrorUnspecified attribute), 333
email (telegram.EncryptedPassportElement attribute), 342
EMAIL (telegram.MessageEntity attribute), 246
email (telegram.OrderInfo attribute), 321
emoji (telegram.Dice attribute), 202
emoji (telegram.Sticker attribute), 277
enabled () (telegram.ext.Job property), 18
encode_conversations_to_json () (in module telegram.utils.helpers), 346
EncryptedCredentials (class in telegram), 343
EncryptedPassportElement (class in telegram), 341
END (telegram.ext.ConversationHandler attribute), 41
entities (telegram.InputTextMessageContent attribute), 312
entities (telegram.Message attribute), 226
entry_points () (telegram.ext.ConversationHandler property), 41
error (telegram.ext.CallbackContext attribute), 14
error_handlers (telegram.ext.Dispatcher attribute), 12
error_handling (telegram.ext.utils.promise.Promise attribute), 91
escape_markdown () (in module telegram.utils.promise.Promise), 346
exc_route (telegram.ext.DelayQueue attribute), 23
exception () (telegram.ext.utils.promise.Promise attribute), 92
exe (telegram.ext.filters.Filters attribute), 52
expire_date (telegram.ChatInviteLink attribute), 176
expiry_date (telegram.IdDocumentData attribute), 337
explanation (telegram.Poll attribute), 249
explanation_entities (telegram.Poll attribute), 249
explanation_parse_mode () (telegram.ext.Defaults property), 26
export_chat_invite_link () (telegram.Bot method), 112
export_invite_link () (telegram.Chat method), 167
exportChatInviteLink () (telegram.Bot method), 112
ExtBot (class in telegram.ext), 5
ExtBot.insert_callback_data () (in module telegram.ext.ExtBot), 5
extract_uuids () (telegram.ext.CallbackDataCache static method), 90
EYES (telegram.MaskPosition attribute), 279

F
FAKE_CHANNEL_ID (in module telegram.constants), 194
fallbacks () (telegram.ext.ConversationHandler property), 41
field_name (telegram.PassportElementErrorDataField attribute), 331
File (class in telegram), 205
file_date (telegram.PassportFile attribute), 340
file_extension (telegram.ext.filters.Filters attribute), 53
file_hash (telegram.PassportElementErrorFile attribute), 328
file_hash (telegram.PassportElementErrorFrontSide attribute), 330
file_hash (telegram.PassportElementErrorReverseSide attribute), 330
file_hash (telegram.PassportElementErrorSelfie attribute), 332
file_hash (telegram.PassportElementErrorTranslationFile attribute), 332
file_hashes (telegram.PassportElementErrorFiles attribute), 329
file_hashes (telegram.PassportElementErrorTranslationFiles attribute), 333
file_id (telegram.Animation attribute), 93
file_id (telegram.Audio attribute), 95
file_id (telegram.Document attribute), 203
file_id (telegram.File attribute), 205
file_id (telegram.PassportFile attribute), 340
file_id (telegram.PhotoSize attribute), 247
file_id (telegram.Sticker attribute), 276
file_id (telegram.Video attribute), 270
file_id (telegram.VideoNote attribute), 271
file_id (telegram.Voice attribute), 273
file_name (telegram.Animation attribute), 94
file_name (telegram.Audio attribute), 95
file_name (telegram.Document attribute), 203
file_name (telegram.Video attribute), 271
file_path (telegram.File attribute), 205
file_size (telegram.Animation attribute), 94
file_size (telegram.Audio attribute), 95
file_size (telegram.Document attribute), 203
file_size (telegram.File attribute), 205
file_size (telegram.PassportFile attribute), 340
file_size (telegram.PhotoSize attribute), 248
file_size (telegram.Sticker attribute), 277
file_size (telegram.Video attribute), 271
file_size (telegram.VideoNote attribute), 272
file_size (telegram.Voice attribute), 273
file_unique_id (telegram.Animation attribute), 93
file_unique_id (telegram.Audio attribute), 95
file_unique_id (telegram.Document attribute), 203
file_unique_id (telegram.File attribute), 205
file_unique_id (telegram.PassportFile attribute), 340
file_unique_id (telegram.PhotoSize attribute), 247
file_unique_id (telegram.Sticker attribute), 276
file_unique_id (telegram.Video attribute), 270
file_unique_id (telegram.VideoNote attribute), 271
file_unique_id (telegram.Voice attribute), 273
FileCredentials (class in telegram), 336
FileInput (in module telegram.utils.types), 350
FileLike (in module telegram.utils.types), 350
filename (telegram.ext.PicklePersistence attribute), 83
filename (telegram.InputFile attribute), 210
files (telegram.EncryptedPassportElement attribute), 342
files (telegram.SecureValue attribute), 336
filter() (telegram.ext.filters.InvertedFilter method), 62
filter() (telegram.ext.filters.MergedFilter method), 62
filter() (telegram.ext.filters.MessageFilter method), 63
filter() (telegram.ext.filters.UpdateFilter method), 63
filter() (telegram.ext.filters.XORFilter method), 63
Filters (class in telegram.ext.filters), 47
filters (telegram.ext.CommandHandler attribute), 38
filters (telegram.ext.MessageHandler attribute), 46
filters (telegram.ext.PrefixHandler attribute), 70
Filters.caption_entity (class in telegram.ext.filters), 48
Filters.caption_regex (class in telegram.ext.filters), 48
Filters.chat (class in telegram.ext.filters), 49
Filters.entity (class in telegram.ext.filters), 53
Filters.forwarded_from (class in telegram.ext.filters), 53
Filters.language (class in telegram.ext.filters), 55
Filters.regex (class in telegram.ext.filters), 55
Filters.sender_chat (class in telegram.ext.filters), 56
Filters.user (class in telegram.ext.filters), 60
Filters.via_bot (class in telegram.ext.filters), 61
FIND_LOCATION (telegram.ChatAction attribute), 175
first_name (telegram.Chat attribute), 163
first_name (telegram.Contact attribute), 201
first_name (telegram.InlineQueryResultContact attribute), 295
first_name (telegram.InputContactMessageContent attribute), 314
first_name (telegram.PersonalDetails attribute), 337
first_name (telegram.User attribute), 261
first_name() (telegram.Bot property), 113
first_name_native (telegram.PersonalDetails attribute), 338
flush() (telegram.ext.BasePersistence method), 80
FOOTBALL (telegram.Dice attribute), 202
football (telegram.ext.filters.Filters attribute), 51
force_reply (telegram.ForceReply attribute), 206
ForceReply (class in telegram), 206
FOREHEAD (telegram.MaskPosition attribute), 279
forward() (telegram.Message method), 233
forward_date (telegram.Message attribute), 225
forward_from (telegram.Message attribute), 225
forward_from_chat (telegram.Message attribute), 225
forward_from_message_id (telegram.Message attribute), 225
forward_from_message_id (telegram.Message attribute), 225
forward_from_message_id (telegram.Message attribute), 225
forward_from_message_id (telegram.Message attribute), 225
forward_message() (telegram.Bot method), 113
forward_sender_name (telegram.Message attribute), 228
forward_signature (telegram.Message attribute), 228
forward_text (telegram.LoginUrl attribute), 220
forwarded (telegram.ext.filters.Filters attribute), 53
forwardMessage() (telegram.Bot method), 113
foursquare_id (telegram.InlineQueryResultVenue attribute), 307
foursquare_id (telegram.InputVenueMessageContent attribute), 314
foursquare_id (telegram.Venue attribute), 269
foursquare_id (telegram.Venue attribute), 269
foursquare_id (telegram.Venue attribute), 269
foursquare_type (telegram.InlineQueryResultVenue attribute), 307
Index
```
input_field_placeholder (telegram.ForceReply attribute), 207
input_field_placeholder (telegram.ReplyKeyboardMarkup attribute), 254
input_file_content (telegram.InputFile attribute), 210
input_message_content (telegram.InlineQueryResultArticle attribute), 282
input_message_content (telegram.InlineQueryResultAudio attribute), 284
input_message_content (telegram.InlineQueryResultCachedAudio attribute), 286
input_message_content (telegram.InlineQueryResultCachedDocument attribute), 287
input_message_content (telegram.InlineQueryResultCachedGif attribute), 288
input_message_content (telegram.InlineQueryResultCachedMpeg4Gif attribute), 290
input_message_content (telegram.InlineQueryResultCachedPhoto attribute), 291
input_message_content (telegram.InlineQueryResultCachedSticker attribute), 292
input_message_content (telegram.InlineQueryResultCachedVideo attribute), 293
input_message_content (telegram.InlineQueryResultCachedVoice attribute), 294
input_message_content (telegram.InlineQueryResultContact attribute), 296
input_message_content (telegram.InlineQueryResultDocument attribute), 297
input_message_content (telegram.InlineQueryResultGif attribute), 300
input_message_content (telegram.InlineQueryResultLocation attribute), 301
input_message_content (telegram.InlineQueryResultMpeg4Gif attribute), 304
input_message_content (telegram.InlineQueryResultPhoto attribute), 305
input_message_content (telegram.InlineQueryResultVenue attribute), 307
input_message_content (telegram.InlineQueryResultVideo attribute), 309
input_message_content (telegram.InlineQueryResultVoice attribute), 311
InputContactMessageContent (class in telegram), 314
InputFile (class in telegram), 210
InputInvoiceMessageContent (class in telegram), 315
InputLocationMessageContent (class in telegram), 312
InputMedia (class in telegram), 211
InputMediaAnimation (class in telegram), 211
InputMediaAudio (class in telegram), 212
InputMediaDocument (class in telegram), 214
InputMediaPhoto (class in telegram), 215
InputMediaVideo (class in telegram), 216
InputMessageContent (class in telegram), 311
InputTextMessageContent (class in telegram), 311
InputVenueMessageContent (class in telegram), 313
```

**insert_bot()** *(telegram.ext.BasePersistence method), 81*

**internal_passport** *(telegram.SecureData attribute), 335*

**InvalidCallbackData** *(class in telegram.ext), 91*

**InvalidToken** *(class in telegram), 204*

**InvertedFilter** *(class in telegram.ext.filters), 62*

**invite_link** *(telegram.Chat attribute), 164*

**invite_link** *(telegram.ChatInviteLink attribute), 176*

**invite_link** *(telegram.ChatJoinRequest attribute), 178*

**invite_link** *(telegram.ChatMemberUpdated attribute), 189*

**Invoice** *(class in telegram), 319*

**invoice** *(telegram.ext.filters.Filters attribute), 55*

**invoice** *(telegram.Message attribute), 227*

**invoice_payload** *(telegram.PreCheckoutQuery attribute), 325*

**invoice_payload** *(telegram.ShippingQuery attribute), 323*

**invoice_payload** *(telegram.SuccessfulPayment attribute), 322*

**ip_address** *(telegram.WebhookInfo attribute), 275*

**is_animated** *(telegram.Sticker attribute), 277*

**is_animated** *(telegram.StickerSet attribute), 278*

**is_anonymous** *(telegram.ChatMember attribute), 181*

**is_anonymous** *(telegram.ChatMemberAdministrator attribute), 185*

**is_anonymous** *(telegram.ChatMemberOwner attribute), 184*
is_anonymous (telegram.Poll attribute), 248
is_automatic_forward (telegram.ext.filters.Filters attribute), 55
is_automatic_forward (telegram.Message attribute), 225
is_bot (telegram.User attribute), 261
is_closed (telegram.Poll attribute), 248
is_flexible (telegram.InputInvoiceMessageContent attribute), 317
is_image() (telegram.InputFile static method), 210
is_local_file() (in module telegram.utils.helpers), 347
is_member (telegram.ChatMember attribute), 182
is_member (telegram.ChatMemberRestricted attribute), 187
is_primary (telegram.ChatInviteLink attribute), 176
is_revoked (telegram.ChatInviteLink attribute), 176
is_video (telegram.Sticker attribute), 277
is_video (telegram.StickerSet attribute), 278
ITALIC (telegram.MessageEntity attribute), 246

J
Job (class in telegram.ext), 17
job (telegram.ext.CallbackContext attribute), 14
job (telegram.ext.Job attribute), 17
job_queue (telegram.ext.Dispatcher attribute), 10
job_queue (telegram.ext.Job attribute), 17
job_queue (telegram.ext.Updater attribute), 7
job_queue() (telegram.ext.CallbackContext property), 16
JobQueue (class in telegram.ext), 18
jobs() (telegram.ext.JobQueue method), 18
jpg (telegram.ext.filters.Filterstribute), 52
JSONDict (in module telegram.utils.types), 350

K
keyboard (telegram.ReplyKeyboardMarkup attribute), 253
KeyboardButton (class in telegram), 218
KeyboardButtonPollType (class in telegram), 219
kick_chat_member() (telegram.Bot method), 119
kick_member() (telegram.Chat method), 168
kickChatMember() (telegram.Bot method), 119
KICKED (telegram.ChatMember attribute), 183
kwargs (telegram.ext.utils.promise.Promise attribute), 91

L
label (telegram.LabeledPrice attribute), 319
LabeledPrice (class in telegram), 319
language (telegram.MessageEntity attribute), 245
language_code (telegram.User attribute), 261
last_error_date (telegram.WebhookInfo attribute), 275
last_error_message (telegram.WebhookInfo attribute), 275
last_name (telegram.Chat attribute), 163
last_name (telegram.Contact attribute), 201
last_name (telegram.InlineQueryResultContact attribute), 295
last_name (telegram.InputContactMessageContent attribute), 314
last_name (telegram.PersonalDetails attribute), 337
last_name (telegram.User attribute), 261
last_name() (telegram.Bot property), 119
last_name_native (telegram.PersonalDetails attribute), 338
latitude (telegram.InlineQueryResultLocation attribute), 301
latitude (telegram.InlineQueryResultVenue attribute), 306
latitude (telegram.InputLocationMessageContent attribute), 312
latitude (telegram.InputVenueMessageContent attribute), 313
latitude (telegram.Location attribute), 219
leave() (telegram.Chat method), 168
leave_chat() (telegram.Bot method), 119
leaveChat() (telegram.Bot method), 119
LEFT (telegram.ChatMember attribute), 183
left_chat_member (telegram.ext.filters.Filterstribute), 58
left_chat_member (telegram.Message attribute), 227
length (telegram.MessageEntity attribute), 245
length (telegram.VideoNote attribute), 272
link() (telegram.Bot property), 119
link() (telegram.Chat property), 168
link() (telegram.Message property), 233
link() (telegram.User property), 262
linked_chat_id (telegram.Chat attribute), 164
live_period (telegram.InlineQueryResultLocation attribute), 301
live_period (telegram.InputLocationMessageContent attribute), 313
live_period (telegram.Location attribute), 219
Location (class in telegram), 219
location (telegram.Chat attribute), 164
location (telegram.ChatLocation attribute), 178
location (telegram.ChosenInlineResult attribute), 318
location (telegram.ext.filters.Filters attribute), 55
location (telegram.InlineQuery attribute), 281
location (telegram.Message attribute), 227
location (telegram.Venue attribute), 269
log_out() (telegram.Bot method), 120
login_url (telegram.InlineKeyboardButton attribute), 208
LoginUrl (class in telegram), 220
logout() (telegram.Bot method), 120
longitude (telegram.InlineQueryResultLocation attribute), 301
longitude (telegram.InlineQueryResultVenue attribute), 306
map_to_parent() (telegram.ext.ConversationHandler property), 42
MARKDOWN (telegram.ParseMode attribute), 247
MARKDOWN_V2 (telegram.ParseMode attribute), 247
mask_position (telegram.Sticker attribute), 277
MatchPosition (class in telegram), 279
match() (telegram.ext.CallbackContext property), 16
matches (telegram.ext.CallbackContext attribute), 14
MAX_ANSWER_CALLBACK_QUERY_TEXT_LENGTH (in module telegram.constants), 193
MAX_ANSWER_TEXT_LENGTH (telegram.CallbackQuery attribute), 158
MAX_CAPTION_LENGTH (in module telegram.constants), 193
max_connections (telegram.WebhookInfo attribute), 275
MAX_FILESIZE_DOWNLOAD (in module telegram.constants), 193
MAX_FILESIZE_UPLOAD (in module telegram.constants), 193
MAX_INLINE_QUERY_RESULTS (in module telegram.constants), 193
MAX_LENGTH (telegram.PollOption attribute), 251
MAX_MESSAGE_ENTITIES (in module telegram.constants), 194
MAX_MESSAGE_LENGTH (in module telegram.constants), 193
MAX_MESSAGES_PER_MINUTE_PER_GROUP (in module telegram.constants), 193
MAX_MESSAGES_PER_SECOND (in module telegram.constants), 193
MAX_MESSAGES_PER_SECOND_PER_CHAT (in module telegram.constants), 193
MAX_OPTION_LENGTH (telegram.Poll attribute), 249
MAX_PHOTOSIZE_UPLOAD (in module telegram.constants), 193
MAX_POLL_OPTION_LENGTH (in module telegram.constants), 198
MAX_POLL_QUESTION_LENGTH (in module telegram.constants), 198
MAX_QUESTION_LENGTH (telegram.Poll attribute), 249
MAX_RESULTS (telegram.InlineQuery attribute), 281
max_tip_amount (telegram.InlineInvoiceMessageContent attribute), 316
maxsize (telegram.ext.CallbackDataCache attribute), 89
media (telegram.InputMediaAnimation attribute), 212
media (telegram.InputMediaAudio attribute), 213
media (telegram.InputMediaDocument attribute), 214
media (telegram.InputMediaPhoto attribute), 215
media (telegram.InputMediaVideo attribute), 217
media_group_id (telegram.Message attribute), 225
member_limit (telegram.ChatMember attribute), 183
MESSAGE (telegram.MessageEntity attribute), 246
Mention (telegram.MessageEntity attribute), 246
mention_button() (telegram.User method), 262
mention_html() (in module telegram.utils.helpers), 347
mention_html() (telegram.User method), 263
mention_markdown() (in module telegram.utils.helpers), 347
mention_markdown() (telegram.User method), 263
mention_markdown_v2() (telegram.User method), 263
MergedFilter (class in telegram.ext.filters), 62
Message (class in telegram), 221
message (telegram.CallbackQuery attribute), 158
message (telegram.error.TelegramError attribute), 204
message (telegram.ext.filters.Filters attribute), 59
message (telegram.PassportElementError attribute), 328
message (telegram.PassportElementErrorDataField attribute), 331
message (telegram.PassportElementErrorFile attribute), 329
message (telegram.PassportElementErrorFiles attribute), 329
message (telegram.PassportElementErrorFrontSide attribute), 330
message (telegram.PassportElementErrorReverseSide attribute), 330
message (telegram.PassportElementErrorSelfie attribute), 332
message (telegram.PassportElementErrorTranslationFile attribute), 332
message (telegram.PassportElementErrorTranslationFiles attribute), 333
message (telegram.PassportElementErrorUnspecified attribute), 334
MESSAGE (telegram.Update attribute), 259
message (telegram.Update attribute), 258
message_auto_delete_time (telegram.Chat attribute), 164
message_auto_delete_time (telegram.Message attribute), 244
message_auto_delete_timer_changed (telegram.ext.filters.Filters attribute), 58
message_auto_delete_timer_changed (telegram.Message attribute), 227
message_id (telegram.Message attribute), 224
message_id (telegram.MessageId attribute), 245
message_text (telegram.InputTextMessageContent attribute), 311
message_updates (telegram.ext.MessageHandler attribute), 46
MessageAutoDeleteTimerChanged (class in telegram), 244
MessageEntity (class in telegram), 245
MESSAGEENTITY_ALL_TYPES (in module telegram.constants), 198
MESSAGEENTITY_BOLD (in module telegram.constants), 197
MESSAGEENTITY_BOT_COMMAND (in module telegram.constants), 197
MESSAGEENTITY_CASHTAG (in module telegram.constants), 197
MESSAGEENTITY_CODE (in module telegram.constants), 197
MESSAGEENTITY_EMAIL (in module telegram.constants), 197
MESSAGEENTITY_HASHTAG (in module telegram.constants), 196
MESSAGEENTITY_ITALIC (in module telegram.constants), 197
MESSAGEENTITY_MENTION (in module telegram.constants), 196
MESSAGEENTITY_PHONE_NUMBER (in module telegram.constants), 197
MESSAGEENTITY_PRE (in module telegram.constants), 197
MESSAGEENTITY_SPOILER (in module telegram.constants), 197
MESSAGEENTITY_STRIKETHROUGH (in module telegram.constants), 197
MESSAGEENTITY_TEXT_LINK (in module telegram.constants), 197
MESSAGEENTITY_TEXT_MENTION (in module telegram.constants), 197
MESSAGEENTITY_UNDERLINE (in module telegram.constants), 197
MESSAGEENTITY_URL (in module telegram.constants), 197
MessageFilter (class in telegram.ext.filters), 62
MessageHandler (class in telegram.ext), 44
MessageId (class in telegram), 245
MessageQueue (class in telegram.ext), 22
messages (telegram.ext.filters.Filters attribute), 59
middle_name (telegram.PersonalDetails attribute), 337
middle_name_native (telegram.PersonalDetails attribute), 338
migrate (telegram.ext.filters.Filters attribute), 58
migrate_from_chat_id (telegram.Message attribute), 227
migrate_to_chat_id (telegram.Message attribute), 227
mime_type (telegram.Animation attribute), 94
mime_type (telegram.Audio attribute), 95
mime_type (telegram.Document attribute), 203
mime_type (telegram.ext.filters.Filters attribute), 52
mime_type (telegram.InlineQueryResultDocument attribute), 297
mime_type (telegram.InlineQueryResultVideo attribute), 309
mime_type (telegram.Video attribute), 271
mime_type (telegram.Voice attribute), 273
module
telegram.constants, 193
telegram.error, 204
telegram.ext.filters, 47
telegram.ext.utils.types, 92
telegram.utils.helpers, 344
telegram.utils.types, 350
MOUTH (telegram.MaskPosition attribute), 279
mp3 (telegram.ext.filters.Filters attribute), 52
mpeg4_duration (telegram.InlineQueryResultMpeg4Gif attribute), 303
mpeg4_file_id (telegram.InlineQueryResultCachedMpeg4Gif attribute), 289
mpeg4_height (telegram.InlineQueryResultMpeg4Gif attribute), 303
mpeg4_url (telegram.InlineQueryResultMpeg4Gif attribute), 303
mpeg4_width (telegram.InlineQueryResultMpeg4Gif attribute), 303
MY_CHAT_MEMBER (telegram.ext.ChatMemberHandler attribute), 34
MY_CHAT_MEMBER (telegram.Update attribute), 259
my_chat_member (telegram.Update attribute), 258
N
name (telegram.ChatInviteLink attribute), 176
name (telegram.ext.DelayQueue attribute), 23
name (telegram.ext.filters.BaseFilter attribute), 47
name (telegram.ext.filters.MessageFilter attribute), 63
name (telegram.ext.filters.UpdateFilter attribute), 63
name (telegram.Job attribute), 17
name (telegram.OrderInfo attribute), 321
name (telegram.StickerSet attribute), 278
name () (telegram.Bot property), 120
name () (telegram.ext.ConversationHandler property), 42
name () (telegram.User property), 263
need_email (telegram.InputInvoiceMessageContent attribute), 317
need_name (telegram.InputInvoiceMessageContent attribute), 317
need_phone_number (telegram.InputInvoiceMessageContent attribute), 317
need_shipping_address (telegram.InputInvoiceMessageContent attribute), 317

Index

399
pass_chat_data (telegram.ext.ChatJoinRequestHandler attribute), 32
pass_chat_data (telegram.ext.ChatMemberHandler attribute), 34
pass_chat_data (telegram.ext.ChosenInlineResultHandler attribute), 36
pass_chat_data (telegram.ext.CommandHandler attribute), 38
pass_job_queue (telegram.ext.InlineQueryHandler attribute), 44
pass_job_queue (telegram.ext.MessageHandler attribute), 46
pass_job_queue (telegram.ext.PollAnswerHandler attribute), 65
pass_job_queue (telegram.ext.PollHandler attribute), 66
pass_job_queue (telegram.ext.PollAnswerHandler attribute), 68
pass_job_queue (telegram.ext.PollHandler attribute), 69
pass_job_queue (telegram.ext.PrefixHandler attribute), 70
pass_job_queue (telegram.ext.RegexHandler attribute), 72
pass_job_queue (telegram.ext.ShippingQueryHandler attribute), 74
pass_job_queue (telegram.ext.StringCommandHandler attribute), 75
pass_job_queue (telegram.ext.StringRegexHandler attribute), 77
pass_job_queue (telegram.ext.TypeHandler attribute), 78
pass_update_queue (telegram.ext.CallbackQueryHandler attribute), 30
pass_update_queue (telegram.ext.InlineQueryHandler attribute), 44
pass_update_queue (telegram.ext.MessageHandler attribute), 46
pass_update_queue (telegram.ext.CallbackQueryHandler attribute), 30
pass_update_queue (telegram.ext.InlineQueryHandler attribute), 44
pass_update_queue (telegram.ext.ChosenInlineResultHandler attribute), 36
pass_update_queue (telegram.ext.CommandHandler attribute), 38
pass_update_queue (telegram.ext.InlineQueryHandler attribute), 43
pass_update_queue (telegram.ext.MessageHandler attribute), 46
passport (telegram.ext.CallbackQueryHandler attribute), 30
pass_user_data (telegram.ext.CallbackQueryHandler attribute), 78
pass_user_data (telegram.ext.ChatJoinRequestHandler attribute), 32
pass_user_data (telegram.ext.ChatMemberHandler attribute), 34
pass_user_data (telegram.ext.CommandHandler attribute), 38
pass_user_data (telegram.ext.Handler attribute), 27
pass_user_data (telegram.ext.InlineQueryHandler attribute), 44
pass_user_data (telegram.ext.MessageHandler attribute), 46
pass_user_data (telegram.ext.PollAnswerHandler attribute), 65
pass_user_data (telegram.ext.PollHandler attribute), 66
pass_user_data (telegram.ext.PreCheckoutQueryHandler attribute), 68
pass_user_data (telegram.ext.PrefixHandler attribute), 70
pass_user_data (telegram.ext.RegexHandler attribute), 72
pass_user_data (telegram.ext.ShippingQueryHandler attribute), 74
pass_user_data (telegram.SecureData attribute), 335
passport_data (telegram.ext.filters.Filters attribute), 55
passport_data (telegram.Message attribute), 228
passport_data (telegram.Message attribute), 228
passport_registration (telegram.SecureData attribute), 335
PassportData (class in telegram), 339
PassportElementError (class in telegram), 328
PassportElementErrorDataField (class in telegram), 331
PassportElementErrorFile (class in telegram), 328
PassportElementErrorFiles (class in telegram), 329
PassportElementErrorFrontSide (class in telegram), 330
PassportElementErrorReverseSide (class in telegram), 329
PassportElementErrorSelfie (class in telegram), 331
PassportElementErrorTranslationFile (class in telegram), 332
PassportElementErrorTranslationFiles (class in telegram), 333
PassportFile (class in telegram), 340
pattern (telegram.ext.CallbackQueryHandler attribute), 30
pattern (telegram.ext.ChosenInlineResultHandler attribute), 36
pattern (telegram.ext.InlineQueryHandler attribute), 44
pattern (telegram.ext.StringRegexHandler attribute), 72
pattern (telegram.ext.stringRegexHandler attribute), 76
pay (telegram.InlineKeyboardButton attribute), 209
payload (telegram.InputInvoiceMessageContent attribute), 316
pdf (telegram.ext.filters.Filters attribute), 52
pending_join_request_count (telegram.ChatInviteLink attribute), 177
pending_update_count (telegram.WebhookInfo attribute), 275
per_chat() (telegram.ext.ConversationHandler property), 42
per_message() (telegram.ext.ConversationHandler property), 42
per_user() (telegram.ext.ConversationHandler property), 42
performer (telegram.Audio attribute), 95
performer (telegram.InlineQueryResultAudio attribute), 284
performer (telegram.InputMediaAudio attribute), 213
permissions (telegram.Chat attribute), 164
persistence (telegram.ext.Dispatcher attribute), 11
persistence (telegram.ext.Updator attribute), 8
persistence() (telegram.ext.ConversationHandler
python-telegram-bot Documentation, Release 13.11

send_photo() (telegram.Chat method), 172
send_poll() (telegram.User method), 266
send_poll() (telegram.Bot method), 136
send_poll() (telegram.Chat method), 172
send_poll() (telegram.User method), 266
send_sticker() (telegram.Bot method), 138
send_sticker() (telegram.Chat method), 172
send_venue() (telegram.Bot method), 138
send_venue() (telegram.Chat method), 172
send_video() (telegram.Bot method), 267
send_video() (telegram.User method), 139
send_video() (telegram.Chat method), 172
send_video() (telegram.User method), 267
send_video_note() (telegram.Bot method), 141
send_video_note() (telegram.User method), 173
send_voice() (telegram.Bot method), 142
send_voice() (telegram.Chat method), 173
send_voice() (telegram.User method), 267
sendAnimation() (telegram.Bot method), 123
sendAudio() (telegram.Bot method), 123
sendChatAction() (telegram.Bot method), 123
sendContact() (telegram.Bot method), 123
sendDice() (telegram.Bot method), 123
sendDocument() (telegram.Bot method), 123
SEND() (telegram.Chat attribute), 165
sender_chat (telegram.Message attribute), 225
sendGame() (telegram.Bot method), 123
sendInvoice() (telegram.Bot method), 123
sendLocation() (telegram.Bot method), 124
sendMediaGroup() (telegram.Bot method), 124
sendMessage() (telegram.Bot method), 124
sendPhoto() (telegram.Bot method), 124
sendPoll() (telegram.Bot method), 124
sendSticker() (telegram.Bot method), 124
sendVenue() (telegram.Bot method), 124
sendVideo() (telegram.Bot method), 124
sendVideoNote() (telegram.Bot method), 124
sendVoice() (telegram.Bot method), 125
SERVICE_CHAT_ID (in module telegram.constants), 194
set_administrator_custom_title() (telegram.Chat method), 173
set_bot() (telegram.Bot method), 82
set_chatAdministratorCustomTitle() (telegram.Bot method), 143
set_chat_description() (telegram.Bot method), 144
set_chat_permissions() (telegram.Bot method), 144
set_chat_photo() (telegram.Bot method), 145
set_chat_sticker_set() (telegram.Bot method), 145
set_chat_title() (telegram.Bot method), 145
set_credential() (telegram.Bot method), 146
set_dispatcher() (telegram.ext.JobQueue method), 21
set_game_score() (telegram.Bot method), 146
set_game_score() (telegram.CallbackQuery method), 161
set_game_score() (telegram.Message method), 242
set_my_commands() (telegram.Bot method), 147
set_name(telegram.Sticker attribute), 277
setPassportDataErrors() (telegram.Bot method), 147
set_permissions() (telegram.Chat method), 173
set_sticker_position_in_set() (telegram.Bot method), 147
set_sticker_set_thumb() (telegram.Bot method), 148
set_webhook() (telegram.Bot method), 148
setChatAdministratorCustomTitle() (telegram.Bot method), 143
setChatDescription() (telegram.Bot method), 143
setChatPermissions() (telegram.Bot method), 143
setChatPhoto() (telegram.Bot method), 143
setChatStickerSet() (telegram.Bot method), 143
setChatTitle() (telegram.Bot method), 143
setGameScore() (telegram.Bot method), 143
setMyCommands() (telegram.Bot method), 144
setPassportDataErrors() (telegram.Bot method), 144
setStickerPositionInSet() (telegram.Bot method), 144
setStickerThumb() (telegram.Bot method), 144
setWebhook() (telegram.Bot method), 144
shipping_address (telegram.OrderInfo attribute), 321
shipping_address (telegram.ShippingQuery attribute), 323
shipping_option_id (telegram.PreCheckoutQuery attribute), 325
shipping_option_id (telegram.SuccessfulPayment attribute), 322
SHIPPING_QUERY (telegram.Update attribute), 260
shipping_query (telegram.Update attribute), 258
ShippingAddress (class in telegram), 320
ShoppingOption (class in telegram), 321
ShippingQuery (class in telegram), 323
ShippingQueryHandler (class in telegram.ext), 73
single_file (telegram.ext.PicklePersistence attribute), 83
SLOT_MACHINE (telegram.Dice attribute), 202
slot_machine (telegram.ext.filters.Filters attribute), 51
slow_mode_delay (telegram.Chat attribute), 164
SLT (in module telegram.utils.types), 350
small_file_id (telegram.ChatPhoto attribute), 192  
small_file_unique_id (telegram.ChatPhoto attribute), 192  
source (telegram.PassportElementError attribute), 328  
SPOILER (telegram.MessageEntity attribute), 246  
start() (telegram.ext.Dispatcher method), 13  
start() (telegram.ext.JobQueue method), 21  
start() (telegram.ext.MessageQueue method), 23  
start_date (telegram.VoiceChatScheduled attribute), 274  
start_parameter (telegram.Invoice attribute), 320  
start_polling() (telegram.ext.Updater method), 8  
start_webhook() (telegram.ext.Updater method), 8  
state (telegram.ext.DispatcherHandlerStop attribute), 13  
state (telegram.ResidentialAddress attribute), 338  
state (telegram.ShippingAddress attribute), 320  
states() (telegram.ext.ConversationHandler property), 42  
status (telegram.ChatMember attribute), 181  
status (telegram.ChatMemberAdministrator attribute), 185  
status (telegram.ChatMemberBanned attribute), 188  
status (telegram.ChatMemberLeft attribute), 188  
status (telegram.ChatMemberMember attribute), 186  
status (telegram.ChatMemberOwner attribute), 183  
status (telegram.ChatMemberRestricted attribute), 187  
status_update (telegram.ext.filters.Filters attribute), 58  
Sticker (class in telegram), 276  
sticker (telegram.ext.filters.Filters attribute), 59  
sticker (telegram.Message attribute), 226  
STICKER_EYES (in module telegram.constants), 198  
STICKER_EYES (in module telegram.constants), 198  
sticker_file_id (telegram.InlineQueryResultCachedSticker attribute), 292  
STICKER_FOREHEAD (in module telegram.constants), 198  
STICKER_MOUTH (in module telegram.constants), 198  
sticker_set_name (telegram.Chat attribute), 164  
stickers (telegram.StickerSet attribute), 278  
StickerSet (class in telegram), 278  
stop() (telegram.ext.DelayQueue method), 24  
stop() (telegram.ext.Dispatcher method), 13  
stop() (telegram.ext.JobQueue method), 21  
stop() (telegram.ext.MessageQueue method), 23  
stop() (telegram.ext.Updater method), 9  
stop() (telegram.utils.request.Request method), 350  
stop_message_live_location() (telegram.Bot method), 150  
stop_message_live_location() (telegram.CallbackQuery method), 161  
stop_poll() (telegram.Bot method), 150  
stop_poll() (telegram.Message method), 242  
stopMessageLiveLocation() (telegram.Bot method), 149  
stopPoll() (telegram.Bot method), 150  
store_bot_data (telegram.ext.BasePersistence attribute), 80  
store_bot_data (telegram.ext.DictPersistence attribute), 86  
store_bot_data (telegram.ext.PicklePersistence attribute), 83  
store_callback_data (telegram.ext.BasePersistence attribute), 80  
store_callback_data (telegram.ext.DictPersistence attribute), 86  
store_callback_data (telegram.ext.PicklePersistence attribute), 83  
store_chat_data (telegram.ext.BasePersistence attribute), 80  
store_chat_data (telegram.ext.DictPersistence attribute), 86  
store_chat_data (telegram.ext.PicklePersistence attribute), 83  
store_user_data (telegram.ext.BasePersistence attribute), 80  
store_user_data (telegram.ext.DictPersistence attribute), 86  
store_user_data (telegram.ext.PicklePersistence attribute), 83  
street_line1 (telegram.ResidentialAddress attribute), 338  
street_line1 (telegram.ShippingAddress attribute), 320  
street_line2 (telegram.ResidentialAddress attribute), 338  
street_line2 (telegram.ShippingAddress attribute), 320  
strict (telegram.ext.TypeHandler attribute), 78  
STRIKETHROUGH (telegram.MessageEntity attribute), 246  
StringCommandHandler (class in telegram.ext), 74  
StringRegexHandler (class in telegram.ext), 76  
successful_payment (telegram.ext.filters.Filters attribute), 59  
successful_payment (telegram.Message attribute), 228  
SuccessfulPayment (class in telegram), 322  
suggested_tip_amounts (telegram.InputInvoiceMessageContent attribute), 316  
super_group (telegram.ext.filters.Filters.sender_chat attribute), 57, 58
Index
type (telegram.BotCommandScopeChatAdministrators attribute), 156

type (telegram.BotCommandScopeChatMember attribute), 156

type (telegram.BotCommandScopeDefault attribute), 154

type (telegram.Chat attribute), 163

type (telegram.EncryptedPassportElement attribute), 342

type (telegram.ext.TypeHandler attribute), 78

type (telegram.InlineQueryResult attribute), 281

type (telegram.InlineQueryResultArticle attribute), 282

type (telegram.InlineQueryResultAudio attribute), 284

type (telegram.InlineQueryResultCachedAudio attribute), 285

type (telegram.InlineQueryResultCachedDocument attribute), 286

type (telegram.InlineQueryResultCachedGif attribute), 288

type (telegram.InlineQueryResultCachedMpeg4Gif attribute), 289

type (telegram.InlineQueryResultCachedPhoto attribute), 290

type (telegram.InlineQueryResultCachedSticker attribute), 292

type (telegram.InlineQueryResultCachedVideo attribute), 293

type (telegram.InlineQueryResultCachedVoice attribute), 294

type (telegram.InlineQueryResultContact attribute), 295

type (telegram.InlineQueryResultDocument attribute), 297

type (telegram.InlineQueryResultGame attribute), 298

type (telegram.InlineQueryResultGif attribute), 299

type (telegram.InlineQueryResultLocation attribute), 301

type (telegram.InlineQueryResultMpeg4Gif attribute), 303

type (telegram.InlineQueryResultPhoto attribute), 304

type (telegram.InlineQueryResultVenue attribute), 306

type (telegram.InlineQueryResultVideo attribute), 308

type (telegram.InlineQueryResultVoice attribute), 310

type (telegram.InputMediaAnimation attribute), 212

type (telegram.InputMediaAudio attribute), 213

type (telegram.InputMediaDocument attribute), 214

type (telegram.InputMediaPhoto attribute), 215

type (telegram.InputMediaVideo attribute), 217

type (telegram.KeyboardButtonPollType attribute), 219

type (telegram.MessageEntity attribute), 245

type (telegram.PASSPORTDOCUMENTERROR attribute), 328

type (telegram.PASSPORTDOCUMENTERRORATTRIBUTEDATAFIELD attribute), 331

type (telegram.PASSPORTDOCUMENTERRORFILE attribute), 328

type (telegram.PASSPORTDOCUMENTERRORFILES attribute), 329

type (telegram.PASSPORTDOCUMENTERRORFRONTSIDE attribute), 330

type (telegram.PASSPORTDOCUMENTERRORREVERSESIDE attribute), 330

type (telegram.PASSPORTDOCUMENTERRORSELFIE attribute), 332

type (telegram.PASSPORTDOCUMENTERRORTRANSLATIONFILE attribute), 332

type (telegram.PASSPORTDOCUMENTERRORTRANSLATIONFILES attribute), 333

type (telegram.PASSPORTDOCUMENTERRORUNSPECIFIED attribute), 333

type (telegram.Poll attribute), 248

TypeHandler (class in telegram.ext), 77

TYPING (telegram.ChatAction attribute), 175

tzinfo () (telegram.ext.Defaults property), 26

U

UD (in module telegram.ext.utils.types), 93

Unauthorized, 204

unban_chat () (telegram.Chat method), 173

unban_chat_member () (telegram.Bot method), 151

unban_chat_sender_chat () (telegram.Bot method), 151

unban_member () (telegram.Chat method), 174

unban_sender_chat () (telegram.Chat method), 174

unbanChatMember () (telegram.Bot method), 151

unbanChatSenderChat () (telegram.Bot method), 151

UNDERLINE (telegram.MessageEntity attribute), 246

unpin() (telegram.Message method), 244

unpin_all_chat_messages () (telegram.Bot method), 151

unpin_all_messages () (telegram.Chat method), 174

unpin_all_messages () (telegram.User method), 268

unpin ChatMessage () (telegram.Bot method), 152

unpin_message () (telegram.CallbackQuery method), 161

unpin_message () (telegram.Chat method), 174

unpin_message () (telegram.User method), 268

unpinAllChatMessages () (telegram.Bot method), 151

unpinChatMessage () (telegram.Bot method), 151

until_date (telegram.ChatMember attribute), 181

until_date (telegram.ChatMemberBanned attribute), 188
UPDATE_MY_CHAT_MEMBER(update_persistence()), 188
UPDATE_MESSAGE(update_conversation()), 199
UPDATE_EDITED_MESSAGE(update_conversation()), 199
UPDATE_CHANNEL_POST(update_conversation()), 199
UPDATE_CHAT_MEMBER(update_conversation()), 199
UPDATE_CHAT_JOIN_REQUEST(update_conversation()), 199
UPDATE_CALLBACK_QUERY(update_callback_data()), 88
UPDATE_INLINE_QUERY(update_callback_data()), 88
UPDATE_SHIPPING_QUERY(update_callback_data()), 88
UPDATE_PRE_CHECKOUT_QUERY(update_callback_data()), 88
UPDATE_POLL(update_callback_data()), 88
UPDATE_POLL_ANSWER(update_callback_data()), 88
UPDATE_ALL_TYPES(update_callback_data()), 88
UPLOADED_AUDIO(update_callback_data()), 63
UPDATE_VIDEO_NOTE(update_callback_data()), 63
UPDATE_VIDEO(update_callback_data()), 63
UPDATE_VOICE(update_callback_data()), 63
UPDATE_DOCUMENT(update_callback_data()), 63
update_queue(update_callback_data()), 7
update_queue(update_callback_data()), 16
update_queue(update_callback_data()), 199
update_user_data(update_callback_data()), 85
update_user_data(update_callback_data()), 85
update_user_data(update_callback_data()), 85
update_user_data(update_callback_data()), 85
update_user_data(update_callback_data()), 85
update_user_data(update_callback_data()), 85
update_user_data(update_callback_data()), 85
update_queue(update_callback_data()), 10
update_queue(update_callback_data()), 10
update_queue(update_callback_data()), 10
update_queue(update_callback_data()), 10
user_id (telegram.BotCommandScopeChatMember attribute), 156
user_id (telegram.Contact attribute), 201
user_ids (telegram.ext.filters.Filters.user attribute), 60
user_ids () (telegram.ext.filters.Filters.user property), 61
user_sig_handler (telegram.ext.Updater attribute), 7
username (telegram.Chat attribute), 163
username (telegram.User attribute), 261
username () (telegram.Bot property), 153
usernames (telegram.ext.filters.Filters.chat attribute), 49
usernames (telegram.ext.filters.Filters.forwarded_from attribute), 54
usernames (telegram.ext.filters.Filters.sender_chat attribute), 57
usernames (telegram.ext.filters.Filters.user attribute), 60
usernames (telegram.ext.filters.Filters.via_bot attribute), 61
UserProfilePhotos (class in telegram), 268
users (telegram.VoiceChatParticipantsInvited attribute), 274
utility_bill (telegram.SecureData attribute), 335

V
value (telegram.Dice attribute), 202
value (telegram.utils.helpers.DefaultValue attribute), 345
vcard (telegram.Contact attribute), 201
vcard (telegram.InlineQueryResultContact attribute), 295
vcard (telegram.InputContactMessageContent attribute), 314
Venue (class in telegram), 269
venue (telegram.ext.filters.Filters attribute), 61
venue (telegram.Message attribute), 227
via_bot (telegram.Message attribute), 228
Video (class in telegram), 270
video (telegram.ext.filters.Filters attribute), 52, 62
video (telegram.Message attribute), 226
video_duration (telegram.InlineQueryResultVideo attribute), 309
video_file_id (telegram.InlineQueryResultCachedVideo attribute), 293
video_height (telegram.InlineQueryResultVideo attribute), 309
video_note (telegram.ext.filters.Filters attribute), 62
video_note (telegram.Message attribute), 226
video_url (telegram.InlineQueryResultVideo attribute), 308
video_width (telegram.InlineQueryResultVideo attribute), 309

VideoNote (class in telegram), 271
Voice (class in telegram), 272
voice (telegram.ext.filters.Filters attribute), 62
voice (telegram.Message attribute), 226
voice_chat-ended (telegram.ext.filters.Filters attribute), 59
voice_chat-ended (telegram.Message attribute), 228
voice_chat_participants-invited (telegram.ext.filters.Filters attribute), 59
voice_chat_participants-invited (telegram.Message attribute), 229
voice_chat-scheduled (telegram.ext.filters.Filters attribute), 58
voice_chat_scheduled (telegram.Message attribute), 228
voice_chat-started (telegram.ext.filters.Filters attribute), 58
voice_chat_started (telegram.Message attribute), 228
voice_duration (telegram.InlineQueryResultVoice attribute), 311
voice_file_id (telegram.InlineQueryResultCachedVoice attribute), 294
voice_url (telegram.InlineQueryResultVoice attribute), 310
VoiceChatEnded (class in telegram), 273
VoiceChatParticipantsInvited (class in telegram), 274
VoiceChatScheduled (class in telegram), 274
VoiceChatStarted (class in telegram), 273
voter_count (telegram.PollOption attribute), 251

W
WAITING (telegram.ext.ConversationHandler attribute), 41
watcher (telegram.ProximityAlertTriggered attribute), 252
wav (telegram.ext.filters.Filters attribute), 52
WebhookInfo (class in telegram), 275
width (telegram.Animation attribute), 93
width (telegram.InputMediaAnimation attribute), 212
width (telegram.InputMediaVideo attribute), 217
width (telegram.PhotoSize attribute), 247
width (telegram.Sticker attribute), 277
width (telegram.Video attribute), 270
workers (telegram.ext.Dispatcher attribute), 10

X
x_shift (telegram.MaskPosition attribute), 279
xml (telegram.ext.filters.Filters attribute), 52
XORFilter (class in telegram.ext.filters), 63

Y
y_shift (telegram.MaskPosition attribute), 279
Z

zip (telegram.ext.filters.Filters attribute), 53