3.2.23 telegram.InputMediaAnimation .................................................. 166
3.2.24 telegram.InputMediaAudio ....................................................... 168
3.2.25 telegram.InputMediaDocument .................................................. 169
3.2.26 telegram.InputMediaPhoto ........................................................ 170
3.2.27 telegram.InputMediaVideo ........................................................ 171
3.2.28 telegram.KeyboardButton ......................................................... 173
3.2.29 telegram.KeyboardButtonPollType .............................................. 173
3.2.30 telegram.Location ................................................................. 174
3.2.31 telegram.LoginUrl ................................................................. 175
3.2.32 telegram.Message ................................................................. 176
3.2.33 telegram.MessageId ............................................................... 194
3.2.34 telegram.MessageEntity .......................................................... 195
3.2.35 telegram.ParseMode ............................................................. 196
3.2.36 telegram.PhotoSize .............................................................. 197
3.2.37 telegram.Poll ...................................................................... 198
3.2.38 telegram.PollAnswer ............................................................. 200
3.2.39 telegram.PollOption ............................................................. 201
3.2.40 telegram.ProximityAlertTriggered ......................................... 201
3.2.41 telegram.ReplyKeyboardRemove .............................................. 202
3.2.42 telegram.ReplyKeyboardMarkup .............................................. 202
3.2.43 telegram.ReplyMarkup ............................................................ 205
3.2.44 telegram.TelegramObject ........................................................ 205
3.2.45 telegram.Update ................................................................. 206
3.2.46 telegram.User .................................................................... 208
3.2.47 telegram.UserProfilePhotos .................................................... 213
3.2.48 telegram.Venue ................................................................. 214
3.2.49 telegram.Video ................................................................. 215
3.2.50 telegram.VideoNote ............................................................ 216
3.2.51 telegram.Voice ................................................................. 218
3.2.52 telegram.WebhookInfo .......................................................... 219
3.2.53 Stickers ......................................................................... 220
3.2.54 Inline Mode ................................................................. 223
3.2.55 Payments ................................................................. 261
3.2.56 Games ................................................................. 268
3.2.57 Passport ................................................................. 270
3.3 telegram.utils package ............................................................. 283
3.3.1 telegram.utils.helpers Module ........................................ 283
3.3.2 telegram.utils.promise.Promise ........................................ 287
3.3.3 telegram.utils.request.Request ............................................ 288
3.3.4 telegram.utils.types Module ............................................... 289
3.4 Changelog ................................................................. 290
3.4.1 Changelog ................................................................. 290

Python Module Index 313

Index 315
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 at our [github in the examples folder](#).
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.Updater

```python
```

**Bases:** object

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

**bot**

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`

**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.
Note:

- You must supply either a bot or a token argument.
- If you supply a bot, you will need to pass defaults to both the bot and the telegram.ext.Updater.

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

```python
idle(stop_signals: Union[List[T], Tuple] = (<Signals.SIGINT: 2>, <Signals.SIGTERM: 15>, <Signals.SIGABRT: 6>) → None
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).

```python
start_polling(poll_interval: float = 0.0, timeout: float = 10, clean: bool = False, bootstrap_retries: int = -1, read_latency: float = 2.0, allowed_updates: List[str] = None) → Optional[queue.Queue]
Starts polling updates from Telegram.
```

Parameters

- **poll_interval** *(float, optional)* – Time to wait between polling updates from Telegram in seconds. Default is 0.0.
- **timeout** *(float, optional)* – Passed to telegram.Bot.get_updates.
- **clean** *(bool, optional)* – Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is False.
- **bootstrap_retries** *(int, optional)* – Whether the bootstrapping phase of the Updater will retry on failures on the Telegram server.
  - < 0 - retry indefinitely (default)
  - 0 - no retries
  - > 0 - retry up to X times
- **allowed_updates** *(List[str], optional)* – Passed to telegram.Bot.get_updates.
- **read_latency** *(float | int, optional)* – Grace time in seconds for receiving the reply from server. Will be added to the timeout value and used as the read timeout from server (Default: 2).

Returns

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

Return type *Queue*

```python
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. To suppress the exception, set force_event_loop to True.
```

Note: Due to an incompatibility of the Tornado library PTB uses for the webhook with Python 3.8+ on Windows machines, PTB will attempt to set the event loop to asyncio.SelectorEventLoop and raise an exception, if an incompatible event loop has already been specified. See this thread for more details. To suppress the exception, set force_event_loop to True.
**Parameters**

- **listen** *(str, optional)* – IP-Address to listen on. Default 127.0.0.1.
- **port** *(int, optional)* – Port the bot should be listening on. Default 80.
- **url_path** *(str, optional)* – Path inside url.
- **cert** *(str, optional)* – Path to the SSL certificate file.
- **key** *(str, optional)* – Path to the SSL key file.
- **clean** *(bool, optional)* – Whether to clean any pending updates on Telegram servers before actually starting the webhook. Default is False.
- **bootstrap_retries** *(int, optional)* – Whether the bootstrapping phase of the Updater will retry on failures on the Telegram server.
  - < 0 - retry indefinitely (default)
  - 0 - no retries
  - > 0 - retry up to X times
- **webhook_url** *(str, optional)* – Explicitly specify the webhook url. Useful behind NAT, reverse proxy, etc. Default is derived from listen, port & url_path.
- **allowed_updates** *(List[str], optional)* – Passed to telegram.Bot.set_webhook.
- **force_event_loop** *(bool, optional)* – Force using the current event loop. See above note for details. Defaults to False

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

**Return type** Queue

`stop()` → None

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

### 3.1.2 telegram.ext.Dispatcher


**Bases:** object

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

**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

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

**Type** telegram.ext.BasePersistence

**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.


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: Update, 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.
add_handler (handler: telegram.ext.handler.Handler, group: int = 0) → None

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: Union[str, Update, None], error: Exception, promise: telegram.utils.promise.Promise = None) → None

Dispatches an error.

Parameters
- update (str | telegram.Update | None) – 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 = None

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 () → telegram.ext.dispatcher.Dispatcher

Get the singleton instance of this class.

Returns telegram.ext.Dispatcher

Raises RuntimeError

groups = None

A list with all groups.

Type List[int]

handlers = None

Holds the handlers per group.

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

process_update (update: Union[str, telegram.update.Update, telegram.error.TelegramError]) → None

Processes a single update.
Parameters **update** (str | telegram.Update | telegram.TelegramError) –

The update to process.

**remove_error_handler** (callback: Callable[[Any, telegram.ext.callbackcontext.CallbackContext], None]) → None

Removes an error handler.

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

**remove_handler** (handler: telegram.ext.handler.Handler, group: int = 0) → None

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: Callable[...], Any, *args, update: Union[str, Update] = None, **kwargs) → telegram.utils.promise.Promise

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() 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, 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** = None

Indicates if this dispatcher is running.

Type bool

**start** (ready: threading.Event = None) → 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** () → None

Stops the thread.

**update_persistence** (update: Union[str, Update] = None) → None

Update **user_data**, **chat_data** and **bot_data** in persistence.

Parameters

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

### 3.1.3 telegram.ext.DispatcherHandlerStop

```python
class telegram.ext.DispatcherHandlerStop(state: object = None)
    Bases: Exception
    Raise this in handler to prevent execution 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:
    ```
    def callback(update, context):
        ...
        raise DispatcherHandlerStop(next_state)
    ```

    state
    Optional. The next state of the conversation.

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

### 3.1.4 telegram.ext.filters Module

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

```python
class telegram.ext.filters.Filters
    Bases: object
    Predefined filters for use as the filter argument of telegram.ext.MessageHandler.
```

Examples

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

```python
all = Filters.all
    All Messages.
```

```python
animation = Filters.animation
    Messages that contain telegram.Animation.
```

```python
audio =Filters.audio
    Messages that contain telegram.Audio.
```

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

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

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

class caption_regex (pattern: Union[str, Pattern[AnyStr]])
    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
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.

class chat (chat_id: Union[int, List[int], Tuple[int, ...]] = None, username: Union[str, List[str], Tuple[str, ...]] = None, allow_empty: bool = False)
    Bases: telegram.ext.filters._ChatUserBaseFilter
    Filters messages to allow only those which are from a specified chat ID or username.

Examples
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.

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.

Type bool, optional

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.

add_chat_ids (chat_id: Union[int, List[int], Tuple[int, ...]]) → None
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: Union[str, List[str], Tuple[str, ...]]) → None
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.

remove_chat_ids (chat_id: Union[int, List[int], Tuple[int, ...]]) → None
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: Union[str, List[str], Tuple[str, ...]]) → None
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:

```python
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 with value 6, use `MessageHandler(Filters.dice(6), callback_method)`. To allow only dice with value 5 or 6, use `MessageHandler(Filters.dice([5, 6]), callback_method)`.

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

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

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`.

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.documents.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")`.

**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.documents.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("video/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 (entity_type: str)
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.

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

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

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

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

`class language(lang: Union[str, List[str], Tuple[str, ...]])`
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’.

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

`passport_data = Filters.passport_data`
Messages that contain `telegram.PassportData`.

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

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

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

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

`class regex(pattern: Union[str, Pattern[AnyStr]])`
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.

```python
reply = Filters.reply
```

Messages that are a reply to another message.

```python
class sender_chat (chat_id: Union[int, List[int], Tuple[int, ...]] = None, username: Union[str, List[str], Tuple[str, ...]] = None, allow_empty: bool = False)
```

Bases: telegram.ext.filters._ChatUserBaseFilter

Filters messages to allow only those which are from a specified sender chats chat ID or username.

Examples

- To filter for messages forwarded from 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(username='anonymous'), callback_method)`.  
- To filter for messages forwarded from 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)`.

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.

```python
chat_ids
```

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

- **Type** set(int), optional

```python
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.

Examples
`Filters.sender_chat.supergroup`

channel
Messages whose sender chat is a channel.

Examples
`Filters.sender_chat.channel`

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 `chat_id` and `username` are both present.

add_chat_ids (`chat_id`: `Union[int, List[int], Tuple[int, ...]]`) → None
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`: `Union[str, List[str], Tuple[str, ...]]`) → None
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`: `Union[int, List[int], Tuple[int, ...]]`) → None
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`: `Union[str, List[str], Tuple[str, ...]]`) → None
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.

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_from_chat_id` or :attr: `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`.

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

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

`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`.  

---

3.1. `telegram.ext` package
• 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.

**update = Filters.update**
Subset for filtering the type of update.

Examples

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

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

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

**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** (user_id: Union[int, List[int], Tuple[int, ...]] = None, username: Union[str, List[str], Tuple[str, ...]] = None, allow_empty: bool = False)
Bases: telegram.ext.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.

**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

Parameters

• user_id(telegram.utils.types.SLT[int], optional) – Which user ID(s)  
to allow through.
• username (telegram.utils.types.SLT[str], optional) – Which user-
name(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.

add_user_ids (user_id: Union[int, List[int], Tuple[int, ...]]) → None
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: Union[str, List[str], Tuple[str, ...]]) → None
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.

remove_user_ids (user_id: Union[int, List[int], Tuple[int, ...]]) → None
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.

remove_usernames (username: Union[str, List[str], Tuple[str, ...]]) → None
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
disabled.

venue = Filters.venue
Messages that contain telegram.Venue.

class via_bot (bot_id: Union[int, List[int], Tuple[int, ...]] = None, username: Union[str, 
List[str], Tuple[str, ...]] = None, allow_empty: bool = False)
Bases: telegram.ext.filters._ChatUserBaseFilter
Filters messages to allow only those which are from specified via_bot ID(s) or username(s).

Examples

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 condi-
tions, as this will complete replace the current set of allowed bots.

bot_ids
Which bot 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 bot is specified in bot_ids and usernames.
Type bool, optional

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.

add_bot_ids (bot_id: Union[int, List[int], Tuple[int, ...]]) → None
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: Union[str, List[str], Tuple[str, ...]]) → None
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.

remove_bot_ids (bot_id: Union[int, List[int], Tuple[int, ...]]) → None
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: Union[str, List[str], Tuple[str, ...]]) → None
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.BaseFilter
Bases: abc.ABC
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.

```python
class telegram.ext.filters.MessageFilter
    Bases: telegram.ext.filters.BaseFilter, abc.ABC

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

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

```python
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
```

```python
class telegram.ext.filters.MessageFilter
    Bases: telegram.ext.filters.BaseFilter, abc.ABC

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

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

```python
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
```

```python
filter(message: telegram.message.Message) -> Union[bool, Dict[KT, VT], None]
    This method must be overwritten.
    Parameters message (telegram.Message) -- The message that is tested.
    Returns dict or bool
```
class telegram.ext.filters.UpdateFilter
    Bases: telegram.ext.filters.BaseFilter, abc.ABC

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

Please see telegram.ext.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

    filter(update: telegram.update.Update) → Union[bool, Dict[KT, VT], None]
        This method must be overwritten.
        Parameters update (telegram.Update) – The update that is tested.
        Returns dict or bool.

class telegram.ext.filters.InvertedFilter (f: telegram.ext.filters.BaseFilter)
    Bases: telegram.ext.filters.UpdateFilter

Represents a filter that has been inverted.

    Parameters f – The filter to invert.

    filter(update: telegram.update.Update) → bool
        This method must be overwritten.
        Parameters update (telegram.Update) – The update that is tested.
        Returns dict or bool.

    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: telegram.update.Update) → Union[bool, Dict[KT, VT]]
        This method must be overwritten.
        Parameters update (telegram.Update) – The update that is tested.
        Returns dict or bool.

    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.

```python
filter(update: telegram.update.Update) → Union[bool, Dict[KT, VT], None]
```

This method must be overwritten.

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

**Returns** *dict or bool.*

### 3.1.5 telegram.ext.Job

**class** telegram.ext.Job(callback: Callable[[CallbackContext], None], context: object = None, name: str = None, job_queue: telegram.ext.jobqueue.JobQueue = None, job: Optional[telegram.ext.jobqueue.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.

**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.

**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

**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.

**enabled**

Whether this job is enabled.

Type bool

**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`

**removed**

Whether this job is due to be removed.

Type bool

**run** (dispatcher: Dispatcher) → None

Executes the callback function independently of the jobs schedule.

**schedule_removal** () → None

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

### 3.1.6 `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: str) → Tuple[telegram.ext.jobqueue.Job, ...]

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

**jobs** () → Tuple[telegram.ext.jobqueue.Job, ...]

Returns a tuple of all pending/scheduled jobs that are currently in the `JobQueue`.

**run_custom** (callback: Callable[[CallbackContext], None], job_kwargs: Dict[str, Any], context: object = None, name: str = None) → telegram.ext.jobqueue.Job

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(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.

- **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.
- **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: Callable[[CallbackContext], None], time: datetime.time, days: Tuple[int, ...] = (0, 1, 2, 3, 4, 5, 6), context: object = None, name: str = None, job_kwargs: Dict[str, Any] = None) → telegram.ext.jobqueue.Job

Creates a new Job that runs on a daily basis and adds it to the queue.

Parameters

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

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

- **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. 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

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

**run_monthly** (callback: Callable[[CallbackContext], None], when: datetime.time, day: int, context: object = None, name: str = None, day_is_strict: bool = True, job_kwargs: Dict[str, Any] = None) → telegram.ext.jobqueue.Job

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

run_once(callback: Callable[[CallbackContext], None], when: Union[float, datetime.timedelta, datetime.datetime, datetime.time], context: object = None, name: str = None, job_kwargs: Dict[ str, Any] = None) → telegram.ext.jobqueue.Job

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:

  ```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 (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: Callable[[CallbackContext], None], interval: Union[float, datetime.timedelta], first: Union[float, datetime.timedelta, datetime.datetime, datetime.time] = None, last: Union[float, datetime.timedelta, datetime.datetime, datetime.time] = None, context: object = None, name: str = None, job_kwargs: Dict[str, Any] = None) → telegram.ext.jobqueue.Job

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

**Parameters**

- **callback** (callable) – The callback function that should be executed by the new job. 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.
  ```

- **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

**Note:** interval is always respected “as-is”. That means that if DST changes during that interval, the job might not run at the time one would expect. It is always recommended to pin servers to UTC time, then time related behaviour can always be expected.
**set_dispatcher** *(dispatcher: Dispatcher) → None*

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** () → None

Starts the job_queue thread.

**stop** () → None

Stops the thread.

### 3.1.7 `telegram.ext.MessageQueue`

**class** `telegram.ext.MessageQueue`

```python
all_burst_limit: int = 30, all_time_limit_ms: int = 1000,
group_burst_limit: int = 20, group_time_limit_ms: int = 60000, exc_route: Callable[[Exception], None] = None,
autostart: bool = True
```

Bases: `object`

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

**Parameters**

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

**__call__** *(promise: Callable, is_group_msg: bool = False) → Callable*

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: int = 30, all_time_limit_ms: int = 1000, group_burst_limit: int = 20, group_time_limit_ms: int = 60000, exc_route: Callable[[Exception], None] = None, autostart: bool = True)
Initialize self. See help(type(self)) for accurate signature.

__weakref__
list of weak references to the object (if defined)

start () ➞ None
Method is used to manually start the MessageQueue processing.

stop (timeout: float = None) ➞ 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.

3.1.8 telegram.ext.DelayQueue

class telegram.ext.DelayQueue (queue: Queue = None, burst_limit: int = 30, time_limit_ms: int = 1000, exc_route: Callable[[Exception], None] = None, autostart: bool = True, name: str = None)
Bases: threading.Thread

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

- 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

- exc_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

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.

3.1. telegram.ext package
• **time_limit_ms** *(int, optional)* – Defines width of time-window used when each processing limit is calculated. Defaults to 1000.

• **exc_route** *(callable, optional)* – A callable, accepting 1 positional argument; used to route exceptions from processor thread to main thread; is called on *Exception* subclass exceptions. If not provided, exceptions are routed through dummy handler, which re-raises them.

• **autostart** *(bool, optional)* – If True, processor is started immediately after object’s creation; if False, should be started manually by *start* method. Defaults to True.

• **name** *(str, optional)* – Thread’s name. Defaults to *'DelayQueue-N'*, where N is sequential number of object created.

**__call__** *(func: Callable, *args, **kwargs) → None*

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: queue.Queue = None, burst_limit: int = 30, time_limit_ms: int = 1000, exc_route: Callable[[Exception], None] = None, autostart: bool = True, name: str = 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() → None**

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

**stop(timeout: float = None) → 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.CallbackContext

**class** *telegram.ext.CallbackContext*(dispatcher: 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*. 3.1.9
Note: `telegram.ext.Dispatcher` will create a single context for an entire update. This means that if you got 2 handlers in different groups and they both get called, they will get passed the same `CallbackContext` object (of course with proper attributes like `.matches` differing). This allows you to add custom attributes in a lower handler group callback, and then subsequently access those attributes in a higher handler group callback. Note that the attributes on `CallbackContext` might change in the future, so make sure to use a fairly unique name for the attributes.

Warning: Do not combine custom attributes and `@run_async`/`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.

**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`

**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`

**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`

**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 `telegram.TelegramError`

**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_kwargs
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`

bot
The bot associated with this context.

Type `telegram.Bot`

dispatcher
The dispatcher associated with this context.

Type `telegram.ext.Dispatcher`

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

Type `telegram.ext.JobQueue`

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`

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

Type `queue.Queue`

3.1.10 telegram.ext.Defaults

class `telegram.ext.Defaults`

```python
```

Bases: `object`

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

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`

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

Type `bool`

disable_web_page_preview
Optional. Disables link previews for links in this message.

Type `bool`
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

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

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

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

Type tzinfo

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

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.

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

• 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.

3.11 Handlers
telegr....

**class** telegram.ext.Handler(callback: Callable[[Union[str, Update], CallbackContext], RT],
pass_update_queue: bool = False, pass_job_queue: bool = False,
pass_user_data: bool = False, pass_chat_data: bool = False,
run_async: Union[bool, telegram.utils.helpers.DefaultValue] =
<telegram.utils.helpers.DefaultValue object>)

**Bases:** abc.ABC

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

**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

**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`. 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`.

**check_update** (`update: Union[str, Update]`) → `Union[bool, object, None]`

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

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

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

**collect_additional_context** (`context: CallbackContext, update: Union[str, Update], dispatcher: Dispatcher, check_result: Any`) → `None`

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: Dispatcher, update: Union[str, Update] = None, check_result: Any = None`) → `Dict[str, Any]`

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

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

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

**handle_update** (`update: Union[str, Update], dispatcher: Dispatcher, check_result: object, context: CallbackContext = None`) → `Union[RT, telegram.utils.promise.Promise]`

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.

---

**telegram.ext.CallbackQueryHandler**

```python
```

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

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

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

**callback**

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 to test `telegram.CallbackQuery.data` against.

  * **Type** str | Pattern

**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

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. DEPRECATED: Please switch to context based callbacks.

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

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

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

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

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

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

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

**check_update** *(update: Union[str, Update]) → Union[bool, object, None]*

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

**Parameters**

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

**Returns**

- **bool**

**collect_additional_context** *(context: CallbackContext, update: Union[str, Update], dispatcher: Dispatcher, check_result: Union[bool, Match[AnyStr]])* → None

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: Dispatcher, update: Union[str, Update] = None, check_result: Union[bool, Match[AnyStr]] = None)* → Dict[str, Any]

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

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

**Parameters**

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

---

**telegram.ext.ChosenInlineResultHandler**


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

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

**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

**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.

**check_update** *(update: Union[str, Update]) → Union[bool, object, None]*)
Determines whether an update should be passed to this handlers `callback`.

Returns **bool**

**telegram.ext.ConversationHandler**

class *telegram.ext.ConversationHandler* (*entry_points*: List[*telegram.ext.handler.Handler*],

  states: Dict[object,

  List[*telegram.ext.handler.Handler*]],

  fallbacks: List[*telegram.ext.handler.Handler*],

  allow_reentry: bool = False, per_chat: bool = True, per_user: bool = True, per_message: bool = False, conversation_timeout: int = None, name: str = None, persistent: bool = False, map_to_parent: Dict[object, object] = None)

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

A handler to hold a conversation with a single user by managing four collections of other handlers.

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.

**entry_points**

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

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

**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*]]
**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]`

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

Type `bool`

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

Type `bool`

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

Type `bool`

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

Type `bool`

**conversation_timeout**
Optional. When this handler is inactive more than this timeout (in seconds), it will be automatically ended. If this value is 0 (default), there will be no timeout. When it’s triggered, the last received update 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`.

Type `float | datetime.timedelta`

**name**
Optional. The name for this conversation handler. Required for persistence

Type `str`

**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`

**map_to_parent**
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.

Type `Dict[object, object]`

**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 conversationkey should contain the Chat’s ID. Default is True.

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

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

**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.

**name** (str, optional) – The name for this conversationhandler. 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 conversationhandler to transition into a mapped state on its parent conversationhandler in place of a specified nested state.

**Raises** ValueError

**END** = -1

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

**TIMEOUT** = -2

Used as a constant to handle state when a conversation is timed out.

**WAITING** = -3

Used as a constant to handle state when a conversation is still waiting on the previous @run_sync decorated running handler to finish.

**check_update** (update: Union[str, Update]) \(\rightarrow\) Optional[Tuple[Tuple[int, ...], telegram.ext.handler.Handler, object]]

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

**Parameters**

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

**Returns** bool

**handle_update** (update: Union[str, Update], dispatcher: Dispatcher, check_result: Optional[Tuple[Tuple[int, ...], telegram.ext.handler.Handler, object]], context: telegram.ext.callbackcontext.CallbackContext = None) \(\rightarrow\) Optional[object]

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

**Parameters**

- **check_result** – The result from check_update. For this handler it’s a tuple of key, handler, and the handler’s check result.

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

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

**END** = -1

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

**TIMEOUT** = -2

Used as a constant to handle state when a conversation is timed out.

**WAITING** = -3

Used as a constant to handle state when a conversation is still waiting on the previous @run_sync decorated running handler to finish.

**check_update**(update: Union[str, Update]) \(\rightarrow\) Optional[Tuple[Tuple[int, ...], telegram.ext.handler.Handler, object]]

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

**Parameters**

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

**Returns** bool

**handle_update**(update: Union[str, Update], dispatcher: Dispatcher, check_result: Optional[Tuple[Tuple[int, ...], telegram.ext.handler.Handler, object]], context: telegram.ext.callbackcontext.CallbackContext = None) \(\rightarrow\) Optional[object]

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

**Parameters**

- **check_result** – The result from check_update. For this handler it’s a tuple of key, handler, and the handler’s check result.

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

- **dispatcher** (telegram.ext.Dispatcher) – Dispatcher that originated the Update.
• **context** (*telegram.ext.CallbackContext*, optional) – The context as provided by the dispatcher.

### `telegram.ext.CommandHandler`

**class telegram.ext.CommandHandler**

```python
class telegram.ext.CommandHandler(command: Union[str, List[str], Tuple[str, ...]],
callback: Callable[[Union[str, Update, CallbackContext], RT], BaseFilter] = None,
allow_edited: bool = None, pass_args: bool = False,
pass_update_queue: bool = False, pass_job_queue: bool = False,
pass_user_data: bool = False, pass_chat_data: bool = False,
```

Bases: `telegram.ext.handler.Handler`

Handler class to handle Telegram commands.

Commands are Telegram messages that start with `/`, optionally followed by an `@` and the bot’s name and/or some additional text. 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:** `telegram.ext.CommandHandler` does **not** handle (edited) channel posts.

- **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

**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**

- **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.

**check_update** (update: Union[str, Update]) → Union[bool, Tuple[List[str], Optional[bool]], None]
Determines whether an update should be passed to this handlers callback.

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

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

**Return type** list

**collect_additional_context** (context: CallbackContext, update: Union[str, Update], dispatcher: Dispatcher, check_result: Union[bool, Tuple[List[str], Optional[bool]], None]) → None
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: Dispatcher, update: Union[str, Update] = None, check_result: Union[bool, Tuple[List[str], Optional[bool]], None] = None) → Dict[str, Any]
Prepares the optional arguments. If the handler has additional optional args, it should subclass this method, but remember to call this super method.

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

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

Bases: telegram.ext.handler.Handler

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

callback
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 to test telegram.InlineQuery.query against.

Type str | Pattern

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

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. DEPRECATED: Please switch to context based callbacks.

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

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

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

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

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

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

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

`check_update` (update: Union[str, Update]) → Union[bool, Match[AnyStr], None]

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

Parameters


Returns` bool

`collect_additional_context` (context: CallbackContext, update: Union[str, Update], dispatcher: Dispatcher, check_result: Union[bool, Match[AnyStr], None]) → None

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: Dispatcher, update: Union[str, Update] = None, check_result: Union[bool, Match[AnyStr], None] = None) → Dict[str, Any]`
Prepares the optional arguments. If the handler has additional optional args, it should subclass this method, but remember to call this super method.

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

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

### `telegram.ext.MessageHandler`

class `telegram.ext.MessageHandler`

Bases: `telegram.ext.handler.Handler`

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


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

**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
Determine whether the callback will run asynchronously.
Type bool

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 op-
  erators (& for and, | for or, ~ for not). Default is telegram.ext.filters.
  Filters.update. This defaults to all message_type updates being: message,
  edited_message, channel_post and edited_channel_post. If you don’t
  want or need any of those pass ~Filters.update.* in the filter argument.

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

  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. 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

**check_update** (update: Union[str, Update]) → Union[bool, Dict[str, Any], None]

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

**Parameters**

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

**Returns** bool

**collect_additional_context** (context: CallbackContext, update: Union[str, Update], patcher: Dispatcher, check_result: Union[bool, Dict[str, Any], None]) → None

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.

**telegram.ext.PollAnswerHandler**


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

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

**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

**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. 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 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.
**check_update** *(update: Union[str, Update]) → bool*

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

**Parameters**

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

**Returns** bool

---

**telegram.ext.PollHandler**


Bases: telegram.ext.handler.Handler

Handler class to handle Telegram updates that contain a poll.

**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

**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`. 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`.

```python
check_update(update: Union[str, Update]) -> bool
```

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

**Parameters**

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

**Returns**

- **bool**

---

**telegram.ext.PreCheckoutQueryHandler**

```python
class telegram.ext.PreCheckoutQueryHandler(callback: Callable[[Union[str, Update], CallbackContext], RT],
                                           pass_update_queue: bool = False,
                                           pass_job_queue: bool = False,
                                           pass_user_data: bool = False,
                                           pass_chat_data: bool = False,
                                           run_async: Union[bool, telegram.utils.helpers.DefaultValue] =
                                           <telegram.utils.helpers.DefaultValue object>)
```

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

Handler class to handle Telegram PreCheckout callback queries.

**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

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 DEPRECATED: Please switch to context based callbacks. instance used by the telegram.ext.Updater and telegram.ext.Dispatcher that contains new updates which can be used to insert updates. Default is False.

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

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

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

- run_async (bool) – Determines whether the callback will run asynchronously. Defaults to False.
check_update (update: Union[str, Update]) → bool
Determines whether an update should be passed to this handler's callback.


Returns bool

telegram.ext.PrefixHandler


Bases: telegram.ext.commandhandler.CommandHandler

Handler class to handle custom prefix commands

This is an 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".

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

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

- `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:
  ```python
def callback(update: Update, context: CallbackContext)```
  The return value of the callback is usually ignored except for the special case of `telegram.ext.ConversationHandler`.
- `filters` (telegram.ext.BaseFilter, optional) – A filter inheriting from `telegram.ext.filters.BaseFilter`. Standard filters can be found in `telegram.ext.filters.Filters`. Filters can be combined using bitwise operators (& for and, | for or, ~ for not).
- `pass_args` (bool, optional) – Determines whether the handler should be passed the arguments passed to the command as a keyword argument called `args`. It will contain a list of strings, which is the text following the command split on single or consecutive whitespace characters. Default is False DEPRECATED: Please switch to context based callbacks.
- `pass_update_queue` (bool, optional) – If set to True, a keyword argument called `update_queue` will be passed to the callback function. It will be the Queue instance used by the `telegram.ext.Updater` and `telegram.ext.Dispatcher` that
contains new updates which can be used to insert updates. Default is False. DEPRECATED: Please switch to context based callbacks.

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

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

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

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

```
check_update (update: Union[str, Update]) → Union[bool, Tuple[List[str], Union[bool, Dict[KT, VT], None]], None]
```

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

**Parameters**

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

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

**Return type** list

```
collect_additional_context (context: CallbackContext, update: Union[str, Update],
 dispatcher: Dispatcher, check_result: Union[bool, Tuple[List[str], Optional[bool]], None]) → None
```

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.

**command**

The list of commands this handler should listen for.

**Returns** List[str]

**prefix**

The prefixes that will precede command.

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


**Bases:** `telegram.ext.messagehandler.MessageHandler`

Handler class to handle Telegram updates based on a regex.

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

**pattern**
- 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`

**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.

- **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**

**collect_optional_args** *(dispatcher: Dispatcher, update: Union[str, Update] = None, check_result: Union[bool, Dict[str, Any], None] = None) → Dict[str, Any]*

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

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

Parameters

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

• **check_result** – The result from check_update

**telegram.ext.ShippingQueryHandler**


Bases: telegram.ext.handler.Handler

Handler class to handle Telegram shipping callback queries.

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

**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. 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.

**check_update** (update: Union[str, Update]) → bool

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

**Parameters**

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

**Returns** bool

**telegram.ext.StringCommandHandler**


Bases: telegram.ext.handler.Handler

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

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

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

**command**

The command this handler should listen for.

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

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

run_async
Determines whether the callback will run asynchronously.
Type bool

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 \texttt{telegram.ext.ConversationHandler}.

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

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

• pass_job_queue (bool, optional) – If set to True, a keyword argument called job_queue will be passed to the callback function. It will be a class: \texttt{telegram.ext.JobQueue} instance created by the \texttt{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.

check_update (update: Union[str, Update]) \rightarrow Optional[List[str]]
Determines whether an update should be passed to this handlers callback.

Parameters update (str) – An incoming command.

Returns bool
collect_additional_context\(\text{context: } \text{CallbackContext}, \text{update: } \text{Union}[\text{str, Update}], \text{dispatcher: } \text{Dispatcher}, \text{check_result: } \text{Optional}[\text{List}[\text{str}]] \)
\(\rightarrow \text{None}\)
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\(\text{dispatcher: } \text{Dispatcher, update: } \text{Union}[\text{str, Update}] = \text{None}, \text{check_result: } \text{Optional}[\text{List}[\text{str}]] = \text{None} \)
\(\rightarrow \text{Dict}[\text{str, Any}]\)
Prepares the optional arguments. If the handler has additional optional args, it should subclass this method, but remember to call this super method.

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

Parameters

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

**telegram.ext.StringRegexHandler**

class telegram.ext.StringRegexHandler\(\text{pattern: } \text{Union}[\text{str, Pattern}[\text{AnyStr}]], \text{callback: } \text{Callable}[\text{Union}[\text{str, Update}], \text{CallbackContext}, \text{RT}], \text{pass_groups: } \text{bool} = \text{False}, \text{pass_groupdict: } \text{bool} = \text{False}, \text{pass_update_queue: } \text{bool} = \text{False}, \text{pass_job_queue: } \text{bool} = \text{False}, \text{run_async: } \text{Union}[\text{bool, telegram.utils.helpers.DefaultValue}] = \text{<telegram.utils.helpers.DefaultValue object>}\)

Bases: telegram.ext.handler.Handler

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

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

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

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

**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

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:
  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.

check_update (update: Union[str, Update]) → Optional[Match[AnyStr]]
    Determines whether an update should be passed to this handlers callback.

Parameters update (str) – An incoming command.

Returns bool
**collect_additional_context** (context: CallbackContext, update: Union[str, Update], dispatcher: Dispatcher, check_result: Optional[Match[AnyStr]]) → None

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: Dispatcher, update: Union[str, Update] = None, check_result: Optional[Match[AnyStr]] = None) → Dict[str, Any]

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

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

**Parameters**

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

**telegram.ext.TypeHandler**

**class telegram.ext.TypeHandler**

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

Handler class to handle updates of custom types.

**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

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:

  ```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`.

- **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. Defaults to `False`.

  ```python
  check_update (update: Any) -> bool
  ```

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


  Returns **bool**

3.1.12 Persistence

`telegram.ext.BasePersistence`

```python
class telegram.ext.BasePersistence (store_user_data: bool = True, store_chat_data: bool = True, store_bot_data: bool = True)
```

Bases: `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 means:

- If `store_bot_data` is True you must overwrite `get_bot_data()` and `update_bot_data()`.

- If `store_chat_data` is True you must overwrite `get_chat_data()` and `update_chat_data()`.
• If `store_user_data` is `True` you must overwrite `get_user_data()` and `update_user_data()`.

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

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

| 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.

```
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
```

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
</table>
| • `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`.

```
REPLACED_BOT = 'bot_instance_replaced_by_ptb_persistence'
    Placeholder for `telegram.Bot` instances replaced in saved data.
    Type str

flush() -> None
    Will be called by `telegram.ext.Updater` upon receiving a stop signal. Gives the persistence a chance to finish up saving or close a database connection gracefully. If this is not of any importance just pass will be sufficient.

get_bot_data() -> Dict[Any, Any]
    "Will be called by `telegram.ext.Dispatcher` upon creation with a persistence object. It should return the bot_data if stored, or an empty `dict`.

    Returns The restored bot data.
    Return type dict
```

3.1. `telegram.ext` package
**get_chat_data** () → DefaultDict[int, Dict[Any, Any]]

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

**Returns** The restored chat data.

**Return type** defaultdict

**get_conversations** (name: str) → Dict[Tuple[int, ...], Optional[object]]

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

**Parameters**
- **name** (str) – The handlers name.

**Returns** The restored conversations for the handler.

**Return type** dict

**get_user_data** () → DefaultDict[int, Dict[Any, Any]]

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

**Returns** The restored user data.

**Return type** defaultdict

**insert_bot** (obj: object) → object

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 __slot__ attribute, excluding objects that can’t be copied with copy.copy.

**Parameters**
- **obj** (object) – The object

**Returns** Copy of the object with Bot instances inserted.

**Return type** obj

**classmethod replace_bot** (obj: object) → object

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 __slot__ attribute, excluding objects that can’t be copied with copy.copy.

**Parameters**
- **obj** (object) – The object

**Returns** Copy of the object with Bot instances replaced.

**Return type** obj

**set_bot** (bot: telegram.bot.Bot) → None

Set the Bot to be used by this persistence instance.

**Parameters**
- **bot** (telegram.Bot) – The bot.

**update_bot_data** (data: Dict[KT, VT]) → None

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

**Parameters**
- **data** (dict) – The telegram.ext.dispatcher.bot_data.

**update_chat_data** (chat_id: int, data: Dict[KT, VT]) → None

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

**Parameters**
- **chat_id** (int) – The chat the data might have been changed for.
- **data** (dict) – The telegram.ext.dispatcher.chat_data[chat_id].
**update_conversation** (name: str, key: Tuple[int, ...], new_state: Optional[object]) → None
Will be called when a telegram.ext.ConversationHandler.update_state is called. This allows the storage of the new state in the persistence.

Parameters
- name (str) – The 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: int, data: Dict[KT, VT]) → None
Will be called by the telegram.ext.Dispatcher after a handler has handled an update.

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

**telegram.ext.PicklePersistence**

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

Bases: telegram.ext.basepersistence.BasePersistence

Using python’s builtin pickle for making you 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().

**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 user_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

**single_file**
Optional. When False will store 3 separate files of filename_user_data, filename_chat_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

Parameters

• filename (str) – The filename for storing the pickle files. When single_file is False this will be used as a prefix.
• store_user_data (bool, optional) – Whether user_data should be saved by this persistence class. Default is True.
• store_chat_data (bool, optional) – Whether user_data should be saved by this persistence class. Default is True.
• store_bot_data (bool, optional) – Whether bot_data should be saved by this persistence class. Default is True.
• single_file (bool, optional) – When False will store 3 separate files of filename_user_data, filename_chat_data and filename_conversations. Default is True.
• on_flush (bool, optional) – When True will only save to file when flush() is called and keep data in memory until that happens. When False will store data on any transaction and on call to flush(). Default is False.

flush() → None
Will save all data in memory to pickle file(s).

get_bot_data() → Dict[Any, Any]
Returns the bot_data from the pickle file if it exists or an empty dict.

Returns The restored bot data.
Return type dict

get_chat_data() → DefaultDict[int, Dict[Any, Any]]
Returns the chat_data from the pickle file if it exists or an empty defaultdict.

Returns The restored chat data.
Return type defaultdict

get_conversations(name: str) → Dict[Tuple[int, ...], Optional[object]]
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() → DefaultDict[int, Dict[Any, Any]]
Returns the user_data from the pickle file if it exists or an empty defaultdict.

Returns The restored user data.
Return type defaultdict

update_bot_data(data: Dict[KT, VT]) → None
Will update the bot_data and depending on on_flush save the pickle file.

Parameters data (dict) – The telegram.ext.dispatcher.bot_data.

update_chat_data(chat_id: int, data: Dict[KT, VT]) → None
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.
• **data** (dict) – The `telegram.ext.dispatcher.chat_data[chat_id]`.

**update_conversation** (name: str, key: Tuple[int, ...], new_state: Optional[object]) → None

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: int, data: Dict[K, V]) → None

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** (dict) – The `telegram.ext.dispatcher.user_data[user_id]`.

**telegram.ext.DictPersistence**

```python
class telegram.ext.DictPersistence(store_user_data: bool = True, store_chat_data: bool = True, store_bot_data: bool = True, user_data_json: str = '', chat_data_json: str = '', bot_data_json: str = '', conversations_json: str = '')
```

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

Using python’s dicts 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()`.

**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

**Parameters**

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

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

• **user_data_json** (str, optional) – Json string that will be used to reconstruct user_data on creating this persistence. Default is "".

• **chat_data_json** (str, optional) – Json string that will be used to reconstruct chat_data on creating this persistence. Default is "".

• **bot_data_json** (str, optional) – Json string that will be used to reconstruct bot_data on creating this persistence. Default is "".

• **conversations_json** (str, optional) – Json string that will be used to reconstruct conversation on creating this persistence. Default is "".

**bot_data**
The bot_data as a dict.

*Type dict*

**bot_data_json**
The bot_data serialized as a JSON-string.

*Type str*

**chat_data**
The chat_data as a dict.

*Type dict*

**chat_data_json**
The chat_data serialized as a JSON-string.

*Type str*

**conversations**
The conversations as a dict.

*Type dict*

**conversations_json**
The conversations serialized as a JSON-string.

*Type str*

**get_bot_data** () → Dict[Any, Any]
Returns the bot_data created from the bot_data_json or an empty dict.

*Returns* The restored bot data.

*Return type* dict

**get_chat_data** () → DefaultDict[int, Dict[Any, Any]]
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: str) → Dict[Tuple[int, ...], Optional[object]]
Returns the conversations created from the conversations_json or an empty dict.

*Returns* The restored conversations data.

*Return type* dict

**get_user_data** () → DefaultDict[int, Dict[Any, Any]]
Returns the user_data created from the user_data_json or an empty defaultdict.
Returns  The restored user data.

Return type  defaultdict

```python
def update_bot_data(data: Dict[KT, VT]) -> None
    Will update the bot_data (if changed).
```

Parameters

- **data** (dict) – The telegram.ext.dispatcher.bot_data.

```python
def update_chat_data(chat_id: int, data: Dict[KT, VT]) -> None
    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].

```python
def update_conversation(name: str, key: Tuple[int, ...], new_state: Optional[object]) -> None
    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.

```python
def update_user_data(user_id: int, data: Dict[KT, VT]) -> None
    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].

```python
user_data
    The user_data as a dict.

    Type  dict
```

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

    Type  str
```

### 3.2 telegram package

#### 3.2.1 telegram.Animation

```python
class telegram.Animation(file_id: str, file_unique_id: str, width: int, height: int, duration: int,
    thumb: telegram.files.photosize.PhotoSize = None, file_name: str = None, mime_type: str = None,
    file_size: int = None, bot: 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.

```python
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

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.
get_file(timeout: int = None, api_kwargs: Dict[str, Any] = None) → File

Convenience wrapper over `telegram.Bot.get_file`

Parameters

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

Returns `telegram.File`

Raises `telegram.TelegramError`

### 3.2.2 telegram.Audio

class `telegram.Audio` (`file_id`: `str`, `file_unique_id`: `str`, `duration`: `int`, `performer`: `str` = `None`, `title`: `str` = `None`, `mime_type`: `str` = `None`, `file_size`: `int` = `None`, `thumb`: `telegram.files.photosize.PhotoSize` = `None`, `bot`: `Bot` = `None`, `file_name`: `str` = `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.

**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`

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.

get_file

Convenience wrapper over `telegram.Bot.get_file`

Parameters

- `timeout` (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- `api_kwargs` (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns `telegram.File`

Raises `telegram.TelegramError`

3.2.3 telegram.Bot

class telegram.Bot (token: str, base_url: str = None, base_file_url: str = None, request: Request = None, private_key: bytes = None, private_key_password: bytes = None, defaults: Defaults = None)

Bases: telegram.base.TelegramObject

This object represents a Telegram Bot.

Parameters

- `token` (str) – Bot’s unique authentication.
- `base_url` (str, optional) – Telegram Bot API service URL.
• `base_file_url` *(str, optional)* – Telegram Bot API file URL.

• `request` *(telegram.utils.request.Request, optional)* – Pre initialized `telegram.utils.request.Request`.

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

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

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

**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.

```python
```

Alias for `add_sticker_to_set`

```python
```

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` or `tgs_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 | 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.

- `tgs_sticker` *(str | filelike object | pathlib.Path, optional)* – TGS animation with the sticker, uploaded using multipart/form-data. See https://core.telegram.org/animated_stickers#technical-requirements for technical requirements.

- `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.TelegramError`

```python
def answerCallbackQuery(callback_query_id: str, text: str = None, show_alert: bool = False, url: str = None, cache_time: int = None, timeout: float = None, api_kwargs: Dict[str, Any] = None) -> bool
```

**Answer Callback Query**

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.TelegramError`

```python
def answer_inline_query(inline_query_id: str, results: List[telegram.inline.inlinequeryresult.InlineQueryResult],
cache_time: int = 300, is_personal: bool = None, next_offset: str = None,
switch_pm_text: str = None, switch_pm_parameter: str = None,
timeout: float = None, current_offset: str = None, api_kwargs: Dict[str, Any] = None) -> bool
```

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.InlineResult` 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.TelegramError

### answer_pre_checkout_query

```
pre_checkout_query_id: str, ok: bool, error_message: str = None, timeout: float = None, api_kwargs: Dict[str, Any] = None) -> bool
```

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

**Note:** The Bot API must receive an answer within 10 seconds after the pre-checkout query was sent.

**Parameters**

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

### answer_shipping_query

```
```

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

**Parameters**

- **shipping_query_id** (str) – Unique identifier for the query to be answered.
- **ok** (bool) – Specify True if delivery to the specified address is possible and False if there are any problems (for example, if delivery to the specified address is not possible).
• **shipping_options** (List[telegram.ShippingOption]) – Required if ok is True. A JSON-serialized array of available shipping options.

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

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

• **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.TelegramError

---

**can_join_groups**
Bot’s can_join_groups attribute.

**Type** bool

**can_read_all_group_messages**
Bot’s can_read_all_group_messages attribute.

**Type** bool

**close() → bool**
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.

**Returns**
On success

**Return type** True

**Raises**
telegram.TelegramError

**commands**
Bot’s commands.

**Type** List[BotCommand]

**copyMessage**

```python
```

**Alias for**
copy_message

**copy_message**

```python
```

Use this method to copy messages of any kind. The method is analogous to the method forwardMes-
messages, but the copied message doesn’t have a link to the original message. Returns the MessageId of
the sent message on success.

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.

- **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 mes-
  sage should be sent even if the specified replied-to message is not found.

- **reply_markup**(telegram.ReplyMarkup, optional) – Additional interface op-
  tions. A JSON-serialized object for an inline keyboard, custom reply keyboard, in-
  structions 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_kwars**(dict, optional) – Arbitrary keyword arguments to be passed to the
  Telegram API.

Returns On success

Return type telegram.MessageId

Raises telegram.TelegramError

createNewStickerSet(user_id: Union[str, int], name: str, title: str, emojis: str, png_sticker:
  Union[str, IO, InputFile, pathlib.Path] = None, contains_masks: bool =
  None, mask_position: telegram.files.sticker.MaskPosition = None, timeout:
  float = 20, tgs_sticker: Union[str, IO, InputFile, pathlib.Path] =
  None, api_kwars: Dict[str, Any] = None) → bool

Alias for create_new_sticker_set

create_new_sticker_set(user_id: Union[str, int], name: str, title: str, emojis: str, png_sticker:
  Union[str, IO, InputFile, pathlib.Path] =
  None, contains_masks: bool = None, mask_position: telegram.files.sticker.MaskPosition = None, timeout:
  float = 20, tgs_sticker: Union[str, IO, InputFile, pathlib.Path] = None,
  api_kwars: Dict[str, Any] = None) → bool

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 png_sticker or tgs_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 png_sticker and tgs_sticker argument can be either a file_id, an URL or a file from disk
do open(filename, 'rb')

Parameters

- **user_id**(int) – User identifier of created sticker set owner.
- **name**(str) – Short name of sticker set, to be used in t.me/addstickers/ URLs (e.g., animals). Can contain only English letters, digits and underscores. Must begin with a letter, can’t contain consecutive underscores and must end in ”_by_<bot_username>“.
  <bot_username> is case insensitive. 1-64 characters.
- **title**(str) – Sticker set title, 1-64 characters.
- **png_sticker**(str | filelike object | 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.
- **tgs_sticker**(str | filelike object | pathlib.Path, optional) – TGS animation
  with the sticker, uploaded using multipart/form-data. See
  https://core.telegram.org/
  animated_stickers#technical-requirements for technical requirements.
- **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.TelegramError

def deleteChatPhoto(chat_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) -> bool
Alias for delete_chat_photo

def deleteChatStickerSet(chat_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) -> bool
Alias for delete_chat_sticker_set

def deleteMessage(chat_id: Union[str, int], message_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) -> bool
Alias for delete_message

def deleteStickerFromSet(sticker: str, timeout: float = None, api_kwargs: Dict[str, Any] = None) -> bool
Alias for delete_sticker_from_set
deleteWebhook (timeout: float = None, api_kwargs: Dict[str, Any] = None, drop_pending_updates: bool = None) → bool

Alias for delete_webhook

delete_chat_photo (chat_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool

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.TelegramError

delete_chat_sticker_set (chat_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool

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: Union[str, int], message_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool

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 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 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_kwargs` (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns On success, True is returned.

Return type bool

Raises telegram.TelegramError

delete_sticker_from_set(sticker: str, timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool

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.TelegramError

delete_webhook(timeout: float = None, api_kwargs: Dict[str, Any] = None, drop_pending_updates: bool = None) → bool

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

Parameters

• `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.

Returns bool On success, True is returned.

Raise telegram.TelegramError


3.2. telegram package
Alias for `edit_message_caption`

```python
```

Alias for `edit_message_live_location`

```python
```

Alias for `edit_message_media`

```python
```

Alias for `edit_message_reply_markup`

```python
```

Alias for `edit_message_text`

```python
```

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.TelegramError

### edit_message_live_location

```python
```

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

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

### Parameters

- **chat_id** (int | str, 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**


gram.files.inputmedia.InputMedia = None, reply_markup: tele

gram.inline.inlinekeyboardmarkup.InlineKeyboardMarkup = None, time


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.TelegramError**

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.TelegramError**


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.TelegramError`

`exportChatInviteLink (chat_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) -> str`

Alias for `export_chat_invite_link`

Use this method to generate a new 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.

Returns New invite link on success.

Return type `str`

Raises `telegram.TelegramError`

`first_name`

Bot’s first name.

Type `str`

`forwardMessage (chat_id: Union[int, str], from_chat_id: Union[str, int], message_id: Union[str, int], disable_notification: bool = False, timeout: float = None, api_kwargs: Dict[str, Any] = None) -> Optional[telegram.message.Message]`

Alias for `forward_message`

Use this method to forward messages of any kind.

Parameters

• `chat_id` (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **from_chat_id** *(int | str)* – Unique identifier for the chat where the original message was sent (or channel username in the format @channelusername).

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

- **message_id** *(int)* – Message identifier in the chat specified in from_chat_id.

- **timeout** *(int | float, optional)* – If this value is specified, use it as 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.TelegramError`

getChat *(chat_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) → telegram.chat.Chat*

Alias for `get_chat`

getChatAdministrators *(chat_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) → List[telegram.chatmember.ChatMember]*

Alias for `get_chat_administrators`

getChatMember *(chat_id: Union[str, int], user_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) → telegram.chatmember.ChatMember*

Alias for `get_chat_member`

getChatMembersCount *(chat_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) → int*

Alias for `get_chat_members_count`


Alias for `get_file`


Alias for `get_game_high_scores`

getMe *(timeout: int = None, api_kwargs: Dict[str, Any] = None) → Optional[telegram.user.User]*

Alias for `get_me`


Alias for `get_my_commands`

getStickerSet *(name: str, timeout: float = None, api_kwargs: Dict[str, Any] = None) → telegram.files.sticker.StickerSet*

Alias for `get_sticker_set`


Alias for `get_updates`
getUserProfilePhotos (user_id: Union[str, int], offset: int = None, limit: int = 100, timeout: float = None, api_kwargs: Dict[str, Any] = None) → Optional[telegram.userprofilephotos.UserProfilePhotos]

Alias for get_user_profile_photos

getWebhookInfo (timeout: float = None, api_kwargs: Dict[str, Any] = None) → telegram.webhookinfo.WebhookInfo

Alias for get_webhook_info

get_chat (chat_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) → telegram.chat.Chat

Use this method to get up to date information about the chat (current name of the user for one-on-one conversations, current username of a user, group or channel, etc.).

Parameters

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

Returns **telegram.Chat**

Raises **telegram.TelegramError**

get_chat_administrators (chat_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) → List[telegram.chatmember.ChatMember]

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 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.TelegramError**

get_chat_member (chat_id: Union[str, int], user_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) → telegram.chatmember.ChatMember

Use this method to get information about a member of a chat.

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **user_id** (int) – Unique identifier of the target user.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
API functions:

- **get_chat_members_count**
  
  Use this method to get the number of members in a chat.

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

  **Returns**
  Number of members in the chat.

  **Return type**
  int

- **get_file**
  
  Use this method to get basic info about a file and prepare it for downloading. For the moment, bots can download files of up to 20MB in size. The file can then be downloaded with `telegram.File.download`. It is guaranteed that the link will be valid for at least 1 hour. When the link expires, a new one can be requested by calling `get_file` again.

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

  **Returns**
  telegram.File

- **get_game_high_scores**
  
  Use this method to get top N scores in a game or on a certain server.

  **Parameters**
  
  - **user_id** (int | str) – Unique identifier of the user who sent a message.
  - **chat_id** (str | int) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
  - **message_id** (int | str) – Identifier of a message sent to a game app to get its score list.
  - **inline_message_id** (str | int) – Identifier of an inline message sent to a game app to get its score list.
  - **timeout** (int | float, optional) – If 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.games.gamehighscore.GameHighScore]
Use this method to get data for high score tables. Will return the score of the specified user and several of his neighbors in a game.

**Parameters**

- **user_id** (int) – 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.TelegramError

**get_me** *(timeout: int = None, api_kwargs: Dict[str, Any] = None) → Optional[telegram.user.User]*

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_kwargs** (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.TelegramError


Use this method to get the current list of the bot’s commands.

**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** On success, the commands set for the bot

**Return type** List[telegram.BotCommand]

**Raises** telegram.TelegramError

**get_sticker_set** *(name: str, timeout: float = None, api_kwargs: Dict[str, Any] = None) → telegram.files.sticker.StickerSet*

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_kwargs** (*dict*, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns** `telegram.StickerSet`

**Raises** `telegram.TelegramError`

`get_updates` (*offset: int = None*, *limit: int = 100*, *timeout: float = 0*, *read_latency: float = 2.0*, *allowed_updates: List[str] = None*, *api_kwargs: Dict[str, Any] = None*) → `List[telegram.update.Update]`

Use this method to receive incoming updates using long polling.

**Parameters**

• **offset** (*int*, optional) – Identifier of the first update to be returned. Must be greater by one than the highest among the identifiers of previously received updates. By default, updates starting with the earliest unconfirmed update are returned. An update is considered confirmed as soon as getUpdates is called with an offset higher than its update_id. The negative offset can be specified to retrieve updates starting from -offset update from the end of the updates queue. All previous updates will forgotten.

• **limit** (*int*, optional) – Limits the number of updates to be retrieved. Values between 1-100 are accepted. Defaults to 100.

• **timeout** (*int*, optional) – Timeout in seconds for long polling. Defaults to 0, i.e. usual short polling. Should be positive, short polling should be used for testing purposes only.

• **allowed_updates** (*List[str]*) (*optional*) – 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 regardless of type (default). If not specified, the previous setting will be used. Please note that this parameter doesn’t affect updates created before the call to the get_updates, so unwanted updates may be received for a short period of time.

• **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.TelegramError`

`get_user_profile_photos` (*user_id: Union[str, int]*, *offset: int = None*, *limit: int = 100*, *timeout: float = 0*, *api_kwargs: Dict[str, Any] = None*) → `Optional[telegram.userprofilephotos.UserProfilePhotos]`

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.TelegramError`

---

**get_webhook_info** (timeout: float = None, api_kwargs: Dict[str, Any] = None) → `telegram.WebhookInfo`

Use this method to get current webhook status. 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_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns** `telegram.WebhookInfo`

**id**

Unique identifier for this bot.

**Type** int

**kickChatMember** (chat_id: Union[str, int], user_id: Union[str, int], timeout: float = None, until_date: Union[int, datetime.datetime] = None, api_kwargs: Dict[str, Any] = None) → bool

Alias for `kick_chat_member`

**kick_chat_member** (chat_id: Union[str, int], user_id: Union[str, int], timeout: float = None, until_date: Union[int, datetime.datetime] = None, api_kwargs: Dict[str, Any] = None) → bool

Use this method to kick a user from a group or a 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 group for this to work.

**Parameters**

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

• **user_id** (int) – Unique identifier of the target user.

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

• **until_date** (int | datetime.datetime, optional) – Date when the user will be unbanned, unix time. If user is banned for more than 366 days or less than 30 seconds from the current time they are considered to be banned forever. For timezone naive `datetime.datetime` objects, the default timezone of the bot will be used.
• **api_kwargs** *(dict, optional)* – Arbitrary keyword arguments to be passed to the Telegram API.

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

**Raises** `telegram.TelegramError`

**last_name**
Optional. Bot’s last name.

**Type** `str`

**leaveChat** *(chat_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool*  
Alias for `leave_chat`

**leave_chat** *(chat_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool*  
Use this method for your bot to leave a group, supergroup or channel.

**Parameters**

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

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

**Raises** `telegram.TelegramError`

**link**
Convenience property. Returns the t.me link of the bot.

**Type** `str`

**logOut** () → `bool`
Alias for `log_out`

**log_out** () → `bool`
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.

**Returns** On success

**Return type** `True`

**Raises** `telegram.TelegramError`

**name**
Bot’s @username.

**Type** `str`

**pinChatMessage** *(chat_id: Union[str, int], message_id: Union[str, int], disable_notification: bool = None, timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool*  
Alias for `pin_chat_message`

**pin_chat_message** *(chat_id: Union[str, int], message_id: Union[str, int], disable_notification: bool = None, timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool*  
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
can_pin_messages admin right in a supergroup or 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.TelegramError

**promoteChatMember**

```python
promoteChatMember(chat_id: Union[str, int], user_id: Union[str, int], can_change_info: bool = None, can_post_messages: bool = None, can_edit_messages: bool = None, can_delete_messages: bool = None, can_invite_users: bool = None, can_restrict_members: bool = None, can_pin_messages: bool = None, can_promote_members: bool = None, timeout: float = None, api_kwargs: Dict[str, Any] = None, is_anonymous: bool = None) -> bool
```

Alias for **promote_chat_member**

**promote_chat_member**

```python
promote_chat_member(chat_id: Union[str, int], user_id: Union[str, int], can_change_info: bool = None, can_post_messages: bool = None, can_edit_messages: bool = None, can_delete_messages: bool = None, can_invite_users: bool = None, can_restrict_members: bool = None, can_pin_messages: bool = None, can_promote_members: bool = None, timeout: float = None, api_kwargs: Dict[str, Any] = None, is_anonymous: bool = None) -> bool
```

Use this method to promote or demote a user in a supergroup or a channel. The bot must be an administrator in the chat for this to work and must have the appropriate admin rights. Pass False for all boolean parameters to demote a user.

Parameters

- **chat_id**(int | str) – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername).
- **user_id**(int) – Unique identifier of the target user.
- **is_anonymous**(bool, optional) – Pass True, if the administrator’s presence in the chat is hidden.
- **can_change_info**(bool, optional) – Pass True, if the administrator can change chat title, photo and other settings.
- **can_post_messages**(bool, optional) – Pass True, if the administrator can create channel posts, channels only.
- **can_edit_messages**(bool, optional) – Pass True, if the administrator can edit messages of other users, channels only.
- **can_delete_messages**(bool, optional) – Pass True, if the administrator can delete messages of other users.
• `can_invite_users` (bool, optional) – Pass `True`, if the administrator can invite new users to the chat.

• `can_restrict_members` (bool, optional) – Pass `True`, if the administrator can restrict, ban or unban chat members.

• `can_pin_messages` (bool, optional) – Pass `True`, if the administrator can pin messages, supergroups only.

• `can_promote_members` (bool, optional) – Pass `True`, if the administrator can add new administrators with a subset of his own privileges or demote administrators that he has promoted, directly or indirectly (promoted by administrators that were appointed by him).

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

• `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.TelegramError`

`restrictChatMember` (chat_id: Union[str, int], user_id: Union[str, int], permissions: telegram.chatpermissions.ChatPermissions, until_date: Union[int, datetime.datetime] = None, timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool

Alias for `restrict_chat_member`

`restrict_chat_member` (chat_id: Union[str, int], user_id: Union[str, int], permissions: telegram.chatpermissions.ChatPermissions, until_date: Union[int, datetime.datetime] = None, timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool

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.TelegramError


Alias for send_animation


Alias for send_audio

sendChatAction (chat_id: Union[str, int], action: str, timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool

Alias for send_chat_action


Alias for send_contact


Alias for send_dice


Alias for send_document

Alias for send_game


Alias for send_invoice


Alias for send_location


Alias for send_media_group


Alias for send_message


Alias for send_photo


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 | 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.
- **duration**(int, optional) – Duration of sent animation in seconds.
- **width**(int, optional) – Animation width.
- **height**(int, optional) – Animation height.
- **thumb**(filelike object | 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.
- **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.
- **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.TelegramError


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 sendVoice method instead.

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

Parameters

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

• audio (str | filelike object | 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.

• 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.

• 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 | 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 only be uploaded as a new file.

• **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.TelegramError*

### send_chat_action

```
send_chat_action(chat_id: Union[str, int], action: str, timeout: float = None, api_kwargs: Dict[str, Any] = None) -> bool
```

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*.

• **timeout** (*int | float, optional*) – If 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.TelegramError*

### send_contact

```
send_contact(chat_id: Union[int, str], phone_number: str = None, first_name: str = None, last_name: str = None, disable_notification: bool = False, reply_to_message_id: Union[int, str] = None, reply_markup: telegram.replymarkup.ReplyMarkup = None, timeout: float = None, contact: telegram.files.contact.Contact = None, vcard: str = None, api_kwargs: Dict[str, Any] = None, allow_sending_without_reply: bool = None) -> Optional[telegram.message.Message]
```

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.

**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.TelegramError


Use this method to send an animated emoji, which will have a random value. On success, the sent Message is returned.

Parameters

**chat_id** (int | str) – Unique identifier for the target private chat.

**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 "️️️️️".

**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.

**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.TelegramError


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 | 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.

• **filename**(str, optional) – File name that shows in telegram message (it is useful when you send file generated by temp module, for example). Undocumented.

• **caption**(str, optional) – Document caption (may also be used when resending documents by file_id), 0-1024 characters 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.

• **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 | 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.

**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.TelegramError`

---


Use this method to send a game.

**Parameters**

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

- **game_short_name** *(str)* – Short name of the game, serves as the unique identifier for the game. Set up your games via @BotFather.

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

- **reply_to_message_id** *(int, optional)* – If the message is a reply, ID of the original message.

- **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.TelegramError`

Use this method to send invoices.

Parameters

- **chat_id** (int | str) – Unique identifier for the target private chat.
- **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.
- **start_parameter** (str) – Unique deep-linking parameter that can be used to generate this invoice when used as a start parameter.
- **currency** (str) – Three-letter ISO 4217 currency code.
- **prices** (List[telegram.LabeledPrice]) – Price breakdown, a JSON-serialized list of components (e.g. product price, tax, discount, delivery cost, delivery tax, bonus, etc.).
- **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.

• `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.TelegramError`

```python
```

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.

• **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.TelegramError`

### send_media_group

```python
@bot.message_handler(func=lambda message: message.chat.type == 'group')
def group_message(message):
    media_group = message.reply('group message', media_group=True)
```

**Parameters**

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

• **media** *(List[Union[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.

• **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 `Message`

**Return type**;
`List[Optional[telegram.message.Message]]`

**Raises**;
`telegram.TelegramError`

3.2. *telegram* package

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.
- **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.TelegramError


Use this method to send photos.
Note: The photo argument can be either a file_id, an URL or a file from disk `open(filename, 'rb')`

Parameters

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

- **photo** (str | filelike object | 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.

- **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.

- **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.TelegramError`


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, 1-255 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.
- **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.
- **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.TelegramError`

Use this method to send static .WEBP or animated .TGS stickers.

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

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target channel (in the format @channelusername).
- **sticker** (str | filelike object | 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.
- **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.
- **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.TelegramError


Use this method to send information about a venue.

Note:

- You can either supply venue, or latitude, longitude, title and address and optionally foursquare_id and foursquare_type or optionally google_place_id and google_place_type.
- Foursquare details and Google Pace 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.
- **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.TelegramError

**send_video**

```
```

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 | 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.
- `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.
- `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 | 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.
- `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`

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 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_note(str | filelike object | 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.
- 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.
- 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 | 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.
- 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.TelegramError

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 | 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.

- **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.

- **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.TelegramError*
setChatAdministratorCustomTitle(chat_id: Union[int, str], user_id: Union[int, str],
custom_title: str, timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool

Alias for set_chat_administrator_custom_title

setChatDescription(chat_id: Union[str, int], description: str, timeout: float = None,
api_kwargs: Dict[str, Any] = None) → bool

Alias for set_chat_description

setChatPermissions(chat_id: Union[str, int], permissions: telegram.chatpermissions.ChatPermissions,
timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool

Alias for set_chat_permissions

setChatPhoto(chat_id: Union[str, int], photo: Union[str, IO, InputFile, pathlib.Path],
timeout: float = 20, api_kwargs: Dict[str, Any] = None) → bool

Alias for set_chat_photo

setChatStickerSet(chat_id: Union[str, int], sticker_set_name: str,
timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool

Alias for set_chat_sticker_set

setChatTitle(chat_id: Union[str, int], title: str, timeout: float = None,
api_kwargs: Dict[str, Any] = None) → bool

Alias for set_chat_title

setGameScore(user_id: Union[int, str], score: int, chat_id: Union[str, int] = None,
message_id: Union[str, int] = None, inline_message_id: Union[str, int] = None,
force: bool = None, disable_edit_message: bool = None, timeout: float = None,

Alias for set_game_score

setMyCommands(commands: List[Union[telegram.botcommand.BotCommand, Tuple[str, str]]],
timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool

Alias for set_my_commands

setPassportDataErrors(user_id: Union[str, int], errors: List[telegram.passport.passportelementerrors.PassportElementError],
timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool

Alias for set_passport_data_errors

setStickerPositionInSet(sticker: str, position: int, timeout: float = None,
api_kwargs: Dict[str, Any] = None) → bool

Alias for set_sticker_position_in_set

setStickerSetThumb(name: str, user_id: Union[str, int], thumb: Union[str, IO, InputFile, pathlib.Path] = None,
timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool

Alias for set_sticker_set_thumb

setWebhook(url: str = None, certificate: Union[str, IO, InputFile, pathlib.Path] = None,
timeout: float = None, max_connections: int = 40, allowed_updates: List[str] = None,
api_kwargs: Dict[str, Any] = None, ip_address: str = None, drop_pending_updates: bool = None) → bool

Alias for set_webhook

set_chat_administrator_custom_title(chat_id: Union[int, str], user_id: Union[int, str],
custom_title: str, timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool

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) – 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.TelegramError`

`set_chat_description` (chat_id: Union[str, int], description: str, timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool
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.
Return type bool
Raises `telegram.TelegramError`

`set_chat_permissions` (chat_id: Union[str, int], permissions: telegram.ChatPermissions, timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool
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 `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.TelegramError`

`set_chat_photo` (chat_id: Union[str, int], photo: Union[str, IO, InputFile, pathlib.Path], timeout: float = 20, api_kwargs: Dict[str, Any] = None) → bool
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 | pathlib.Path) – New chat photo.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **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.TelegramError

```python
set_chat_sticker_set(chat_id: Union[str, int], sticker_set_name: str, timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool
```

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

Parameters

- **chat_id** (int | str) – Unique identifier for the target chat or username of the target supergroup (in the format @supergroupusername).
- **sticker_set_name** (str) – Name of the sticker set to be set as the group sticker set.
- **timeout** (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
- **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns On success, True is returned.

Return type bool

```python
set_chat_title(chat_id: Union[str, int], title: str, timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool
```

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
Raises telegram.TelegramError

```python
set_game_score
```


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.TelegramError – If the new score is not greater than the user’s current score in the chat and force is False.

```python
set_my_commands
```


Use this method to change the list of the bot’s 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.

Returns On success

Return type True

Raises telegram.TelegramError
**set_passport_data_errors** *(user_id: Union[int, str], errors: List[telegram.passport.passportelemenerrors.PassportElementError], timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool*

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.TelegramError

**set_sticker_position_in_set** *(sticker: str, position: int, timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool*

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.TelegramError

**set_sticker_set_thumb** *(name: str, user_id: Union[str, int], thumb: Union[str, IO, InputFile, pathlib.Path] = None, timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool*

Use this method to set the thumbnail of a sticker set. Animated thumbnails can be set for animated sticker sets only.

**Parameters**

- **name** *(str) – Sticker set name*
- **user_id** *(int) – User identifier*
- **thumb** *(str, IO, InputFile, pathlib.Path) – An URL or a file from disk*

**Raises** telegram.TelegramError

**Note:** The thumb can be either a file_id, an URL or a file from disk open(filename, 'rb')
• **user_id** (int) – User identifier of created sticker set owner.

• **thumb** (str | filelike object | 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/animated_stickers#technical-requirements](https://core.telegram.org/animated_stickers#technical-requirements) for animated 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 thumbnail can’t be uploaded via HTTP URL.

• **timeout** (int | float, optional) – If 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.TelegramError*

---

**set_webhook**

```python
```

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>](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 regardless of type (default). If not specified, the previous setting will be used. Please note that this parameter doesn’t affect updates created
before the call to the set_webhook, so unwanted updates may be received for a short period of time.

- **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 as long as an outgoing webhook is set up.
2. To use a self-signed certificate, you need to upload your public key certificate using certificate parameter. Please upload as InputFile, sending a String will not work.
3. Ports currently supported for Webhooks: 443, 80, 88, 8443.

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.TelegramError`

```python
```

Alias for `stop_message_live_location`

```python
```

Alias for `stop_poll`

```python
```

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

```
stop_poll(chat_id: Union[int, str], message_id: Union[int, str], reply_markup: telegram.inline.inlinekeyboardmarkup.InlineKeyboardMarkup = None, timeout: float = None, api_kwargs: Dict[str, Any] = None)
```

→ *telegram.poll.Poll*

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.TelegramError*

### supports_inline_queries

Bot's supports_inline_queries attribute.

**Type** *bool*

### unbanChatMember

```
unbanChatMember(chat_id: Union[str, int], user_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None, only_if_banned: bool = None)
```

→ *bool*

Alias for *unban_chat_member*

### unban_chat_member

```
unban_chat_member(chat_id: Union[str, int], user_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None, only_if_banned: bool = None)
```

→ *bool*

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 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 bool On success, True is returned.

Raises telegram.TelegramError

unpinAllChatMessages (chat_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool

Alias for unpin_all_chat_messages


Alias for unpin_chat_message

unpin_all_chat_messages (chat_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None) → bool

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 can_pin_messages admin right in a supergroup or 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

Raises telegram.TelegramError

unpin_chat_message (chat_id: Union[str, int], timeout: float = None, api_kwargs: Dict[str, Any] = None, message_id: Union[str, int] = None) → bool

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 can_pin_messages admin right in a supergroup or 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.TelegramError

Alias for upload_sticker_file


Use this method to upload a .png file with a sticker for later use in create_new_sticker_set and add_sticker_to_set methods (can be used multiple times).

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

Parameters

- **user_id** (int) – User identifier of sticker file owner.
- **png_sticker** (str | filelike object | 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.
- **timeout** (int | float, optional) – If 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.TelegramError

username
Bot’s username.

Type str

3.2.4 telegram.BotCommand

class telegram.BotCommand(command: str, description: str, **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.

command
Text of the command.

Type str

description
Description of the command.

Type str

Parameters

- **command** (str) – Text of the command, 1-32 characters. Can contain only lowercase English letters, digits and underscores.
3.2.5 telegram.CallbackQuery

class telegram.CallbackQuery (id: str, from_user: telegram.user.User, chat_instance: str, message: telegram.message.Message = None, data: str = None, inline_message_id: str = None, game_short_name: str = None, bot: Bot = None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents an incoming callback query from a callback button in an inline keyboard.

If the button that originated the query was attached to a message sent by the bot, the field message will be present. If the button was attached to a message sent via the bot (in inline mode), the field inline_message_id will be present.

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.

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

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
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
• **bot** (telegram.Bot, optional) – The Bot to use for instance methods.

**Note:** After the user presses an inline button, Telegram clients will display a progress bar until you call `answer`. It is, therefore, necessary to react by calling `telegram.Bot.answer_callback_query` even if no notification to the user is needed (e.g., without specifying any of the optional parameters).

```python
answer(*args, **kwargs) → bool

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

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

**Return type** bool

```python
copy_message(chat_id: int, *args, **kwargs) → MessageId

Shortcut for:
update.callback_query.message.copy(
    chat_id,
    from_chat_id=update.message.chat_id,
    message_id=update.message.message_id,
    *args,
    **kwargs)
```

**Returns** On success, returns the MessageId of the sent message.

**Return type** telegram.MessageId

```python
delete_message(*args, **kwargs) → Union[telegram.message.Message, bool]

Shortcut for:
update.callback_query.message.delete(*args, **kwargs)
```

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

**Return type** bool

3.2. telegram package
**edit_message_caption** *(caption: str, *args, **kwargs) → Union[telegram.message.Message, bool]*

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)
```

**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** *(**args, **kwargs) → Union[telegram.message.Message, bool]*

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)
```

**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** *(**args, **kwargs) → Union[telegram.message.Message, bool]*

Shortcut for either:

```python
update.callback_query.message.edit_media(*args, **kwargs)
```

or:

```python
bot.edit_message_media(inline_message_id=update.callback_query.inline_message_id, *args, **kwargs)
```

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

**Return type** *telegram.Message*

**edit_message_reply_markup** *(reply_markup: InlineKeyboardMarkup, *args, **kwargs) → Union[telegram.message.Message, bool]*

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
)
```

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

**Return type** `telegram.Message`

`edit_message_text` *(text: str, *args, **kwargs) → Union[telegram.message.Message, bool]*
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)
```

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

**Return type** `telegram.Message`

`get_game_high_scores` *(args, **kwargs) → List[GameHighScore]*
Shortcut for either:

```python
update.callback_query.message.get_game_high_scores(*args, **kwargs)
```

or:

```python
bot.get_game_high_scores(inline_message_id=update.callback_query.inline_message_id,
*args, **kwargs)
```

**Returns** List[`telegram.GameHighScore`]

`pin_message` *(args, **kwargs) → bool*
Shortcut for:

```python
bot.pin_chat_message(chat_id=message.chat_id,
    message_id=message.message_id,
*args,
**kwargs)
```

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

**Return type** `bool`

`set_game_score` *(args, **kwargs) → Union[telegram.message.Message, bool]*
Shortcut for either:

```python
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)

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(*args, **kwargs) \[\rightarrow \text{Union[telegram.message.Message, bool]}\]

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)

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(*args, **kwargs) \[\rightarrow \text{bool}\]

Shortcut for:

bot.unpin_chat_message(chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs)

Returns On success, True is returned.

Return type bool

### 3.2.6 telegram.Chat

class telegram.Chat(id: int, type: str, title: str = None, username: str = None, first_name: str = None, last_name: str = None, bot: Bot = None, photo: telegram.file.ChatPhoto = None, description: str = None, invite_link: str = None, pinned_message: Message = None, permissions: telegram.chatpermissions.ChatPermissions = None, sticker_set_name: str = None, can_set_sticker_set: bool = None, slow_mode_delay: int = None, bio: str = None, linked_chat_id: int = None, location: telegram.chatlocation.ChatLocation = 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.

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**

Type `str`

**description**
Optional. Description, for groups, supergroups and channel chats.

Type `str`

**invite_link**
Optional. Chat invite link, for supergroups and channel chats.

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`

**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`

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()`.
- `description` (str, optional) – Description, for groups, supergroups and channel chats. Returned only in `telegram.Bot.get_chat()`.
- `invite_link` (str, optional) – Chat invite link, for groups, supergroups and channel chats. Each administrator in a chat generates their own invite links, so the bot must first generate the link using `export_chat_invite_link()`. 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()`.
- `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()`.
- `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.

CHANNEL = 'channel'
telegram.constants.CHAT_CHANNEL

GROUP = 'group'
telegram.constants.CHAT_GROUP

PRIVATE = 'private'
telegram.constants.CHAT_PRIVATE

SUPERGROUP = 'supergroup'
telegram.constants.CHAT_SUPERGROUP

copy_message(*args, **kwargs) → MessageId
Shortcut for:

```python
bot.copy_message(from_chat_id=update.effective_chat.id, *args, **kwargs)
```

Returns On success, instance representing the message posted.

Return type telegram.Message

get_administrators(*args, **kwargs) → List[ChatMember]
Shortcut for:

```python
bot.get_chat_administrators(update.effective_chat.id, *args, **kwargs)
```

Returns A list of administrators in a chat. An Array of telegram.ChatMember objects that contains information about all chat administrators except other bots. If the chat is a group or a supergroup and no administrators were appointed, only the creator will be returned.

Return type List[telegram.ChatMember]

get_member(*args, **kwargs) → ChatMember
Shortcut for:

```python
bot.get_chat_member(update.effective_chat.id, *args, **kwargs)
```

Returns telegram.ChatMember

get_members_count(*args, **kwargs) → int
Shortcut for:

```python
bot.get_chat_members_count(update.effective_chat.id, *args, **kwargs)
```

Returns int

kick_member(*args, **kwargs) → bool
Shortcut for:

```python
bot.kick_chat_member(update.effective_chat.id, *args, **kwargs)
```

Returns If the action was sent successfully.

Return type bool
Note: This method will only work if the All Members Are Admins setting is off in the target group. Otherwise members may only be removed by the group’s creator or by the member that added them.

```python
leave(*args, **kwargs) → bool
Shortcut for:

bot.leave_chat(update.effective_chat.id, *args, **kwargs)
```

**Returns** bool If the action was sent successfully.

```python
link
Convenience property. If the chat has a username, returns a t.me link of the chat.
```

**Type** str

```python
pin_message(*args, **kwargs) → bool
Shortcut for:

bot.pin_chat_message(chat_id=update.effective_chat.id, *args, **kwargs)
```

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

**Return type** bool

```python
send_action(*args, **kwargs) → bool
Alias for send_chat_action
```

```python
send_animation(*args, **kwargs) → Message
Shortcut for:

bot.send_animation(update.effective_chat.id, *args, **kwargs)
```

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

**Return type** telegram.Message

```python
send_audio(*args, **kwargs) → Message
Shortcut for:

bot.send_audio(update.effective_chat.id, *args, **kwargs)
```

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

**Return type** telegram.Message

```python
send_chat_action(*args, **kwargs) → bool
Shortcut for:

bot.send_chat_action(update.effective_chat.id, *args, **kwargs)
```

**Returns** On success.

**Return type** True

```python
send_contact(*args, **kwargs) → Message
Shortcut for:
```

```
```
bot.send_contact(update.effective_chat.id, *args, **kwargs)

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

**Return type**  `telegram.Message`

**send_copy** (*args, **kwargs)  →  MessageId

Shortcut for:

bot.copy_message(chat_id=update.effective_chat.id, *args, **kwargs)

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

**Return type**  `telegram.Message`

**send_dice** (*args, **kwargs)  →  Message

Shortcut for:

bot.send_dice(update.effective_chat.id, *args, **kwargs)

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

**Return type**  `telegram.Message`

**send_document** (*args, **kwargs)  →  Message

Shortcut for:

bot.send_document(update.effective_chat.id, *args, **kwargs)

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

**Return type**  `telegram.Message`

**send_game** (*args, **kwargs)  →  Message

Shortcut for:

bot.send_game(update.effective_chat.id, *args, **kwargs)

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

**Return type**  `telegram.Message`

**send_invoice** (*args, **kwargs)  →  Message

Shortcut for:

bot.send_invoice(update.effective_chat.id, *args, **kwargs)

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

**Return type**  `telegram.Message`

**send_location** (*args, **kwargs)  →  Message

Shortcut for:

bot.send_location(update.effective_chat.id, *args, **kwargs)

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

**Return type**  `telegram.Message`
**send_media_group** (*args, **kwargs) → List[Message]
Shortcut for:

```python
bot.send_media_group(update.effective_chat.id, *args, **kwargs)
```

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

**Return type** List[telegram.Message]

**send_message** (*args, **kwargs) → Message
Shortcut for:

```python
bot.send_message(update.effective_chat.id, *args, **kwargs)
```

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

**Return type** telegram.Message

**send_photo** (*args, **kwargs) → Message
Shortcut for:

```python
bot.send_photo(update.effective_chat.id, *args, **kwargs)
```

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

**Return type** telegram.Message

**send_poll** (*args, **kwargs) → Message
Shortcut for:

```python
bot.send_poll(update.effective_chat.id, *args, **kwargs)
```

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

**Return type** telegram.Message

**send_sticker** (*args, **kwargs) → Message
Shortcut for:

```python
bot.send_sticker(update.effective_chat.id, *args, **kwargs)
```

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

**Return type** telegram.Message

**send_venue** (*args, **kwargs) → Message
Shortcut for:

```python
bot.send_venue(update.effective_chat.id, *args, **kwargs)
```

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

**Return type** telegram.Message

**send_video** (*args, **kwargs) → Message
Shortcut for:

```python
bot.send_video(update.effective_chat.id, *args, **kwargs)
```

**Returns** On success, instance representing the message posted.
**send_video_note** (*args, **kwargs) → Message

Shortcut for:
```python
bot.send_video_note(update.effective_chat.id, *args, **kwargs)
```

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

**Return type** `telegram.Message`

**send_voice** (*args, **kwargs) → Message

Shortcut for:
```python
bot.send_voice(update.effective_chat.id, *args, **kwargs)
```

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

**Return type** `telegram.Message`

**set_administrator_custom_title** (*args, **kwargs) → bool

Shortcut for:
```python
bot.set_chat_administrator_custom_title(update.effective_chat.id, *args, **kwargs)
```

**Returns**: bool: If the action was sent successfully.

**Return type** `bool`

**set_permissions** (*args, **kwargs) → bool

Shortcut for:
```python
bot.set_chat_permissions(update.effective_chat.id, *args, **kwargs)
```

**Returns** If the action was sent successfully.

**Return type** `bool`

**unban_member** (*args, **kwargs) → bool

Shortcut for:
```python
bot.unban_chat_member(update.effective_chat.id, *args, **kwargs)
```

**Returns** If the action was sent successfully.

**Return type** `bool`

**unpin_all_messages** (*args, **kwargs) → bool

Shortcut for:
```python
bot.unpin_all_chat_messages(chat_id=update.effective_chat.id, *args, **kwargs)
```

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

**Return type** `bool`

**unpin_message** (*args, **kwargs) → bool

Shortcut for:
bot.unpin_chat_message(chat_id=update.effective_chat.id, *args, **kwargs)

Returns On success, True is returned.

Return type bool

3.2.7 telegram.ChatAction

class telegram.ChatAction
Bases: object

Helper class to provide constants for different chat actions.

FIND_LOCATION = 'find_location'
telegram.constants.CHATACTION_FIND_LOCATION

RECORD_AUDIO = 'record_audio'
telegram.constants.CHATACTION_RECORD_AUDIO

RECORD_VIDEO = 'record_video'
telegram.constants.CHATACTION_RECORD_VIDEO

RECORD_VIDEO_NOTE = 'record_video_note'
telegram.constants.CHATACTION_RECORD_VIDEO_NOTE

TYPING = 'typing'
telegram.constants.CHATACTION_TYPING

UPLOAD_AUDIO = 'upload_audio'
telegram.constants.CHATACTION_UPLOAD_AUDIO

UPLOAD_DOCUMENT = 'upload_document'
telegram.constants.CHATACTION_UPLOAD_DOCUMENT

UPLOAD_PHOTO = 'upload_photo'
telegram.constants.CHATACTION_UPLOAD_PHOTO

UPLOAD_VIDEO = 'upload_video'
telegram.constants.CHATACTION_UPLOAD_VIDEO

UPLOAD_VIDEO_NOTE = 'upload_video_note'
telegram.constants.CHATACTION_UPLOAD_VIDEO_NOTE

3.2.8 telegram.ChatLocation

class telegram.ChatLocation(location: telegram.files.location.Location, address: str, **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.

location
The location to which the supergroup is connected.

Type telegram.Location

address
Location address, as defined by the chat owner

Type str

146 Chapter 3. Reference
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.

3.2.9 telegram.ChatMember

class telegram.ChatMember(user: telegram.user.User, status: str, until_date: datetime = None, can_be_edited: bool = None, can_change_info: bool = None, can_post_messages: bool = None, can_edit_messages: bool = None, can_delete_messages: bool = None, can_invite_users: bool = None, can_restrict_members: bool = None, can_pin_messages: bool = None, can_promote_members: bool = None, can_send_messages: bool = None, can_send_media_messages: bool = None, can_send_polls: bool = None, can_send_other_messages: bool = None, can_add_web_page_previews: bool = None, is_member: bool = None, custom_title: str = None, is_anonymous: bool = None, **kwargs)

This object contains information about one member of a chat.

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.

**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.

Type str

**is_anonymous**

Optional. True, if the user’s presence in the chat is hidden.

Type bool

**until_date**

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

Type datetime.datetime

**can_be_edited**

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

Type bool

**can_change_info**

Optional. If the user can change the chat title, photo and other settings.

Type bool

**can_post_messages**

Optional. If the administrator can post in the channel.
**Type** bool

**can_edit_messages**
Optional. If the administrator can edit messages of other users.

**Type** bool

**can_delete_messages**
Optional. If the administrator can delete messages of other users.

**Type** bool

**can_invite_users**
Optional. If the user can invite new users to the chat.

**Type** bool

**can_restrict_members**
Optional. If the administrator can restrict, ban or unban chat members.

**Type** bool

**can_pin_messages**
Optional. If the user can pin messages.

**Type** bool

**can_promote_members**
Optional. If the administrator can add new administrators.

**Type** bool

**is_member**
Optional. Restricted only. True, if the user is a member of the chat at the moment of the request.

**Type** bool

**can_send_messages**
Optional. If the user can send text messages, contacts, locations and venues.

**Type** bool

**can_send_media_messages**
Optional. If the user can send media messages, implies can_send_messages.

**Type** bool

**can_send_polls**
Optional. True, if the user is allowed to send polls.

**Type** bool

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

**Type** bool

**can_add_web_page_previews**
Optional. If user may add web page previews to his messages, implies can_send_media_messages

**Type** bool

**Parameters**

- **user** *(telegram.User)* – Information about the user.
- **custom_title** *(str, optional)* – Owner and administrators only. Custom title for this user.
• **is_anonymous** (bool, optional) – Owner and administrators only. True, if the user’s presence in the chat is hidden.

• **until_date** (datetime.datetime, optional) – Restricted and kicked only. Date when restrictions will be lifted for this user.

• **can_be_edited** (bool, optional) – Administrators only. True, if the bot is allowed to edit administrator privileges of that user.

• **can_change_info** (bool, optional) – Administrators and restricted only. True, if the user can change the chat title, photo and other settings.

• **can_post_messages** (bool, optional) – Administrators only. True, if the administrator can post in the channel, channels only.

• **can_edit_messages** (bool, optional) – Administrators only. True, if the administrator can edit messages of other users and can pin messages; channels only.

• **can_delete_messages** (bool, optional) – Administrators only. True, if the administrator can delete messages of other users.

• **can_invite_users** (bool, optional) – Administrators and restricted only. True, if the user can invite new users to the chat.

• **can_restrict_members** (bool, optional) – Administrators only. True, if the administrator can restrict, ban or unban chat members.

• **can_pin_messages** (bool, optional) – Administrators and restricted only. True, if the user can pin messages, groups and supergroups only.

• **can_promote_members** (bool, optional) – Administrators only. True, if the administrator can add new administrators with a subset of his own privileges or demote administrators that he has promoted, directly or indirectly (promoted by administrators that were appointed by the user).

• **is_member** (bool, optional) – Restricted only. True, if the user is a member of the chat at the moment of the request.

• **can_send_messages** (bool, optional) – Restricted only. True, if the user can send text messages, contacts, locations and venues.

• **can_send_media_messages** (bool, optional) – Restricted only. True, if the user can send audios, documents, photos, videos, video notes and voice notes.

• **can_send_polls** (bool, optional) – Restricted only. True, if the user is allowed to send polls.

• **can_send_other_messages** (bool, optional) – Restricted only. True, if the user can send animations, games, stickers and use inline bots.

• **can_add_web_page_previews** (bool, optional) – Restricted only. True, if user may add web page previews to his messages.

```python
ADMINISTRATOR = 'administrator'
telegram.constants.CHATMEMBER_ADMINISTRATOR

CREATOR = 'creator'
telegram.constants.CHATMEMBER_CREATOR

KICKED = 'kicked'
telegram.constants.CHATMEMBER_KICKED

LEFT = 'left'
telegram.constants.CHATMEMBER_LEFT

MEMBER = 'member'
telegram.constants.CHATMEMBER_MEMBER
```
RESTRICTED = 'restricted'

```
 telegram.constants.CHATMEMBER_RESTRICTED
```

### 3.2.10 `telegram.ChatPermissions`

**class** `telegram.ChatPermissions`

```python
can_send_messages: bool = None,
can_send_media_messages: bool = None, can_send_polls:
    bool = None, can_send_other_messages: bool =
    None, can_add_web_page_previews: bool = None,
    can_change_info: bool = None, can_invite_users: bool =
    None, can_pin_messages: bool = 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_message` 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.

- **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

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.

3.2.11 telegram.ChatPhoto


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.

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
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.

### get_big_file

```python
get_big_file(timeout: int = None, **kwargs) -> File
```

Convenience wrapper over `telegram.Bot.get_file` for getting the big (640x640) chat photo

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.File`

Raises `telegram.TelegramError`

### get_small_file

```python
def get_small_file(timeout: int = None, **kwargs) -> File
```

Convenience wrapper over `telegram.Bot.get_file` for getting the small (160x160) chat photo

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.File`

Raises `telegram.TelegramError`

### 3.2.12 telegram.constants Module

Constants in the Telegram network.

The following constants were extracted from the Telegram Bots FAQ and Telegram Bots API.

- `telegram.constants.MAX_MESSAGE_LENGTH`

  Type int

  4096

- `telegram.constants.MAX_CAPTION_LENGTH`

  Type int

  1024
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

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

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.ChatAction:
telegram.constants.CHATACTION_FIND_LOCATION
    'find_location'

Type str
telegram.constants.CHATACTION_RECORD_AUDIO
    'record_audio'

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'

Type str
telegram.constants.CHATACTION_UPLOAD_DOCUMENT
    'upload_document'

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
telegram.constants.CHATMEMBER_CREATOR
    'creator'

Type str
telegram.constants.CHATMEMBER_KICKED
    'kicked'
Type str
telegram.constants.CHATMEMBER_LEFT
    'left'
Type str
telegram.constants.CHATMEMBER_MEMBER
    'member'
Type str
telegram.constants.CHATMEMBER_RESTRICTED
    'restricted'

telegram.Dice:
telegram.constants.DICE_DICE
    ''
Type str
telegram.constants.DICE_DARTS
    ''
Type str
telegram.constants.DICE_BASKETBALL
    ''
Type str
telegram.constants.DICE_FOOTBALL
    ''
Type str
telegram.constants.DICE_SLOT_MACHINE
    ''
Type str
telegram.constants.DICE_ALL_EMOJI
    List of all supported base emoji.
     Type List[str]
telegram.MessageEntity:
telegram.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_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`  
  Type `str`  
  Value 'regular'

- `telegram.constants.POLL_QUIZ`  
  Type `str`  
  Value 'quiz'

- `telegram.constants.MAX_POLL_QUESTION_LENGTH`  
  Type `int`  
  Value 300

- `telegram.constants.MAX_POLL_OPTION_LENGTH`  
  Type `int`  
  Value 100

**telegram.files.MaskPosition:**

- `telegram.constants.STICKER_FOREHEAD`  
  Type `str`  
  Value 'forehead'

- `telegram.constants.STICKER_EYES`  
  Type `str`  
  Value 'eyes'

- `telegram.constants.STICKER_MOUTH`  
  Type `str`  
  Value 'mouth'

- `telegram.constants.STICKER_CHIN`  
  Type `str`  
  Value 'chin'

### 3.2.13 telegram.Contact

**class telegram.Contact**

```python
class telegram.Contact(phone_number: str, first_name: str, last_name: str = None, user_id: int = None, vcard: str = None, **kwargs)
```

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.

- `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

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.

3.2.14 telegram.Dice

class telegram.Dice(value: int, emoji: str, **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 3 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 “”, 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.

value
Value of the dice.
Type int

emoji
Emoji on which the dice throw animation is based.
Type str

Parameters
• value (int) – Value of the dice. 1-6 for dice and darts, 1-5 for basketball and football/soccer ball, 1-64 for slot machine.
• emoji (str) – Emoji on which the dice throw animation is based.

ALL_EMOJI = [‘’, ‘’, ‘’, ‘’, ‘’, ‘’]
telegram.constants.DICE_ALL_EMOJI
BASKETBALL = ''
    telegram.constants.DICE_BASKETBALL
DARTS = ''
    telegram.constants.DICE_DARTS
DICE = ''
    telegram.constants.DICE_DICE
FOOTBALL = ''
    telegram.constants.DICE_FOOTBALL
SLOT_MACHINE = ''
    telegram.constants.DICE_SLOT_MACHINE

3.2.15 telegram.Document

class telegram.Document(file_id: str, file_unique_id: str, thumb: telegram.files.photosize.PhotoSize = None, file_name: str = None, mime_type: str = None, file_size: int = None, bot: 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.

**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

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.

`get_file (timeout: int = None, api_kwargs: Dict[str, Any] = None) → File

Convenience wrapper over `telegram.Bot.get_file

Parameters

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

• `api_kwargs (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns `telegram.File

Raises `telegram.TelegramError

### 3.2.16 telegram.error module

This module contains an object that represents Telegram errors.

**exception** `telegram.error.BadRequest (message: str)

Bases: `telegram.error.NetworkError

**exception** `telegram.error.ChatMigrated (new_chat_id: int)

Bases: `telegram.error.TelegramError

Parameters `new_chat_id (int) – The new chat id of the group.

**exception** `telegram.error.Conflict (message: str)

Bases: `telegram.error.TelegramError

Raised when a long poll or webhook conflicts with another one.

Parameters `msg (str) – The message from telegrams server.

**exception** `telegram.error.InvalidToken

Bases: `telegram.error.TelegramError

**exception** `telegram.error.NetworkError (message: str)

Bases: `telegram.error.TelegramError

**exception** `telegram.error.RetryAfter (retry_after: int)

Bases: `telegram.error.TelegramError

Parameters `retry_after (int) – Time in seconds, after which the bot can retry the request.

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

Bases: `Exception

**exception** `telegram.error.TimedOut

Bases: `telegram.error.NetworkError
exception telegram.error.Unauthorized(message: str)
    Bases: telegram.error.TelegramError

3.2.17 telegram.File

class telegram.File(file_id: str, file_unique_id: str, bot: Bot = None, file_size: int = None, file_path: str = None, **kwargs)
    Bases: telegram.base.TelegramObject

This object represents a file ready to be downloaded. The file can be downloaded with download. It is guaranteed that the link will be valid for at least 1 hour. When the link expires, a new one can be requested by calling 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.

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

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.

Note: If you obtain an instance of this class from telegram.PassportFile.get_file, then it will automatically be decrypted as it downloads when you call download().

download(custom_path: str = None, out: IO = None, timeout: int = None) → Union[str, IO]
    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: bytearray = None) → bytes
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

### 3.2.18 telegram.ForceReply

class telegram.ForceReply (force_reply: bool = True, selective: bool = False, **kwargs)
Bases: telegram.replymarkup.ReplyMarkup

Upon receiving a message with this object, Telegram clients will display a reply interface to the user (act as if the user has selected the bot’s message and tapped ‘Reply’). This can be extremely useful if you want to create user-friendly step-by-step interfaces without having to sacrifice privacy mode.

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

**force_reply**
Shows reply interface to the user, as if they manually selected the bot’s message and tapped ‘Reply’.

**Type** True

**selective**
Optional. Force reply from specific users only.

**Type** bool

**Parameters**
- **selective** (bool, optional) – Use this parameter if you want to force reply from specific users only. Targets:
1) Users that are @mentioned in the text of the Message object.

2) If the bot’s message is a reply (has reply_to_message_id), sender of the original message.

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

3.2.19 telegram.InlineKeyboardButton

class telegram.InlineKeyboardButton (text: str, url: str = None, callback_data: str = None, switch_inline_query: str = None, switch_inline_query_current_chat: str = None, callback_game: CallbackGame = None, pay: bool = None, login_url: LoginUrl = 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.

- **text**
  - Label text on the button.
  - Type str

- **url**
  - Optional. HTTP or tg:// url to be opened when button is pressed.
  - Type str

- **login_url**
  - authorize the user. Can be used as a replacement for the Telegram Login Widget.
  - Type telegram.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

- **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

**Parameters**

- **text** (str) – Label text on the button.
- **url** (str) – HTTP or tg:// url to be opened when button is pressed.
- **login_url** (telegram.LoginUrl, optional) – authorize the user. Can be used as a replacement for the Telegram Login Widget.
- **callback_data** (str, optional) – Data to be sent in a callback query to the bot when button is pressed, UTF-8 1-64 bytes.
- **switch_inline_query** (str, optional) – If set, pressing the button will prompt the user to select one of their chats, open that chat and insert the bot’s username and the specified inline query in the input field. Can be empty, in which case just the bot’s username will be inserted. This offers an easy way for users to start using your bot in inline mode when they are currently in a private chat with it. Especially useful when combined with switch_pm* actions - in this case the user will be automatically returned to the chat they switched from, skipping the chat selection screen.
- **switch_inline_query_current_chat** (str, optional) – If set, pressing the button will insert the bot’s username and the specified inline query in the current chat’s input field. Can be empty, in which case only the bot’s username will be inserted. This offers a quick way for the user to open your bot in inline mode in the same chat - good for selecting something from multiple options.
- **callback_game** (telegram.CallbackGame, optional) – Description of the game that will be launched when the user presses the button. This type of button must always be the first button in the first row.
- **pay** (bool, optional) – Specify True, to send a Pay button. This type of button must always be the first button in the first row.
- ****kw**args** (dict) – Arbitrary keyword arguments.

### 3.2.20 telegram.InlineKeyboardMarkup

**class** telegram.InlineKeyboardMarkup(**inline_keyboard**: List[List[InlineKeyboardButton]], **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 the size of **inline_keyboard** and all the buttons are equal.

**inline_keyboard**

List of button rows, each represented by a list of InlineKeyboardButton objects.

**Type** List[List[telegram.InlineKeyboardButton]]

**Parameters**

- **inline_keyboard** (List[List[InlineKeyboardButton]]) – List of button rows, each represented by a list of InlineKeyboardButton objects.
- ****kw**args** (dict) – Arbitrary keyword arguments.

**classmethod from_button**(button: telegram.inline.inlinekeyboardbutton.InlineKeyboardButton, **kwargs)

Shortcut for:
Return an InlineKeyboardMarkup from a single InlineKeyboardButton

**Parameters**

- **button** (*telegram.InlineKeyboardButton*) – The button to use in the markup
- ****kwargs (dict) – Arbitrary keyword arguments.

```python
InlineKeyboardMarkup([[button]], **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.

```python
classmethod from_column(button_column: List[telegram.inline.inlinekeyboardbutton.InlineKeyboardButton], **kwargs) -> telegram.inline.inlinekeyboardmarkup.InlineKeyboardMarkup
```

Shortcut for:

```python
InlineKeyboardMarkup([[button] for button in button_column], **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
classmethod from_row(button_row: List[telegram.inline.inlinekeyboardbutton.InlineKeyboardButton], **kwargs) -> telegram.inline.inlinekeyboardmarkup.InlineKeyboardMarkup
```

Shortcut for:

```python
InlineKeyboardMarkup([button_row], **kwargs)
```

### 3.2.21 telegram.InputFile

**class telegram.InputFile**

Bases: *object*

This object represents a Telegram InputFile.

**input_file_content**

The binary content of the file to send.

**typename** bytes

**filename**

Optional. Filename for the file to be sent.

**typename** str

**attach**

Optional. Attach id for sending multiple files.

**typename** str

**Parameters**

- **obj** (File handler) – An open file descriptor.
- **filename** (str, optional) – Filename for this InputFile.
- **attach** *(bool, optional)* – Whether this should be send as one file or is part of a collection of files.

**Raises** *TelegramError*

```python
def is_image(stream: bytes) -> Optional[str]
```

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*

### 3.2.22 telegram.InputMedia

**class** *telegram.InputMedia*

**Bases:** *telegram.base.TelegramObject*

Base class for Telegram InputMedia Objects.


### 3.2.23 telegram.InputMediaAnimation

**class** *telegram.InputMediaAnimation*

**Bases:** *telegram.files.inputmedia.InputMedia*

Represents an animation file (GIF or H.264/MPEG-4 AVC video without sound) to be sent.

**Type** *animation.*

**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.

**width**
Optional. Animation width.

**height**
Optional. Animation height.

**duration**
Optional. Animation duration.

**Parameters**
- **media** (str | filelike object | 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.
- **thumb** (filelike object | 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.
- **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.

**Note:** When using a telegram.Animation for the media attribute. It will take the width, height and duration from that video, unless otherwise specified with the optional arguments.
3.2.24 telegram.InputMediaAudio

class telegram.InputMediaAudio:

```
```

Bases: telegram.files.inputmedia.InputMedia

Represents an audio file to be treated as music to be sent.

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

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

Parameters

- media (str | filelike object | 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.

- 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` | `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.

---

**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.

### 3.2.25 telegram.InputMediaDocument

**class telegram.InputMediaDocument**

```python
```

**Bases:** `telegram.files.inputmedia.InputMedia`

Represents a general file to be sent.

**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

Parameters

• media (str | filelike object | 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.

• 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 | 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.

• 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.

3.2.26 telegram.InputMediaPhoto


Bases: telegram.files.inputmedia.InputMedia

type
photo.

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]

### Parameters

- **media** *(str | filelike object | 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.

- **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**.

### 3.2.27 telegram.InputMediaVideo

class telegram.InputMediaVideo

Represents a video to be sent.

**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

Parameters

- **media (str | filelike object | 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.

- **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 | 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.

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.
3.2.28 telegram.KeyboardButton

class telegram.KeyboardButton (text: str, request_contact: bool = None, request_location: bool = None, request_poll: bool = None, **kwargs)

Bases: telegram.base.TelegramObject

This object represents one button of the reply keyboard. For simple text buttons String can be used instead of this object to specify text of the button.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their text, request_contact, request_location and request_poll are equal.

Note: Optional fields are mutually exclusive.

text
   Text of the button.
   Type str
request_contact
   Optional. The user's phone number will be sent.
   Type bool
request_location
   Optional. The user’s current location will be sent.
   Type bool
request_poll
   Optional. If the user should create a poll.
   Type KeyboardButtonPollType

Parameters

* text (str) – Text of the button. If none of the optional fields are used, it will be sent to the bot as a message when the button is pressed.
* request_contact (bool, optional) – If True, the user’s phone number will be sent as a contact when the button is pressed. Available in private chats only.
* request_location (bool, optional) – If True, the user’s current location will be sent when the button is pressed. Available in private chats only.
* request_poll (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.

Note: request_contact and request_location options will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.
request_poll option will only work in Telegram versions released after 23 January, 2020. Older clients will receive unsupported message.

3.2.29 telegram.KeyboardButtonPollType

class telegram.KeyboardButtonPollType (type: str = 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`

### 3.2.30 `telegram.Location`

**class** `telegram.Location`(*longitude: float, latitude: float, horizontal_accuracy: float = None, live_period: int = None, heading: int = None, proximity_alert_radius: int = 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.

**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.

  Type `int`

**heading**

Optional. The direction in which user is moving, in degrees. For active live locations only.

  Type `int`

**proximity_alert_radius**

Optional. Maximum distance for proximity alerts about approaching another chat member, in meters. For sent live locations only.

  Type `int`

**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.

### 3.2.31 telegram>LoginUrl

```python
class telegram.LoginUrl(url: str, forward_text: str = None, bot_username: str = None, request_write_access: bool = 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.

**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`

**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.

**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.
3.2.32 telegram.Message

```python
class telegram.Message(
    message_id: int, date: datetime.datetime, chat: telegram.chat.Chat,

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.

**message_id**

Unique message identifier inside this chat.

Type  int

**from_user**

Optional. Sender.

Type  telegram.User
sender_chat
Optional. Sender of the message, sent on behalf of a chat. The channel itself for channel messages. The supergroup itself for messages from anonymous group administrators. The linked channel for messages automatically forwarded to the discussion group.
  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
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
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.
Type telegram.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).
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

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`

**reply_markup**
Optional. Inline keyboard attached to the message.

Type `telegram.InlineKeyboardMarkup`

**bot**
Optional. The Bot to use for instance methods.

Type `telegram.Bot`

**Parameters**
- **message_id**(int) – Unique message identifier inside this chat.
- **from_user**(telegram.User, optional) – Sender, empty for messages sent to channels.
- **sender_chat**(telegram.Chat, optional) – Sender of the message, sent on behalf of a chat. The channel itself for channel messages. The supergroup itself for messages from anonymous group administrators. The linked channel for messages automatically forwarded to the discussion group.
- **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`.

• **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`.

• **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.

• **document** *(telegram.Document, optional)* – Message is a general file, information about the file.

• **animation** *(telegram.Animation, optional)* – Message is an animation, information about the animation. For backward compatibility, when this field is set, the document field will also be set.

• **game** *(telegram.Game, optional)* – Message is a game, information about the game.

• **photo** *(List[telegram.PhotoSize], optional)* – Message is a photo, available sizes of the photo.

• **sticker** *(telegram.Sticker, optional)* – Message is a sticker, information about the sticker.

• **video** *(telegram.Video, optional)* – Message is a video, information about the video.

• **voice** *(telegram.Voice, optional)* – Message is a voice message, information about the file.

• **video_note** *(telegram.VideoNote, optional)* – Message is a video note, information about the video message.

• **new_chat_members** *(List[telegram.User], optional)* – New members that were added to the group or supergroup and information about them (the bot itself may be one of these members).

• **caption** *(str, optional)* – Caption for the 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.

• **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.
• **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.

**caption_html**
Creates an HTML-formatted string from the markup entities found in the message’s caption.

Use this if you want to retrieve the message caption with the caption entities formatted as HTML in the same way the original message was formatted.

**Returns**  
Message caption with caption entities formatted as HTML.

**Return type**  
str

**caption_html_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.

**Returns**  
Message caption with caption entities formatted as HTML.

**Return type**  
str

**caption_markdown**
Creates an Markdown-formatted string from the markup entities found in the message’s caption 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

**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 is a legacy mode, retained by Telegram for backward compatibility. You should use *caption_markdown_v2_urled()* instead.

**Returns**  
Message caption with caption entities formatted as Markdown.

**Return type**  
str
caption_markdown_v2

Creates an 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.

**Returns**  
Message caption with caption entities formatted as Markdown.

**Return type**  
`str`

caption_markdown_v2_url

Creates an 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.

**Returns**  
Message caption with caption entities formatted as Markdown.

**Return type**  
`str`

chat_id

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

**Type**  
`int`

copy(
    chat_id: int,
    *args,
    **kwargs)

→  
`MessageId`

Shortcut for:

```python
bot.copy_message(chat_id=chat_id,  
                 from_chat_id=update.message.chat_id,  
                 message_id=update.message.message_id,  
                 *args,  
                 **kwargs)
```

**Returns**  
On success, returns the MessageId of the sent message.

**Return type**  
`telegram.MessageId`

delete(*args,**kwargs)

→  
`bool`

Shortcut for:

```python
bot.delete_message(chat_id=message.chat_id,  
                   message_id=message.message_id,  
                   *args,  
                   **kwargs)
```

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

**Return type**  
`bool`

edit_caption(*args,**kwargs)

→ Union[telegram.message.Message, bool]

Shortcut for:

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

**Note:** You can only edit messages that the bot sent itself (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** (*args, **kwargs*) → Union[telegram.message.Message, bool]

Shortcut for:

```python
bot.edit_message_live_location(chat_id=message.chat_id,
message_id=message.message_id,
*args,
**kwargs)
```

**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: InputMedia, *args, **kwargs) → Union[Message, bool]

Shortcut for:

```python
bot.edit_message_media(chat_id=message.chat_id,
message_id=message.message_id,
*args,
**kwargs)
```

**Note:** You can only edit messages that the bot sent itself (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** (*args, **kwargs) → Union[telegram.message.Message, bool]

Shortcut for:

```python
bot.edit_message_reply_markup(chat_id=message.chat_id,
message_id=message.message_id,
*args,
**kwargs)
```

**Note:** You can only edit messages that the bot sent itself (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`

---

3.2. `telegram` package 185
edit_text (*args, **kwargs) → Union[telegram.message.Message, bool]

Shortcut for:

```python
def bot.edit_message_text(chat_id=message.chat_id,
                        message_id=message.message_id,
                        *args,
                        **kwargs):
```

**Note:** You can only edit messages that the bot sent itself (i.e. of the `bot.send_*` family of methods) or channel posts, if the bot is an admin in that channel. However, this behavior 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`

effective_attachment
telegram.Audio or telegram.Contact or telegram.Document or telegram.Animation or telegram.Game or telegram.Invoice or telegram.Location or List[telegram.PhotoSize] or telegram.Sticker or telegram.SuccessfulPayment or telegram.Venue or telegram.Video or telegram.VideoNote or telegram.Voice: The attachment that this message was sent with. May be `None` if no attachment was sent.

forward (chat_id: int, *args, **kwargs) → telegram.message.Message

Shortcut for:

```python
def bot.forward_message(chat_id=chat_id,
                       from_chat_id=update.message.chat_id,
                       message_id=update.message.message_id,
                       *args,
                       **kwargs):
```

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

**Return type** `telegram.Message`

game_high_scores (*args, **kwargs) → List[GameHighScore]

Shortcut for:

```python
def bot.get_game_high_scores(chat_id=message.chat_id,
                            message_id=message.message_id,
                            *args,
                            **kwargs):
```

**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 behavior is undocumented and might be changed by Telegram.

**Returns** List[telegram.GameHighScore]

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
Returns a dict that maps telegram.MessageEntity to str. It contains entities from this
message’s caption filtered by their telegram.MessageEntity.type attribute as the key, and
the text that each entity belongs to as the value of the dict.

Note: This method should always be used instead of the caption_entities attribute,
since it calculates the correct substring from the message text based on UTF-16 codepoints. See
parse_entity for more info.

Parameters:

- types (List[str], optional) – List of telegram.MessageEntity types
  as strings. If the type attribute of an entity is contained in this list, it will be returned.
  Defaults to a list of all types. All types can be found as constants in telegram.
  MessageEntity.

Returns:
A dictionary of entities mapped to the text that belongs to them, calculated based
on UTF-16 codepoints.

Return type: Dict[telegram.MessageEntity, str]

parse_caption_entity
entity: telegram.MessageEntity) → str
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
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]

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(**

`*args, **kwargs`) → bool`  

Shortcut for:  

```python
bot.pin_chat_message(chat_id=message.chat_id,  
    message_id=message.message_id,  
    *args,  
    **kwargs)
```

**Returns**

On success, `True` is returned.

**Return type**

`bool`

---

**reply_animation(**

`*args, **kwargs`) → telegram.message.Message`  

Shortcut for:  

```python
bot.send_animation(update.message.chat_id,  
    *args,  
    **kwargs)
```

**Keyword Arguments**

- `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(**

`*args, **kwargs`) → telegram.message.Message`  

Shortcut for:  

```python
bot.send_audio(update.message.chat_id,  
    *args,  
    **kwargs)
```

**Keyword Arguments**

- `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_contact(**

`*args, **kwargs`) → telegram.message.Message`  

Shortcut for:  

```python
bot.send_contact(update.message.chat_id,  
    *args,  
    **kwargs)
```

**Keyword Arguments**

- `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: int, message_id: int, *args, **kwargs*) → MessageId

Shortcut for:

```python
bot.copy_message(chat_id=message.chat.id,
                 from_chat_id=from_chat_id,
                 message_id=message_id,
                 *args,
                 **kwargs)
```

Returns On success, returns the MessageId of the sent message.

Return type `telegram.MessageId`

`reply_dice` (*args, **kwargs*) → `telegram.message.Message`

Shortcut for:

```python
bot.send_dice(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments

- `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` (*args, **kwargs*) → `telegram.message.Message`

Shortcut for:

```python
bot.send_document(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments

- `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_html` (*args, **kwargs*) → `telegram.message.Message`

Shortcut for:

```python
bot.send_message(update.message.chat_id, parse_mode=ParseMode.HTML, *args,
                 **kwargs)
```

Sends a message with HTML formatting.

Keyword Arguments

- `quote` (bool, optional) – If set to True, the message is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.

Return type `telegram.Message`

`reply_location` (*args, **kwargs*) → `telegram.message.Message`

Shortcut for:
```python
bot.send_location(update.message.chat_id, *args, **kwargs)
```

**Keyword Arguments**

- `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

*Sends a message with Markdown version 1 formatting.*

```python
reply_markdown(*args, **kwargs) → telegram.message.Message
```

**Shortcut for:**

```python
bot.send_message(update.message.chat_id, parse_mode=ParseMode.MARKDOWN, _,
                  *args,                  ...
                  **kwargs)
```

**Keyword Arguments**

- `quote` *(bool, optional)*: If set to True, the message is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

**Returns**

On success, instance representing the message posted.

**Return type** `telegram.Message`

### reply_markdown_v2

*Sends a message with markdown version 2 formatting.*

```python
reply_markdown_v2(*args, **kwargs) → telegram.message.Message
```

**Shortcut for:**

```python
bot.send_message(update.message.chat_id, parse_mode=ParseMode.MARKDOWN_V2, _
                  *args,                  ...
                  **kwargs)
```

**Keyword Arguments**

- `quote` *(bool, optional)*: If set to True, the message is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

**Returns**

On success, instance representing the message posted.

**Return type** `telegram.Message`

### reply_media_group

*Sends a message with a Media Group.*

```python
reply_media_group(*args, **kwargs) → List[Optional[telegram.message.Message]]
```

**Shortcut for:**

```python
bot.send_media_group(update.message.chat_id, *args, **kwargs)
```

**Keyword Arguments**

- `quote` *(bool, optional)*: If set to True, the media group is sent as an actual reply to this message. If `reply_to_message_id` is passed in `kwargs`, this parameter will be ignored. Default: True in group chats and False in private chats.

**Returns**

An array of the sent Messages.

**Return type** `List[telegram.Message]`

**Raises** `telegram.TelegramError`
reply_photo(*args, **kwargs) → telegram.message.Message
Shortcut for:

```python
bot.send_photo(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments quote (bool, optional) – If set to True, the photo is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.
Return type telegram.Message

reply_poll(*args, **kwargs) → telegram.message.Message
Shortcut for:

```python
bot.send_poll(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments 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

reply_sticker(*args, **kwargs) → telegram.message.Message
Shortcut for:

```python
bot.send_sticker(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments 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

reply_text(*args, **kwargs) → telegram.message.Message
Shortcut for:

```python
bot.send_message(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments quote (bool, optional) – If set to True, the message is sent as an actual reply to this message. If reply_to_message_id is passed in kwargs, this parameter will be ignored. Default: True in group chats and False in private chats.

Returns On success, instance representing the message posted.
Return type telegram.Message

reply_venue(*args, **kwargs) → telegram.message.Message
Shortcut for:

```python
bot.send_venue(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments 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(*args, **kwargs) → telegram.message.Message
Shortcut for:

```
bot.send_video(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments 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(*args, **kwargs) → telegram.message.Message
Shortcut for:

```
bot.send_video_note(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments 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(*args, **kwargs) → telegram.message.Message
Shortcut for:

```
bot.send_voice(update.message.chat_id, *args, **kwargs)
```

Keyword Arguments 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(*args, **kwargs) → Union[telegram.message.Message, bool]
Shortcut for:

```
bot.set_game_score(chat_id=message.chat_id, message_id=message.message_id, *args, **kwargs)
```

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(*args, **kwargs) → Union[telegram.message.Message, bool]

Shortcut for:

```python
bot.stop_message_live_location(chat_id=message.chat_id,
    message_id=message.message_id,
    *args,
    **kwargs)
```

**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(*args, **kwargs) → telegram.poll.Poll

Shortcut for:

```python
bot.stop_poll(chat_id=message.chat_id,
    message_id=message.message_id,
    *args,
    **kwargs)
```

**Returns** On success, the stopped Poll with the final results is returned.

**Return type** `telegram.Poll`

text_html

Creating an HTML-formatted string from the markup entities found in the message.

Use this if you want to retrieve the message text with the entities formatted as HTML in the same way the original message was formatted.

**Returns** Message text with entities formatted as HTML.

**Return type** `str`

text_html_urlred

Creating an HTML-formatted string from the markup entities found in the message.

Use this if you want to retrieve the message text with the entities formatted as HTML. This also formats `telegram.MessageEntity.URL` as a hyperlink.

**Returns** Message text with entities formatted as HTML.

**Return type** `str`

text_markdown

Creating 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

**text_markdown_url**

Creates an 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_url` instead.

**Returns**  
Message text with entities formatted as Markdown.

**Return type**  
str

**text_markdown_v2**

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 in the same way the original message was formatted.

**Returns**  
Message text with entities formatted as Markdown.

**Return type**  
str

**text_markdown_v2_url**

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.

**Returns**  
Message text with entities formatted as Markdown.

**Return type**  
str

**unpin**

Shortcut for:

```python
bot.unpin_chat_message(chat_id=message.chat_id,  
message_id=message.message_id,  
*args,  
**kwargs)
```

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

**Return type**  
bool

---

### 3.2.33 telegram.MessageId

**class**  
`telegram.MessageId(message_id: int, **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
3.2.34 telegram.MessageEntity

class telegram.MessageEntity(type: str, offset: int, length: int, url: str = None, user: telegram.User = None, language: str = 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.

**type**
Type of the entity.

**offset**
Offset in UTF-16 code units to the start of the entity.

**length**
Length of the entity in UTF-16 code units.

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

**user**
Optional. The mentioned user.

**language**
Optional. Programming language of the entity text.

**Parameters**
- **type** (str) – Type of the entity. Can be mention (@username), hashtag, bot_command, url, email, phone_number, bold (bold text), italic (italic text), strikethrough, 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.

**ALL_TYPES** = ['mention', 'hashtag', 'cashtag', 'phone_number', 'bot_command', 'url', telegram.constants.MESSAGEENTITY_ALL_TYPES]

List of all the types
BOLD = 'bold'
    telegram.constants.MESSAGEENTITY_BOLD

BOT_COMMAND = 'bot_command'
    telegram.constants.MESSAGEENTITY_BOT_COMMAND

CASHTAG = 'cashtag'
    telegram.constants.MESSAGEENTITY_CASHTAG

CODE = 'code'
    telegram.constants.MESSAGEENTITY_CODE

EMAIL = 'email'
    telegram.constants.MESSAGEENTITY_EMAIL

HASHTAG = 'hash_tag'
    telegram.constants.MESSAGEENTITY_HASHTAG

ITALIC = 'italic'
    telegram.constants.MESSAGEENTITY_ITALIC

MENTION = 'mention'
    telegram.constants.MESSAGEENTITY_MENTION

PHONE_NUMBER = 'phone_number'
    telegram.constants.MESSAGEENTITY_PHONE_NUMBER

PRE = 'pre'
    telegram.constants.MESSAGEENTITY_PRE

STRIKETHROUGH = 'strikethrough'
    telegram.constants.MESSAGEENTITY_STRIKETHROUGH

TEXT_LINK = 'text_link'
    telegram.constants.MESSAGEENTITY_TEXT_LINK

TEXT_MENTION = 'text_mention'
    telegram.constants.MESSAGEENTITY_TEXT_MENTION

UNDERLINE = 'underline'
    telegram.constants.MESSAGEENTITY_UNDERLINE

URL = 'url'
    telegram.constants.MESSAGEENTITY_URL

### 3.2.35 `telegram.ParseMode`

class `telegram.ParseMode`

    Bases: object

    This object represents a Telegram Message Parse Modes.

    HTML = 'HTML'
        telegram.constants.PARSEMODE_HTML

    MARKDOWN = '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 = 'MarkdownV2'
        telegram.constants.PARSEMODE_MARKDOWN_V2
3.2.36 telegram.PhotoSize

class telegram.PhotoSize(file_id: str, file_unique_id: str, width: int, height: int, file_size: int = None, bot: 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.

**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

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.

get_file(timeout: int = None, api_kwargs: Dict[str, Any] = None) → File
    Convenience wrapper over telegram.Bot.get_file

Parameters

- **timeout**(int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
• `api_kwargs` (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns `telegram.File`

Raises `telegram.TelegramError`

### 3.2.37 `telegram.Poll`

```python
class telegram.Poll(id: str, question: str, options: List[telegram.poll.PollOption],
                  total_voter_count: int, is_closed: bool, is_anonymous: bool,
                  type: str, allows_multiple_answers: bool, correct_option_id: int = None,
                  explanation: str = None, explanation_entities: List[telegram.messageentity.MessageEntity] = None,
                  open_period: int = None, close_date: datetime.datetime = 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 = 100
telegram.constants.MAX_POLL_OPTION_LENGTH

MAX_QUESTION_LENGTH = 300
telegram.constants.MAX_POLL_QUESTION_LENGTH

QUIZ = 'quiz'
telegram.constants.POLL_QUIZ

REGULAR = 'regular'
telegram.constants.POLL_REGULAR
parse_explanation_entities(types: List[str] = None) → Dict[telegram.MessageEntity, str]

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: telegram.MessageEntity) → str

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.

3.2.38 telegram.PollAnswer

class telegram.PollAnswer(poll_id: str, user: telegram.user.User, option_ids: List[int], **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 option_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.

3.2.39 `telegram.PollOption`

class `telegram.PollOption (text: str, voter_count: int, **_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.

- `text`  
  Option text, 1-100 characters.  
  Type `str`

- `voter_count`  
  Number of users that voted for this option.  
  Type `int`

Parameters

- `text (str)` – Option text, 1-100 characters.
- `voter_count (int)` – Number of users that voted for this option.

`MAX_LENGTH = 100`  
`telegram.constants.MAX_POLL_OPTION_LENGTH`

3.2.40 `telegram.ProximityAlertTriggered`

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.

- `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`

Parameters
• **traveler** (*telegram.User*) – User that triggered the alert
• **watcher** (*telegram.User*) – User that set the alert
• **distance** (*int*) – The distance between the users

### 3.2.41 telegram.ReplyKeyboardRemove

#### class telegram.ReplyKeyboardRemove

**Bases:** telegram.replymarkup.ReplyMarkup

Upon receiving a message with this object, Telegram clients will remove the current custom keyboard and display the default letter-keyboard. By default, custom keyboards are displayed until a new keyboard is sent by a bot. An exception is made for one-time keyboards that are hidden immediately after the user presses a button (see `telegram.ReplyKeyboardMarkup`).

**remove_keyboard**

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

**Example**

A user votes in a poll, bot returns confirmation message in reply to the vote and removes the keyboard for that user, while still showing the keyboard with poll options to users who haven’t voted yet.

**Parameters**

- **selective** (*bool*, optional) – Use this parameter if you want to remove the keyboard for specific users only. Targets:
  1) Users that are @mentioned in the text of the `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.

**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`.

### 3.2.42 telegram.ReplyKeyboardMarkup

#### class telegram.ReplyKeyboardMarkup

**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 the size of `keyboard` and all the buttons are equal.

**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

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

classmethod from_button (button: Union[telegram.keyboardbutton.KeyboardButton, str], resize_keyboard: bool = False, one_time_keyboard: bool = False, selective: bool = False, **kwargs) → telegram.replykeyboardmarkup.ReplyKeyboardMarkup

Shortcut for:

```python
ReplyKeyboardMarkup([[button]], **kwargs)
```

Return a ReplyKeyboardMarkup from a single KeyboardButton.

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`.

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

```python
```

Shortcut for:

```
ReplyKeyboardMarkup([[button] for button in button_column], **kwargs)
```

Return a ReplyKeyboardMarkup from a single column of KeyboardButtons.

Parameters

• **button_column** *(List[telegram.KeyboardButton | str])* – The button to use in the markup.

• **resize_keyboard** *(bool, optional)* – Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to `False`, in which case the custom keyboard is always of the same height as the app’s standard keyboard.

• **one_time_keyboard** *(bool, optional)* – Requests clients to hide the keyboard as soon as it’s been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat - the user can press a special button in the input field to see the custom keyboard again. Defaults to `False`.

• **selective** *(bool, optional)* – Use this parameter if you want to show the keyboard to specific users only. Targets:

  1) Users that are @mentioned in the text of the Message object.

  2) If the bot’s message is a reply (has `reply_to_message_id`), sender of the original message.

  Defaults to `False`.

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

```python
```

Shortcut for:
ReplyKeyboardMarkup([button_row], **kwargs)

Return a ReplyKeyboardMarkup from a single row 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.

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

### 3.2.43 `telegram.ReplyMarkup`

**class** `telegram.ReplyMarkup`

Bases: `telegram.base.TelegramObject`

Base class for Telegram ReplyMarkup Objects.


### 3.2.44 `telegram.TelegramObject`

**class** `telegram.TelegramObject`

Bases: `object`

Base class for most telegram objects.

**to_json** () → str

    Returns str
class telegram.Update(
    update_id: int = None,
    message: telegram.message.Message = None,
    edited_message: telegram.message.Message = None,
    channel_post: telegram.message.Message = None,
    edited_channel_post: telegram.message.Message = None,
    inline_query: telegram.inline.inlinequery.InlineQuery = None,
    chosen_inline_result: telegram.choseninlineresult.ChosenInlineResult = None,
    callback_query: telegram.callbackquery.CallbackQuery = None,
    shipping_query: telegram.payment.shippingquery.ShippingQuery = None,
    pre_checkout_query: telegram.payment.precheckoutquery.PreCheckoutQuery = None,
    poll: telegram.poll.Poll = None,
    poll_answer: telegram.poll.PollAnswer = 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.

**update_id**

The update’s unique identifier.

**message**

Optional. New incoming message.

**edited_message**

Optional. New version of a message.

**channel_post**

Optional. New incoming channel post.

**edited_channel_post**

Optional. New version of a channel post.

**inline_query**

Optional. New incoming inline query.

**chosen_inline_result**

Optional. The result of an inline query that was chosen by a user.

**callback_query**

Optional. New incoming callback query.

**shipping_query**

Optional. New incoming shipping query.
**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*

**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.


- **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.

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

**classmethod de_json**(data: Optional[Dict[str, Any]], bot: Bot) → Optional[Update]

**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`

**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` and `poll_answer`.

**Type** `telegram.Message`

**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`

3.2.46 `telegram.User`

```python
class telegram.User(id: int, first_name: str, is_bot: bool, last_name: str = None, username: str = None, language_code: str = None, can_join_groups: bool = None, can_read_all_group_messages: bool = None, supports_inline_queries: bool = None, bot: 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.

**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 \texttt{str}

\texttt{supports_inline_queries}
Optional. True, if the bot supports inline queries. Returned only in \texttt{telegram.Bot.get_me}
requests.

Type \texttt{str}

\texttt{bot}
Optional. The Bot to use for instance methods.

Type \texttt{telegram.Bot}

Parameters

- \texttt{id} (int) – Unique identifier for this user or bot.
- \texttt{is_bot} (bool) – True, if this user is a bot.
- \texttt{first_name} (str) – User’s or bot’s first name.
- \texttt{last_name} (str, optional) – User’s or bot’s last name.
- \texttt{username} (str, optional) – User’s or bot’s username.
- \texttt{language_code} (str, optional) – IETF language tag of the user’s language.
- \texttt{can_join_groups} (str, optional) – True, if the bot can be invited to groups. Returned only in \texttt{telegram.Bot.get_me}
requests.
- \texttt{can_read_all_group_messages} (str, optional) – True, if privacy mode is disabled for the bot. Returned only in \texttt{telegram.Bot.get_me}
requests.
- \texttt{supports_inline_queries} (str, optional) – True, if the bot supports inline queries. Returned only in \texttt{telegram.Bot.get_me}
requests.
- \texttt{bot} (\texttt{telegram.Bot}, optional) – The Bot to use for instance methods.

\texttt{copy_message(*args, **kwargs)} → MessageId

Shortcut for:

\begin{verbatim}
bot.copy_message(from_chat_id=update.effective_user.id, *args, **kwargs)
\end{verbatim}

Returns
On success, instance representing the message posted.

Return type \texttt{telegram.Message}

\texttt{full_name}
Convenience property. The user’s \texttt{first_name}, followed by (if available) \texttt{last_name}.

Type \texttt{str}

\texttt{get_profile_photos(*args, **kwargs)} → UserProfilePhotos

Shortcut for:

\begin{verbatim}
bot.get_user_profile_photos(update.effective_user.id, *args, **kwargs)
\end{verbatim}

\texttt{link}
Convenience property. If \texttt{username} is available, returns a \texttt{t.me} link of the user.

Type \texttt{str}

\texttt{mention_html(name: str = None)} → str

Parameters \texttt{name} (str) – The name used as a link for the user. Defaults to \texttt{full_name}.

Returns The inline mention for the user as HTML.

Return type \texttt{str}
mention_markdown (name: str = None) → str

Note: telegram.ParseMode.MARKDOWN is a legacy mode, retained by Telegram for backward compatibility. You should use mention_markdown_v2() instead.

Parameters name (str) – The name used as a link for the user. Defaults to full_name.

Returns The inline mention for the user as markdown (version 1).

Return type str

mention_markdown_v2 (name: str = None) → str

Parameters name (str) – The name used as a link for the user. Defaults to full_name.

Returns The inline mention for the user as markdown (version 2).

Return type str

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

Type str

pin_message (*args, **kwargs) → bool

Shortcut for:

```python
bot.pin_chat_message(chat_id=update.effective_user.id, *args, **kwargs)
```

Returns On success, True is returned.

Return type bool

send_action (*args, **kwargs) → bool

Alias for send_chat_action

send_animation (*args, **kwargs) → Message

Shortcut for:

```python
bot.send_animation(update.effective_user.id, *args, **kwargs)
```

Returns On success, instance representing the message posted.

Return type telegram.Message

send_audio (*args, **kwargs) → Message

Shortcut for:

```python
bot.send_audio(update.effective_user.id, *args, **kwargs)
```

Returns On success, instance representing the message posted.

Return type telegram.Message

send_chat_action (*args, **kwargs) → bool

Shortcut for:
bot.send_chat_action(update.effective_user.id, *args, **kwargs)

**Returns**  On success.

**Return type**  True

**send_contact** (*args, **kwargs) → Message
Shortcut for:

bot.send_contact(update.effective_user.id, *args, **kwargs)

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

**Return type**  telegram.Message

**send_copy** (*args, **kwargs) → MessageId
Shortcut for:

bot.copy_message(chat_id=update.effective_user.id, *args, **kwargs)

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

**Return type**  telegram.Message

**send_dice** (*args, **kwargs) → Message
Shortcut for:

bot.send_dice(update.effective_user.id, *args, **kwargs)

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

**Return type**  telegram.Message

**send_document** (*args, **kwargs) → Message
Shortcut for:

bot.send_document(update.effective_user.id, *args, **kwargs)

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

**Return type**  telegram.Message

**send_game** (*args, **kwargs) → Message
Shortcut for:

bot.send_game(update.effective_user.id, *args, **kwargs)

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

**Return type**  telegram.Message

**send_invoice** (*args, **kwargs) → Message
Shortcut for:

bot.send_invoice(update.effective_user.id, *args, **kwargs)

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

**Return type**  telegram.Message

3.2. **telegram package** 211
send_location(*args, **kwargs) → Message
Shortcut for:

```
bot.send_location(update.effective_user.id, *args, **kwargs)
```

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

**Return type** `telegram.Message`

send_media_group(*args, **kwargs) → List[Message]
Shortcut for:

```
bot.send_media_group(update.effective_user.id, *args, **kwargs)
```

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

**Return type** List[`telegram.Message`]

send_message(*args, **kwargs) → Message
Shortcut for:

```
bot.send_message(update.effective_user.id, *args, **kwargs)
```

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

**Return type** `telegram.Message`

send_photo(*args, **kwargs) → Message
Shortcut for:

```
bot.send_photo(update.effective_user.id, *args, **kwargs)
```

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

**Return type** `telegram.Message`

send_poll(*args, **kwargs) → Message
Shortcut for:

```
bot.send_poll(update.effective_user.id, *args, **kwargs)
```

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

**Return type** `telegram.Message`

send_sticker(*args, **kwargs) → Message
Shortcut for:

```
bot.send_sticker(update.effective_user.id, *args, **kwargs)
```

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

**Return type** `telegram.Message`

send_venue(*args, **kwargs) → Message
Shortcut for:

```
bot.send_venue(update.effective_user.id, *args, **kwargs)
```

**Returns** On success, instance representing the message posted.
**Return type**  `telegram.Message`

`send_video` (*args, **kwargs) → Message

Shortcut for:

```python
bot.send_video(update.effective_user.id, *args, **kwargs)
```

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

**Return type**  `telegram.Message`

`send_video_note` (*args, **kwargs) → Message

Shortcut for:

```python
bot.send_video_note(update.effective_user.id, *args, **kwargs)
```

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

**Return type**  `telegram.Message`

`send_voice` (*args, **kwargs) → Message

Shortcut for:

```python
bot.send_voice(update.effective_user.id, *args, **kwargs)
```

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

**Return type**  `telegram.Message`

`unpin_all_messages` (*args, **kwargs) → bool

Shortcut for:

```python
bot.unpin_all_chat_messages(chat_id=update.effective_user.id, *args, **kwargs)
```

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

**Return type**  `bool`

`unpin_message` (*args, **kwargs) → bool

Shortcut for:

```python
bot.unpin_chat_message(chat_id=update.effective_user.id, *args, **kwargs)
```

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

**Return type**  `bool`

### 3.2.47 `telegram.UserProfilePhotos`

**class**  `telegram.UserProfilePhotos` (total_count: `int`, photos: `List[List[telegram.files.photosize.PhotoSize]]`, **kwargs)

**Bases:** `telegram.base.TelegramObject`

This object represent 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.

**total_count**
Total number of profile pictures.

  Type  int

**photos**
Requested profile pictures.

  Type  List[List[telegram.PhotoSize]]

**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).

### 3.2.48 `telegram.Venue`

class `telegram.Venue`

```python
class telegram.Venue(
    location: telegram.files.location.Location,  
    title: str,  
    address: str,  
    foursquare_id: str = None,  
    foursquare_type: str = None,  
    google_place_id: str = None,  
    google_place_type: str = None,  
    **_kwargs)
```

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.

**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

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.

3.2.49 telegram.Video

class telegram.Video(file_id: str, file_unique_id: str, width: int, height: int, duration: int, thumb: telegram.files.photosize.PhotoSize = None, mime_type: str = None, file_size: int = None, bot: Bot = None, file_name: str = 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.

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

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.

get_file(timeout: int = None, api_kwargs: Dict[str, Any] = None) -> File

Convenience wrapper over telegram.Bot.get_file

Parameters

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

Returns telegram.File

Raises telegram.TelegramError

3.2.50 telegram.VideoNote

class telegram.VideoNote (file_id: str, file_unique_id: str, length: int, duration: int, thumb: telegram.files.photosize.PhotoSize = None, file_size: int = None, bot: 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.

**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`

**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.

**get_file** (`timeout: int = None, api_kwargs: Dict[str, Any] = None`) → `File`

Convenience wrapper over `telegram.Bot.get_file`

**Parameters**

- `timeout` (`int | float, optional`) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
• `api_kwargs` (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns** `telegram.File`

**Raises** `telegram.TelegramError`

### 3.2.51 `telegram.Voice`

```python
class telegram.Voice(file_id: str, file_unique_id: str, duration: int, mime_type: str = None, file_size: int = None, bot: 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.

**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`

**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.

**get_file** (`timeout: int = None, api_kwargs: Dict[str, Any] = None`) → `File`

Convenience wrapper over `telegram.Bot.get_file`
Parameters

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

- **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

**Returns** telegram.File

**Raises** telegram.TelegramError

### 3.2.52 telegram.WebhookInfo

class telegram.WebhookInfo (url: str, has_custom_certificate: bool, pending_update_count: int, last_error_date: int = None, last_error_message: str = None, max_connections: int = None, allowed_updates: List[str] = None, ip_address: str = None, **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.

**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.

Type List[str]
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.

### 3.2.53 Stickers

telegram.Sticker
class telegram.Sticker (file_id: str, file_unique_id: str, width: int, height: int, is_animated: bool, thumb: telegram.files.photosize.PhotoSize = None, emoji: str = None, file_size: int = None, set_name: str = None, mask_position: MaskPosition = None, bot: 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.

**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

**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

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.
• 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.

get_file (timeout: str = None, api_kwargs: Dict[str, Any] = None) → File
Convenience wrapper over telegram.Bot.get_file

Parameters

• timeout (int | float, optional) – If this value is specified, use it as the read timeout from the server (instead of the one specified during creation of the connection pool).
• api_kwargs (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns telegram.File

Raises telegram.TelegramError
**telegram.StickerSet**

```python
class telegram.StickerSet(name: str, title: str, is_animated: bool, contains_masks: bool, stickers: List[telegram.files.sticker.Sticker], thumb: telegram.files.photosize.PhotoSize = 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.

- **name**
  - Sticker set name.
  - Type `str`

- **title**
  - Sticker set title.
  - Type `str`

- **is_animated**
  - True, if the sticker set contains animated stickers.
  - 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 or .TGS format.
  - Type `telegram.PhotoSize`

**Parameters**

- `name (str)` – Sticker set name.
- `title (str)` – Sticker set title.
- `is_animated (bool)` – True, if the sticker set contains animated stickers.
- `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 or .TGS format.

**telegram.MaskPosition**

```python
class telegram.MaskPosition(point: str, x_shift: float, y_shift: float, scale: float, **kwargs)
```

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 = 'chin'
telegram.constants.STICKER_CHIN

EYES = 'eyes'
telegram.constants.STICKER_EYES

FOREHEAD = 'forehead'
telegram.constants.STICKER_FOREHEAD

MOUTH = 'mouth'
telegram.constants.STICKER_MOUTH

### 3.2.54 Inline Mode

telegram.InlineQuery

**class telegram.InlineQuery**

```python
class telegram.InlineQuery(id: str, from_user: telegram.user.User, query: str, offset: str, location: telegram.files.location.Location = None, bot: Bot = None, **kwargs)
```

Bases: telegram.baseTelegramObject

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.
- `location` (telegram.Location, optional) – Sender location, only for bots that request user location.
- `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.
- `bot` (telegram.Bot, optional) – The Bot to use for instance methods.
- **kwargs (dict) – Arbitrary keyword arguments.

### answer (*args, auto_pagination: bool = False, **kwargs) → bool
Shortcut for:

```python
bot.answer_inline_query(update.inline_query.id, *args,
    current_offset=self.offset if auto_pagination else None,
    **kwargs)
```

### Parameters
- `results` (List[telegram.InlineQueryResult] | Callable) – A list of results for the inline query. In case `auto_pagination` is set to `True`, `results` may also be a callable may also be a callable accepts the current page index starting from 0. It must return either a list of `telegram.InlineResult` 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.

• **auto_pagination** (bool, optional) – If set to True, offset will be passed as current_offset to :meth:`telegram.Bot.answer_inline_query`. Defaults to False.

### telegram.InlineQueryResult

**class** telegram.InlineQueryResult (**type**: str, **id**: str, **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.

**type**

Type of the result.

**Type** str

**id**

Unique identifier for this result, 1-64 Bytes.

**Type** str

**Parameters**

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

### telegram.InlineQueryResultArticle

**class** telegram.InlineQueryResultArticle (**id**: str, **title**: str, **input_message_content**: InputMessageContent, **reply_markup**: ReplyMarkup = None, **url**: str = None, **hide_url**: bool = None, **description**: str = None, **thumb_url**: str = None, **thumb_width**: int = None, **thumb_height**: int = None, **kwargs**)

**Bases:** telegram.inline.inlinequeryresult.InlineQueryResult

This object represents a Telegram InlineQueryResultArticle.

**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`  

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.

telemtelegram.InlineQueryResultAudio


Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to an mp3 audio file. By default, this audio file will be sent by the user. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the audio.

**type**

`'audio'`

**id**

Unique identifier for this result, 1-64 bytes.

**audio_url**

A valid URL for the audio file.

**title**

Title.

**performer**

Optional. Performer.

**audio_duration**

Optional. Audio duration in seconds.

**caption**

Optional. Caption, 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.
```python
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

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.
```

```python
class telegram.InlineQueryResultCachedAudio


Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to an mp3 audio file stored on the Telegram servers. By default, this audio file will be sent by the user. Alternatively, you can use input_message_content to send a message with the specified content instead of the audio.

type
    ‘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

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.

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to a file stored on the Telegram servers. By default, this file will be sent by the user with an optional caption. Alternatively, you can use input_message_content to send a message with the specified content instead of the file.

type
   ‘document’.

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

title
   Title for the result.

document_file_id
   A valid file identifier for the file.

description
   Optional. Short description of the result.

caption
   Optional. Caption of the document 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.

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.
**telegram.InlineKeyboardMarkup**

Optional. Content of the message to be sent instead of the file.

**Type** `telegram.InputMessageContent`

### 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.

### telegram.InlineQueryResultCachedGif

**class** `telegram.InlineQueryResultCachedGif`

```python
```

**Bases:** `telegram.inline.inlinequeryresult.InlineQueryResult`

Represents a link to an animated GIF file stored on the Telegram servers. By default, this animated GIF file will be sent by the user with an optional caption. Alternatively, you can use `input_message_content` to send a message with specified content instead of the animation.

**type**

"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.

### 3.2. telegram package

231
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`

**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 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.
class telegram.InlineQueryResultCachedMpeg4Gif

Represented link to a video animation (H.264/MPEG-4 AVC video without sound) stored on the Telegram servers. By default, this animated MPEG-4 file will be sent by the user with an optional caption. Alternatively, you can use input_message_content to send a message with the specified content instead of the animation.

type
    'mpeg4_gif'.

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

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.

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 MPEG-4 file.
Type \texttt{telegram.InputMessageContent}

Parameters

\begin{itemize}
\item \texttt{id (str)} – Unique identifier for this result, 1-64 bytes.
\item \texttt{mpeg4\_file\_id (str)} – A valid file identifier for the MP4 file.
\item \texttt{title (str, optional)} – Title for the result.
\item \texttt{caption (str, optional)} – Caption of the MPEG-4 file to be sent, 0-1024 characters after entities parsing.
\item \texttt{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 \texttt{telegram.ParseMode} for the available modes.
\item \texttt{caption\_entities} (List[\texttt{telegram.MessageEntity}], optional) – List of special entities that appear in the caption, which can be specified instead of \texttt{parse\_mode}.
\item \texttt{reply\_markup} (\texttt{telegram.InlineKeyboardMarkup}, optional) – Inline keyboard attached to the message.
\item \texttt{input\_message\_content} (\texttt{telegram.InputMessageContent}, optional) – Content of the message to be sent instead of the MPEG-4 file.
\item \texttt{**kwargs (dict)} – Arbitrary keyword arguments.
\end{itemize}

\texttt{telegram.InlineQueryResultCachedPhoto}

\begin{verbatim}

Bases: telegram.inline.inlinequeryresult.InlineQueryResult
\end{verbatim}

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 \texttt{input\_message\_content} to send a message with the specified content instead of the photo.

\begin{verbatim}
type
    'photo'.
    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.
\end{verbatim}
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`

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.
**telegram.InlineQueryResultCachedSticker**

class telegram.InlineQueryResultCachedSticker(id: str, sticker_file_id: str, reply_markup: ReplyMarkup = None, input_message_content: InputMessageContent = None, **kwargs)

Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to a sticker stored on the Telegram servers. By default, this sticker will be sent by the user. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the sticker.

**type**

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

**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.

**telegram.InlineQueryResultCachedVideo**


Bases: telegram.inline.inlinequeryresult.InlineQueryResult
Represents a link to a video file stored on the Telegram servers. By default, this video file will be sent by the user with an optional caption. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the video.

**type**

    'video'.

**id**

    Unique identifier for this result, 1-64 bytes.

**video_file_id**

    A valid file identifier for the video file.

**title**

    Title for the result.

**description**

    Optional. Short description of the result.

**caption**

    Optional. Caption of the video 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.

**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 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.

### telegram.InlineQueryResultCachedVoice

**class** `telegram.InlineQueryResultCachedVoice`

```python
```

**Bases:** `telegram.inline.inlinequeryresult.InlineQueryResult`

Represents a link to a voice message stored on the Telegram servers. By default, this voice message will be sent by the user. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the voice message.

**type**

Type `str`

**id**

Unique identifier for this result, 1-64 bytes.

Type `str`

**voice_file_id**

A valid file identifier for the voice message.

Type `str`

**title**

Voice message 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]`

**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 message.

Type `telegram.InputMessageContent`

### 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.

---

**telegram.InlineQueryResultContact**

### class `telegram.InlineQueryResultContact`  
(id: `str`,  
phone_number: `str`,  
first_name: `str`,  
last_name: `str = None`,  
reply_markup: `ReplyMarkup = None`,  
input_message_content: `InputMessageContent = None`,  
thumb_url: `str = None`,  
thumb_width: `int = None`,  
thumb_height: `int = None`,  
vcard: `str = None`,  
**kwargs)

Bases: `telegram.inline.inlinequeryresult.InlineQueryResult`

Represents a contact with a phone number. By default, this contact will be sent by the user. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the contact.

**type**

'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

thumb_height
  Optional. Thumbnail height.
  Type  int

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.
**telegram.InlineQueryResultDocument**

```python
```

**Bases:** `telegram.inline.inlinequeryresult.InlineQueryResult`

Represents a link to a file. By default, this file will be sent by the user with an optional caption. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the file. Currently, only .PDF and .ZIP files can be sent using this method.

- **type**
  - `str`

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

- **title**
  - Title for the result.

- **caption**
  - Optional. Caption of the document 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.

- **caption_entities**
  - Optional. List of special entities that appear in the caption, which can be specified instead of `parse_mode`.

- **document_url**
  - A valid URL for the file.

- **mime_type**
  - Mime type of the content of the file, either “application/pdf” or “application/zip”.

- **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 file.  
Type `telegram.InputMessageContent`  

thumb_url  
Optional. URL of the thumbnail (jpeg only) for the file.  
Type `str`  

thumb_width  
Optional. Thumbnail width.  
Type `int`  

thumb_height  
Optional. Thumbnail height.  
Type `int`  

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.

telemeg.InlineQueryResultGame  

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

**type**

'type' game'.

**id**

Unique identifier for this result, 1-64 bytes.

**game_short_name**

Short name of the game.

**reply_markup**

Optional. Inline keyboard attached to the message.

**Parameters**

- id (str) – Unique identifier for this result, 1-64 bytes.
- game_short_name (str) – Short name of the game.
- reply_markup (telegram.InlineKeyboardMarkup, optional) – Inline keyboard attached to the message.
- **kwargs (dict) – Arbitrary keyword arguments.

### telegram.InlineQueryResultGif

**class** `telegram.InlineQueryResultGif`

```
```

**Bases:** `telegram.inline.inlinequeryresult.InlineQueryResult`

Represents a link to an animated GIF file. By default, this animated GIF file will be sent by the user with optional caption. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the animation.

**type**

'gif'.

**id**

Unique identifier for this result, 1-64 bytes.

**gif_url**

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

**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.

```
class telegram.InlineQueryResultLocation

id: str
latitude: float
longitude: float
title: str
live_period: int = None
reply_markup: ReplyMarkup = None
input_message_content: InputMessageContent = None
thumb_url: str = None
thumb_width: int = None
thumb_height: int = None
horizontal_accuracy: float = None
heading: int = None
proximity_alert_radius: int = None

Bases: telegram.inline.inlinequeryresult.InlineQueryResult
```

Represents a location on a map. By default, the location will be sent by the user. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the location.

```
type
'type': str
id
Unique identifier for this result, 1-64 bytes.

latitude
Location latitude in degrees.

longitude
Location longitude in degrees.

title
Location title.
```

3.2. telegram package 245
**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

**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.

telemo.InlineQueryResultMpeg4Gif


Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to a video animation (H.264/MPEG-4 AVC video without sound). By default, this animated MPEG-4 file will be sent by the user with optional caption. Alternatively, you can use *input_message_content* to send a message with the specified content instead of the animation.

type
  'mpeg4_gif'.

  Type str

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`

**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.

telegram.InlineQueryResultPhoto


Bases: telegram.inline.inlinequeryresult.InlineQueryResult

Represents a link to a photo. By default, this photo will be sent by the user with optional caption. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the photo.

type
  `'photo'`.

  Type *str*

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`

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

```python
class telegram.InlineQueryResultVenue(id: str, latitude: float, longitude: float, title: str, address: str, foursquare_id: str = None, foursquare_type: str = None, reply_markup: ReplyMarkup = None, input_message_content: InputMessageContent = None, thumb_url: str = None, thumb_width: int = None, thumb_height: int = None, google_place_id: str = None, google_place_type: str = 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 Place details are mutually exclusive. However, this behaviour is undocumented and might be changed by Telegram.

**type**

- `str`

**id**

- `str`

**latitude**

- `str`

**longitude**

- `str`

**title**

- `str`

**address**

- `str`

**foursquare_id**

- `str`

**foursquare_type**

- `str`

**google_place_id**

- `str`

**google_place_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 venue.
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`

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.
telegram.InlineQueryResultVideo


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.

- **type**
  
  Type str

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

  Type str

- **video_url**
  
  A valid URL for the embedded video player or video file.

  Type str

- **mime_type**
  
  Mime type of the content of video url, “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

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 key-

    board attached to the message.
• **input_message_content** ([telegram.InputMessageContent](https://python-telegram-bot.readthedocs.io/en/stable/telegram.InputMessageContent.html), 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.

### telegram.InlineQueryResultVoice


```python
```


Represents a link to a voice recording in an .ogg container encoded with OPUS. By default, this voice recording will be sent by the user. Alternatively, you can use `input_message_content` to send a message with the specified content instead of the voice message.

**type** `str`

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

**voice_url** A valid URL for the voice recording.

**title** Recording title.

**caption** Optional. Caption, 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](https://python-telegram-bot.readthedocs.io/en/stable/telegram.ParseMode.html) for the available modes.

**caption_entities** Optional. List of special entities that appear in the caption, which can be specified instead of `parse_mode`.

**voice_duration** Optional. Recording duration in seconds.
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

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.

telegram.InputMessageContent
class telegram.InputMessageContent
Bases: telegram.base.TelegramObject
Base class for Telegram InputMessageContent Objects.

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.

**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`

**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.

### `telegram.InputLocationMessageContent`

**class** `telegram.InputLocationMessageContent` *(latitude: float, longitude: float, live_period: int = None, horizontal_accuracy: float = None, heading: int = None, proximity_alert_radius: int = 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.

**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

**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.

**telegram.InputVenueMessageContent**

**class** telegram.InputVenueMessageContent** (latitude: float, longitude: float, title: str, address: str, foursquare_id: str = None, foursquare_type: str = None, google_place_id: str = None, google_place_type: str = 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 Pace details are mutually exclusive. However, this behaviour is undocumented and might be changed by Telegram.

**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

foursquare_id
Optional. Foursquare identifier of the venue, if known.
Type str

foursquare_type
Optional. Foursquare type of the venue, if known.
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

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.

telegram.InputContactMessageContent

class telegram.InputContactMessageContent (phone_number: str, first_name: str, last_name: str = None, vcard: str = None, **kwargs)

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.

**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`

**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.

telegram.ChosenInlineResult

class telegram.ChosenInlineResult(result_id: str, from_user: telegram.user.User, query: str, location: telegram.files.location.Location = None, inline_message_id: str = 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.

**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`
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.

**Note:** It is necessary to enable inline feedback via @Botfather in order to receive these objects in updates.

### 3.2.55 Payments

**telegram.LabeledPrice**

```
class telegram.LabeledPrice(label: str, amount: int, **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.

**label**

Portion label.

- **Type** str

**amount**

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

- **Type** int

**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.

**telegram.Invoice**

```
class telegram.Invoice(title: str, description: str, start_parameter: str, currency: str, total_amount: int, **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.
**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

**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.

**telegram.ShippingAddress**


**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.

**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

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.

telemetg.OrderInfo

class telegram.OrderInfo (name: str = None, phone_number: str = None, email: str = None, shipping_address: str = 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.

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

Parameters
    • name (str, optional) – User name.
    • phone_number (str, optional) – User’s phone number.
    • email (str, optional) – User email.
• **shipping_address** ([`telegram.ShippingAddress`][1], optional) – User shipping address.

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

### `telegram.ShippingOption`

class `telegram.ShippingOption`

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.

**id**

Shipping option identifier.  
Type `str`

**title**

Option title.  
Type `str`

**prices**

List of price portions.  
Type `List[telegram.LabeledPrice]`

**Parameters**

• **id** (`str`) – Shipping option identifier.

• **title** (`str`) – Option title.

• **prices** (`List[telegram.LabeledPrice]`) – List of price portions.

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

### `telegram.SuccessfulPayment`

class `telegram.SuccessfulPayment`

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.

**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`  

**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.

**telegram.ShippingQuery**  

```python  
class telegram.ShippingQuery(id: str, from_user: telegram.user.User, invoice_payload: str, shipping_address: telegram.payment.shippingaddress.ShippingAddress, bot: 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.

**Note:**  
- In Python `from` is a reserved word, use `from_user` instead.

**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

Parameters
- id (str) – Unique query identifier.
- from_user (telegram.User) – User who sent the query.
- invoice_payload (str) – Bot specified invoice payload.
- shipping_address (telegram.ShippingAddress) – User specified shipping address.
- bot (telegram.Bot, optional) – The Bot to use for instance methods.
- **kwargs (dict) – Arbitrary keyword arguments.

answer(*args,**kwargs) → bool
Shortcut for:

```python
bot.answer_shipping_query(update.shipping_query.id, *args, **kwargs)
```

Parameters
- ok (bool) – Specify True if delivery to the specified address is possible and False if there are any problems (for example, if delivery to the specified address is not possible).
- shipping_options (List[telegram.ShippingOption], optional) – Required if ok is True. A JSON-serialized array of available shipping options.
- error_message (str, optional) – Required if ok is False. Error message in human readable form that explains why it is impossible to complete the order (e.g. “Sorry, delivery to your desired address is unavailable’). Telegram will display this message to the user.

telegram.PreCheckoutQuery
class telegram.PreCheckoutQuery(id: str, from_user: telegram.user.User, currency: str, total_amount: int, invoice_payload: str, shipping_option_id: str = None, order_info: telegram.payment.orderinfo.OrderInfo = None, bot: 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.

```python
id
Unique query identifier.
Type str
```

```python
from_user
User who sent the query.
Type telegram.User
```

```python
currency
Three-letter ISO 4217 currency code.
Type str
```

```python
total_amount
Total price in the smallest units of the currency.
Type int
```

```python
invoice_payload
Bot specified invoice payload.
Type str
```

```python
shipping_option_id
Optional. Identifier of the shipping option chosen by the user.
Type str
```

```python
order_info
Optional. Order info provided by the user.
Type telegram.OrderInfo
```

```python
bot
Optional. The Bot to use for instance methods.
Type telegram.Bot
```

**Parameters**
- `id` (str) – Unique query identifier.
- `from_user` (telegram.User) – User who sent the query.
- `currency` (str) – Three-letter ISO 4217 currency code.
- `total_amount` (int) – Total price in the smallest units of the currency (integer, not float/double). For example, for a price of US$ 1.45 pass `amount = 145`. See the `exp` parameter in `currencies.json`, it shows the number of digits past the decimal point for each currency (2 for the majority of currencies).
- `invoice_payload` (str) – Bot specified invoice payload.
- `shipping_option_id` (str, optional) – Identifier of the shipping option chosen by the user.
- `order_info` (telegram.OrderInfo, optional) – Order info provided by the user.
- `bot` (telegram.Bot, optional) – The Bot to use for instance methods.
- `**kwargs` (dict) – Arbitrary keyword arguments.
answer(*args, **kwargs) → bool

Shortcut for:

```python
bis歧答.checkout_query(update.pre_checkout_query.id, *args, **kwargs)
```

Parameters

- **ok**(bool) – Specify True if everything is alright (goods are available, etc.) and the bot is ready to proceed with the order. Use False if there are any problems.

- **error_message**(str, optional) – Required if ok is False. Error message in human readable form that explains the reason for failure to proceed with the 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.

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

### 3.2.56 Games

**telegram.Game**

```python
```

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.

**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
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()`. 1-4096 characters. Also found as `telegram.constants.MAX_MESSAGE_LENGTH`.
- **text_entities** (List[telegram.MessageEntity], optional) – Special entities that appear in text, such as usernames, URLs, bot commands, etc.
- **animation** (telegram.Animation, optional) – Animation that will be displayed in the game message in chats. Upload via BotFather.

`parse_text_entities` (types: List[str] = None) → Dict[telegram.messageentity.MessageEntity, str]

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: telegram.messageentity.MessageEntity) → str

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

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

Parameters

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

Returns The text of the given entity.

Return type str

Raises `RuntimeError` – If this game has no text.

```
from telegram import TelegramObject, Message
from telegram.messageentity import MessageEntity

class CallbackGame(TelegramObject):
    pass
```

**3.2. telegram package**
A placeholder, currently holds no information. Use BotFather to set up your game.

**telegram.GameHighScore**

class *telegram.GameHighScore* *(position: int, user: telegram.user.User, score: int)*  
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.  

**position**  
Position in high score table for the game.  
*Type* int

**user**  
User.  
*Type* telegram.User

**score**  
Score.  
*Type* int

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

**3.2.57 Passport**

**telegram.PassportElementError**

class *telegram.PassportElementError* *(source: str, type: str, message: str, **_kwargs)*  
Bases: *telegram.base.TelegramObject*  
Baseclass for the PassportElementError* classes.  
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.  

**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

Parameters  
• *source* (str) – Error source.  
• *type* (str) – The section of the user’s Telegram Passport which has the error.
• **kwargs (dict) – Arbitrary keyword arguments.

telegram.PassportElementErrorFile

class telegram.PassportElementErrorFile (type: str, file_hash: str, message: str, **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, data_hash and message are equal.

**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

**file_hash**
Base64-encoded file hash.

Type str

**message**
Error message.

Type str

Parameters

• type (str) – The section of the user’s Telegram Passport which has the issue, one of “utility_bill”, “bank_statement”, “rental_agreement”, “passport_registration”, “temporary_registration”.

• file_hash (str) – Base64-encoded file hash.

• message (str) – Error message.

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

telegram.PassportElementErrorReverseSide

class telegram.PassportElementErrorReverseSide (type: str, file_hash: str, message: str, **kwargs)
Bases: telegram.passport.passportelementerrors.PassportElementError

Represents an issue with the front side of a document. The error is considered resolved when the file with the reverse side of the document changes.

Objects of this class are comparable in terms of equality. Two objects of this class are considered equal, if their source, type, file_hash, data_hash and message are equal.

**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 reverse side of the document.

Type str

**message**
Error message.
**Type** `str`

**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.

```python
telegram.PassportElementErrorFrontSide
class telegram.PassportElementErrorFrontSide (type: str, file_hash: str, message: str, **kwargs)
```

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, data_hash` and `message` are equal.

**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`

**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.

```python
telegram.PassportElementErrorFiles
class telegram.PassportElementErrorFiles (type: str, file_hashes: str, message: str, **kwargs)
```

Represents an issue with a list of scans. 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_hashes, data_hash` and `message` are equal.
**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`

**file_hashes**
List of base64-encoded file hashes.

Type `List[str]`

**message**
Error message.

Type `str`

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.

---

**telegram.PassportElementErrorDataField**

**class** `telegram.PassportElementErrorDataField` (**type:** `str`, **field_name:** `str`, **data_hash:** `str`, **message:** `str`, **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.

**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`

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.

telegram.Credentials
class telegram.Credentials *(secure_data: SecureData, nonce: str, bot: Bot = None, **kwargs)*
Bases: telegram.base.TelegramObject
secure_data
  Credentials for encrypted data

  Type telegram.SecureData
nonce
  Bot-specified nonce

  Type str

telegram.DataCredentials
class telegram.DataCredentials *(data_hash: str, secret: str, **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

telegram.SecureData
class telegram.SecureData *(personal_details: SecureValue = None, passport: SecureValue = None, internal_passport: SecureValue = None, driver_license: SecureValue = None, identity_card: SecureValue = None, address: SecureValue = None, utility_bill: SecureValue = None, bank_statement: SecureValue = None, rental_agreement: SecureValue = None, passport_registration: SecureValue = None, temporary_registration: SecureValue = None, bot: 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
### Python Telegram Bot Documentation, Release 13.1

**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`

### telegram.FileCredentials

**class** `telegram.FileCredentials`
- `file_hash: str, secret: str, **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
  - `Type str`
secret
    Secret of encrypted file
    Type  str

telegram.IdDocumentData

class telegram.IdDocumentData(document_no: str, expiry_date: str, bot: 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: str, last_name: str, birth_date: str, gender: str, country_code: str, residence_country_code: str, first_name_native: str = None, last_name_native: str = None, middle_name: str = None, middle_name_native: str = None, bot: 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: str, street_line2: str, city: str, state: str, country_code: str, post_code: str, bot: 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: List[telegram.passport.encryptedpassportelement.EncryptedPassportElement], credentials: telegram.passport.credentials.EncryptedCredentials, bot: Bot = None, **kwargs)
Bases: telegram.base.TelegramObject
Contains information about Telegram Passport data shared with the bot by the user.

data
Array with encrypted information about documents and other Telegram Passport elements that was shared with the bot.
Type List[telegram.EncryptedPassportElement]

3.2. telegram package
credentials
Encrypted credentials.

Type `telegram.EncryptedCredentials`

bot
The Bot to use for instance methods.

Type `telegram.Bot`, optional

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.

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`.

`decrypted_credentials`
Lazily decrypt and return credentials that were used to decrypt the data. This object also contains the user specified payload as `decrypted_data.payload`.

Raises `telegram.TelegramDecryptionError` – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type `telegram.Credentials`

`decrypted_data`
Lazily decrypt and return information about documents and other Telegram Passport elements which were shared with the bot.

Raises `telegram.TelegramDecryptionError` – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

Type `List[telegram.EncryptedPassportElement]`

`telegram.PassportFile`

class `telegram.PassportFile`(*file_id: str, file_unique_id: str, file_date: int, file_size: int = None, bot: Bot = None, credentials: FileCredentials = 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.

**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

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.

get_file (timeout: int = None, api_kwargs: Dict[str, Any] = None) → File


Parameters

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

• **api_kwargs** (dict, optional) – Arbitrary keyword arguments to be passed to the Telegram API.

Returns telegram.File

Raises telegram.TelegramError
telegram.EncryptedPassportElement

class telegram.EncryptedPassportElement:


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.

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

reverse_side

    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`  

**selfie**

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`

**translation**


Type List[`telegram.PassportFile`]  

**hash**

Base64-encoded element hash for using in `telegram.PassportElementErrorUnspecified`.

Type `str`  

**bot**

Optional. The Bot to use for instance methods.

Type `telegram.Bot`

**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”.

• hash (str) – Base64-encoded element hash for using in telegram.
  PassportElementErrorUnspecified.
• bot (telegram.Bot, optional) – The Bot to use for instance methods.
• **kwargs (dict) – Arbitrary keyword arguments.

Note: This object is decrypted only when originating from telegram.PassportData.
decrypted_data.

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

**decrypted_secret**

Lazily decrypt and return secret.

*Rais*es `telegram.TelegramDecryptionError` – Decryption failed. Usually due to bad private/public key but can also suggest malformed/tampered data.

*Type* `str`

### 3.3 telegram.utils package

#### 3.3.1 telegram.utils.helpers Module

This module contains helper functions.

```python
telegram.utils.helpers.DEFAULT_FALSE = <telegram.utils.helpers.DefaultValue object>

Default False

*Type* `DefaultValue`
```
```python
telegram.utils.helpers.DEFAULT_NONE = <telegram.utils.helpers.DefaultValue object>

Default None

*Type* `DefaultValue`
```
```python
class telegram.utils.helpers.DefaultValue(value: Any = None)

Bases: object

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('`arg` = ' + str(arg))
```

This yields:

```console
>>> 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:
    ...
```
value
The value of the default argument

Type obj

Parameters value(obj) – The value of the default argument

telegram.utils.helpers.create_deep_linked_url(bot_username: str, payload: str = None, group: bool = False) → str

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
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: str) → Dict[str, Dict[Tuple, Any]]

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: str) → DefaultDict[int, Dict[Any, Any]]

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: Union[Message, Update]) → Optional[str]

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

Parameters entity (Update | Message) –

Returns One of Message.MESSAGE_TYPES
Return type: str

```
telegram.utils.helpers.encode_conversations_to_json(conversations: Dict[str, Dict[Tuple, Any]]) → str
```

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: str, version: int = 1, entity_type: str = None) → str
```

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: Optional[int], tzinfo: datetime.tzinfo = <UTC>) → Optional[datetime.datetime]
```

Converts an (integer) unix timestamp to a timezone aware datetime object. None’s are left alone (i.e. ``from_timestamp (None)` is None).

**Parameters**

- **unixtime (int)** – Integer POSIX timestamp.
- **tzinfo (datetime.tzinfo, optional)** – The timezone, the timestamp is to be converted to. Defaults to UTC.

**Returns**

timezone aware equivalent datetime.datetime value if timestamp is not None; else None

```
telegram.utils.helpers.get_signal_name(signum: int) → str
```

Returns the signal name of the given signal number.

```
telegram.utils.helpers.is_local_file(obj: Union[str, pathlib.Path, None]) → bool
```

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: Union[int, str], name: str) → str
```

Parameters

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

**Returns**

The inline mention for the user as html.

**Return type**

str

```
telegram.utils.helpers.mention_markdown(user_id: Union[int, str], name: str, version: int = 1) → str
```

Parameters

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

3.3. `telegram.utils` package 285
• **version**(int | str) – Use to specify the version of telegrams Markdown. Either 1 or 2. Defaults to 1

Returns The inline mention for the user as markdown.

Return type str

```
```

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 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 | 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: Union[int, float, datetime.timedelta, datetime.datetime, datetime.time], reference_timestamp: float = None, tzinfo: pytz.tzinfo.BaseTzInfo = None) → float
```

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. Any 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** *(datetime.tzinfo, optional)*  – If \( t \) is a naive object from the `datetime` module, it will be interpreted as this timezone. Defaults to `pytz.utc`.

**Returns**

*(float | None)*  The return value depends on the type of argument \( t \). If \( t \) is given as a

time increment (i.e. as a `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.

**Raises**  Type`Error` – if \( t \)’s type is not one of those described above

```python
from telegram.utils.helpers import to_timestamp
dt_obj = Union[int, float, datetime.timedelta,
datetime.datetime, datetime.time, None],
reference_timestamp: float = None, tzinfo:
pytz.tzinfo.BaseTzInfo = 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.
```

### 3.3.2 `telegram.utils.promise.Promise`

```python
class telegram.utils.promise.Promise(pooled_function: Callable[..., RT], args: Union[List[T], Tuple],
kwargs: Dict[str, Any], update: Union[str, Update] = None, error_handling:
bool = 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, 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

error_handling
Optional. Whether exceptions raised by func may be handled by error handlers. Defaults to True.
Type bool

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

result (timeout: float = None) → Optional[RT]
Return the result of the Promise.

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

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

Raises Any exception raised by pooled_function.

run () → None
Calls the pooled_function callable.

3.3.3 telegram.utils.request.Request

class telegram.utils.request.Request (con_pool_size: int = 1, proxy_url: str = None, urllib3_proxy_kwargs: Dict[str, Any] = None, connect_timeout: float = 5.0, read_timeout: float = 5.0)

Bases: object

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

Parameters

• con_pool_size (int) – Number of connections to keep in the connection pool.

• proxy_url (str) – The URL to the proxy server. For example: http://127.0.0.1:3128.

• urllib3_proxy_kwargs (dict) – Arbitrary arguments passed as-is to urllib3.ProxyManager. This value will be ignored if proxy_url is not set.

• connect_timeout (int|float) – The maximum amount of time (in seconds) to wait for a connection attempt to a server to succeed. None will set an infinite timeout for connection attempts. (default: 5.)

• read_timeout (int|float) – The maximum amount of time (in seconds) to wait between consecutive read operations for a response from the server. None will set an infinite timeout. This value is usually overridden by the various telegram.Bot methods. (default: 5.)
con_pool_size
The size of the connection pool used.

download(url: str, filename: str, timeout: float = None) → None
Download 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).
- **filename** (str) – The filename within the path to download the file.

post(url: str, data: Dict[str, Any], timeout: float = None) → Union[Dict[str, Any], bool]
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: str, timeout: float = None) → bytes
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).

3.3.4 telegram.utils.types Module

This module contains custom typing aliases.

- **ConversationDict** = typing.Dict[typing.Tuple[int, ...], typing.Union[object, NoneType]]
  Dicts as maintained by the telegram.ext.ConversationHandler.

- **FileInput** = typing.Union[str, typing.IO, ForwardRef('InputFile'), pathlib.Path]
  Valid input for passing files to Telegram. Either a file id as string, a file like object or a local file path as string or pathlib.Path.

- **FileLike** = typing.Union[typing.IO, ForwardRef('InputFile')]
  Either an open file handler or a telegram.InputFile.

- **HandlerArg** = typing.Union[str, ForwardRef('Update')]
  The argument that handlers parse for telegram.ext.handler.check_update() etc.

- **JSONDict** = typing.Dict[str, typing.Any]
  Dictionary containing response from Telegram or data to send to the API.

- **SLT** = typing.Union[~RT, typing.List[~RT], typing.Tuple[~RT, ...]]
  Single instance or list/tuple of instances.
3.4 Changelog

3.4.1 Changelog

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/Usernames 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 en 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 a 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:

• Stabalize 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, #1750, #1752)
• Fix spelling issue for encode_conversations_to_json (#1661)
• Remove double assignment 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-urlib3 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, #1612, #1611, #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 depndency (#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_filter 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 types: PassportData, PassportFile, EncryptedPassportElement, EncryptedCredentials, PassportElementType, PassportElementTypeError, PassportElementTypeErrorDataField, PassportElementTypeErrorFrontSide, PassportElementTypeErrorReverseSide, PassportElementTypeErrorSelfie, PassportElementTypeErrorFile and PassportElementTypeErrorFiles.
  - 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.send_animation, add width, height, and duration to Animation, and add Filters.animation. (#1172)

Non Bot API 4.0 changes:

- Minor integer comparison fix (#1147)
- Fix Filters.regex failing on non-text message (#1158)
- Fix ProcessLookupError if process finishes before we kill it (#1126)
- Add t.me links for User, Chat and Message if available and update User.mention_* (#1092)
- Fix mention_markdown/html on py2 (#1112)
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 compatible changes and changed defaults

- JobQueue: Remove deprecated prevent_autostart & put() (PR #1012)
• Bot, Updater: Remove deprecated network_delay (PR #1012)
• Remove deprecated Message.new_chat_member (PR #1012)
• Retry bootstrap phase indefinitely (by default) on network errors (PR #1018)

New Features
• Support v3.6 API (PR #1006)
• User.full_name convenience property (PR #949)
• Add send_phone_number_to_provider and send_email_to_provider arguments to send_invoice (PR #986)
• Bot: Add shortcut methods reply_{markdown,html} (PR #827)
• Bot: Add shortcut method reply_media_group (PR #994)
• Added utils.helpers.effective_message_type (PR #826)
• Bot.get_file now allows passing a file in addition to file_id (PR #963)
• Add .get_file() to Audio, Document, PhotoSize, Sticker, Video, VideoNote and Voice (PR #963)
• Add .send_*() methods to User and Chat (PR #963)
• Get jobs by name (PR #1011)
• Add Message caption html/markdown methods (PR #1013)
• File.download_as_bytearray - new method to get a d/led file as bytearray (PR #1019)
• File.download(): Now returns a meaningful return value (PR #1019)
• Added conversation timeout in ConversationHandler (PR #895)

Changes
• Store bot in PreCheckoutQuery (PR #953)
• Updater: Issue INFO log upon received signal (PR #951)
• JobQueue: Thread safety fixes (PR #977)
• WebhookHandler: Fix exception thrown during error handling (PR #985)
• Explicitly check update.effective_chat in ConversationHandler.check_update (PR #959)
• Updater: Better handling of timeouts during get_updates (PR #1007)
• Remove unnecessary to_dict() (PR #834)
• CommandHandler - ignore strings in entities and “/” followed by whitespace (PR #1020)
• Documentation & style fixes (PR #942, PR #956, PR #962, PR #980, PR #983)

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). - Unitests are now also running on AppVeyor (Windows VM). - Various unitest 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 handler for 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 comparision 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

Released 1.3

• Added support to `setWebhook` (special thanks to macrojames)

2015-07-09

Released 1.2

• `CustomKeyboard` classes now available
• Emojis available
• PEP8 improvements

2015-07-08

Released 1.1

• PyPi package now available

2015-07-08

Released 1.0

• Initial checkin of python-telegram-bot
Python Module Index

**t**
- telegram.constants, 152
- telegram.error, 160
- telegram.ext.filters, 12
- telegram.utils.helpers, 283
- telegram.utils.types, 289
Symbols

__call__() (telegram.ext.DelayQueue method), 34
__call__() (telegram.ext.MessageQueue method), 34
__init__() (telegram.ext.DelayQueue method), 34
__init__() (telegram.ext.MessageQueue method), 33
__weakref__ (telegram.ext.MessageQueue attribute), 33
add_bot_ids() (telegram.ext.filters.Filters.via_bot method), 24
add_chat_ids() (telegram.ext.filters.Filters.chat method), 14
add_chat_ids() (telegram.ext.filters.Filters.sender_chat method), 20
add_error_handler() (telegram.ext.Dispatcher method), 9
add_handler() (telegram.ext.Dispatcher method), 10
add_sticker_to_set() (telegram.Bot method), 81
add_user_ids() (telegram.ext.filters.Filters.user method), 23
add_usernames() (telegram.ext.filters.Filters.chat method), 14
add_usernames() (telegram.ext.filters.Filters.sender_chat method), 20
add_usernames() (telegram.ext.filters.Filters.user method), 23
add_usernames() (telegram.ext.filters.Filters.via_bot method), 24
address (telegram.ChatLocation attribute), 146
address (telegram.InlineQueryResultVenue attribute), 251
address (telegram.InputVenueMessageContent attribute), 259
address (telegram.SecureData attribute), 275
address (telegram.Venue attribute), 214
addStickerToSet() (telegram.Bot method), 81
ADMINISTRATOR (telegram.ChatMember attribute), 149
all (telegram.ext.filters.Filters attribute), 12
ALL_EMOJI (telegram.Dice attribute), 158
ALL_TYPES (telegram.MessageEntity attribute), 195
allow_edited (telegram.ext.CommandHandler attribute), 47
allow_empty (telegram.ext.filters.Filters.chat attribute), 13
allow_empty (telegram.ext.filters.Filters.sender_chat attribute), 19
allow_empty (telegram.ext.filters.Filters.user attribute), 22
allow_empty (telegram.ext.filters.Filters.via_bot attribute), 24
allow_reentry (telegram.ext.ConversationHandler attribute), 45
allow_sending_without_reply (telegram.extDefaults attribute), 36
allowed_updates (telegram.WebhookInfo attribute), 219
allows_multiple_answers (telegram.Poll attribute), 198
amount (telegram.LabeledPrice attribute), 261
Animation (class in telegram), 77
animation (telegram.ext.filters.Filters attribute), 12
animation (telegram.Game attribute), 268
animation (telegram.Message attribute), 178
ANONYMOUS_ADMIN_ID (in module telegram.constants), 153
answer() (telegram.CallbackQuery method), 135
answer() (telegram.InlineQuery method), 224
answer() (telegram.ShippingQuery method), 268
answer() (telegram.ShippingQuery method), 266
answer_callback_query() (telegram.Bot method), 82
answer_inline_query() (telegram.Bot method), 83
answer_pre_checkout_query() (telegram.Bot method), 84
answer_shipping_query() (telegram.Bot method), 84
answerCallbackQuery() (telegram.Bot method), 82
answerInlineQuery() (telegram.Bot method), 82
answerPreCheckoutQuery() (telegram.Bot method), 82
answerShippingQuery() (telegram.Bot method), 82
apk (telegram.ext.filters.Filters attribute), 16
application (telegram.ext.filters.Filters attribute), 16
args (telegram.ext.CallbackContext attribute), 35
args (telegram.utils.promise.Promise attribute), 287
async_args (telegram.ext.CallbackContext attribute), 35
async_kwargs (telegram.ext.CallbackContext attribute), 35
attach (telegram.InputFile attribute), 165
Audio (class in telegram), 79
audio (telegram.ext.filters.Filters attribute), 12, 16
audio (telegram.Message attribute), 178
audio_duration (telegram.InlineQueryResultAudio attribute), 227
audio_file_id (telegram.InlineQueryResultCachedAudio attribute), 229
audio_url (telegram.InlineQueryResultAudio attribute), 227
author_signature (telegram.Message attribute), 179

B

BadRequest 160
bank_statement (telegram.SecureData attribute), 275
BaseFilter (class in telegram.ext.filters), 24
BasePersistence (class in telegram.ext), 70
BASKETBALL (telegram.Dice attribute), 158
basketball (telegram.ext.filters.Filters attribute), 15
big_file_id (telegram.ChatPhoto attribute), 151
big_file_unique_id (telegram.ChatPhoto attribute), 151
bio (telegram.Chat attribute), 139
birth_date (telegram.PersonalDetails attribute), 276
BOLD (telegram.MessageEntity attribute), 195
Bot (class in telegram), 80
bot (telegram.Animation attribute), 78
bot (telegram.Audio attribute), 80
bot (telegram.CallbackQuery attribute), 134
bot (telegram.Document attribute), 159
bot (telegram.EncryptedPassportElement attribute), 281
bot (telegram.ext.CallbackContext attribute), 36
bot (telegram.ext.Dispatcher attribute), 8
bot (telegram.ext.JobQueue attribute), 28
bot (telegram.ext.Updater attribute), 5
bot (telegram.Message attribute), 180
bot (telegram.PassportData attribute), 278
bot (telegram.PassportFile attribute), 279
bot (telegram.PhotoSize attribute), 197
bot (telegram.PreCheckoutQuery attribute), 267
bot (telegram.ShippingQuery attribute), 266
bot (telegram.Sticker attribute), 221
bot (telegram.User attribute), 209
bot (telegram.Video attribute), 216
bot (telegram.VideoNote attribute), 217
bot (telegram.Voice attribute), 218
BOT_COMMAND (telegram.MessageEntity attribute), 196
bot_data (telegram.ext.CallbackContext attribute), 35
bot_data (telegram.ext.DictPersistence attribute), 76
bot_data (telegram.ext.Dispatcher attribute), 9
bot_data_json (telegram.ext.DictPersistence attribute), 76
bot_ids (telegram.ext.filters.Filters.via_bot attribute), 23
bot_username (telegram.LoginUrl attribute), 175
BotCommand (class in telegram), 133
burst_limit (telegram.ext.DelayQueue attribute), 33

C
callback (telegram.ext.CallbackQueryHandler attribute), 40
callback (telegram.ext.ChosenInlineResultHandler attribute), 42
callback (telegram.ext.CommandHandler attribute), 47
callback (telegram.ext.Handler attribute), 38
callback (telegram.ext.InlineQueryHandler attribute), 50
callback (telegram.ext.Job attribute), 27
callback (telegram.ext.MessageHandler attribute), 52
callback (telegram.ext.PollAnswerHandler attribute), 54
callback (telegram.ext.PollHandler attribute), 56
callback (telegram.ext.PreCheckoutQueryHandler attribute), 57
callback (telegram.extPREFIXHandler attribute), 59
callback (telegram.ext.RegexHandler attribute), 62
callback (telegram.ext.ShippingQueryHandler attribute), 64
callback (telegram.ext.StringCommandHandler attribute), 65
callback (telegram.ext.StringRegexHandler attribute), 67
callback (telegram.ext.TypeRegexHandler attribute), 69
callback_data (telegram.InlineKeyboardButton attribute), 163
callback_game (telegram.InlineKeyboardButton attribute), 163
callback_query (telegram.Update attribute), 206
CallbackContext (class in telegram.ext), 34
CallbackGame (class in telegram), 269
CallbackQuery (class in telegram), 134
CallbackQueryHandler (class in telegram.ext), 40
can_add_web_page_previews (telegram.ChatMember attribute), 148
can_add_web_page_previews (telegram.ChatPermissions attribute), 150
can_be_edited (telegram.ChatMember attribute), 147
can_change_info (telegram.ChatMember attribute), 147
can_change_info (telegram.ChatPermissions attribute), 150
can_delete_messages (telegram.ChatMember attribute), 148
can_edit_messages (telegram.ChatMember attribute), 148
can_invite_users (telegram.ChatMember attribute), 148
can_invite_users (telegram.ChatPermissions attribute), 150
can_join_groups (telegram.Bot attribute), 85
can_join_groups (telegram.User attribute), 208
can_pin_messages (telegram.ChatMember attribute), 148
can_pin_messages (telegram.ChatPermissions attribute), 150
can_post_messages (telegram.ChatMember attribute), 147
can_promote_members (telegram.ChatMember attribute), 148
can_read_all_group_messages (telegram.Bot attribute), 85
can_read_all_group_messages (telegram.User attribute), 208
can_restrict_members (telegram.ChatMember attribute), 148
can_send_media_messages (telegram.ChatMember attribute), 148
can_send_media_messages (telegram.ChatPermissions attribute), 150
can_send_messages (telegram.ChatMember attribute), 148
can_send_other_messages (telegram.ChatPermissions attribute), 150
can_send_other_messages (telegram.ChatMember attribute), 148
can_send_polls (telegram.ChatMember attribute), 148
can_send_polls (telegram.ChatPermissions attribute), 150
can_set_sticker_set (telegram.Chat attribute), 139
caption (telegram.InlineQueryResultAudio attribute), 227
caption (telegram.InlineQueryResultCachedAudio attribute), 229
caption (telegram.InlineQueryResultCachedDocument attribute), 230
caption (telegram.InlineQueryResultCachedGif attribute), 232
caption (telegram.InlineQueryResultCachedMpeg4Gif attribute), 233
caption (telegram.InlineQueryResultCachedPhoto attribute), 235
caption (telegram.InlineQueryResultCachedVideo attribute), 237
caption (telegram.InlineQueryResultCachedVoice attribute), 238
caption (telegram.InlineQueryResultDocument attribute), 241
caption (telegram.InlineQueryResultGif attribute), 244
caption (telegram.InlineQueryResultMpeg4Gif attribute), 248
caption (telegram.InlineQueryResultPhoto attribute), 249
caption (telegram.InlineQueryResultVideo attribute), 253
caption (telegram.InlineQueryResultVoice attribute), 255
caption (telegram.InputMediaAnimation attribute), 166
caption (telegram.InputMediaAudio attribute), 168
caption (telegram.InputMediaDocument attribute), 169
caption (telegram.InputMediaPhoto attribute), 170
caption (telegram.InputMediaVideo attribute), 171
caption (telegram.Message attribute), 178
caption_entities (telegram.InlineQueryResultAudio attribute), 227
caption_entities (telegram.InlineQueryResultCachedAudio attribute), 229
caption_entities (telegram.InlineQueryResultCachedDocument attribute), 230
caption_entities (telegram.InlineQueryResultCachedGif attribute), 232
caption_entities (telegram.InlineQueryResultCachedMpeg4Gif attribute), 233
caption_entities (telegram.InlineQueryResultCachedPhoto attribute), 235
caption_entities (telegram.InlineQueryResultCachedVideo attribute), 237
caption_entities (telegram.InlineQueryResultCachedVoice attribute), 238
Index 317
gram.InlineQueryResultCachedVoice attribute), 238
caption_entities (telegram.InlineQueryResultDocument attribute), 241
caption_entities (telegram.InlineQueryResultGif attribute), 244
caption_entities (telegram.InlineQueryResultMpeg4Gif attribute), 248
caption_entities (telegram.InlineQueryResultPhoto attribute), 250
caption_entities (telegram.InlineQueryResultVideo attribute), 253
caption_entities (telegram.InlineQueryResultVoice attribute), 255
caption_entities (telegram.InputMediaAnimation attribute), 166
caption_entities (telegram.InputMediaAudio attribute), 168
caption_entities (telegram.InputMediaDocument attribute), 169
caption_entities (telegram.InputMediaPhoto attribute), 171
caption_entities (telegram.InputMediaVideo attribute), 171
caption_entities (telegram.Message attribute), 177
caption_html (telegram.Message attribute), 183
caption_html_urlencoded (telegram.Message attribute), 183
caption_markdown (telegram.Message attribute), 183
caption_markdown_urlencoded (telegram.Message attribute), 183
caption_markdown_v2 (telegram.Message attribute), 183
caption_markdown_v2_urlencoded (telegram.Message attribute), 184
CASHTAG (telegram.MessageEntity attribute), 196
category (telegram.ext.filters.Filters attribute), 15
CHANNEL (telegram.Chat attribute), 141
class (telegram.Message attribute), 138
chat (telegram.Message attribute), 177
channel (telegram.Message attribute), 141
channel_chat_created (telegram.Message attribute), 179
channel_post (telegram.ext.filters.Filters attribute), 22
channel_post (telegram.Message attribute), 206
channel_post_updates (telegram.ext.MessageHandler attribute), 53
channel_posts (telegram.ext.filters.Filters attribute), 22
Chat (class in telegram), 138
chat (telegram.Message attribute), 177
CHAT_CHANNEL (in module telegram.constants), 154
chat_created (telegram.ext.filters.Filters attribute), 21
chat_data (telegram.ext.CallbackContext attribute), 35
chat_data (telegram.ext.DictPersistence attribute), 76
chat_data (telegram.ext.Message attribute), 184
chat_data (telegram.ext.Dispatcher attribute), 9
chat_data (telegram.ext.DictPersistence attribute), 76
CHAT_GROUP (in module telegram.constants), 153
chat_id (telegram.Message attribute), 184
chat_ids (telegram.ext.filters.Filters.chat attribute), 13
chat_ids (telegram.ext.filters.Filters.sender_chat attribute), 19
chat_instance (telegram.CallbackQuery attribute), 134
CHAT_PRIVATE (in module telegram.constants), 153
CHAT_SUPERGROUP (in module telegram.constants), 153
chat_type (telegram.ext.filters.Filters attribute), 14
ChatAction (class in telegram), 146
CHATACTION_FIND_LOCATION (in module telegram.constants), 154
CHATACTION_RECORD_AUDIO (in module telegram.constants), 154
CHATACTION_RECORD_VIDEO (in module telegram.constants), 154
CHATACTION_RECORD_VIDEO_NOTE (in module telegram.constants), 154
CHATACTION_TYPING (in module telegram.constants), 154
CHATACTION_UPLOAD_AUDIO (in module telegram.constants), 154
CHATACTION_UPLOAD_AUDIO_INFORMATION (in module telegram.constants), 154
CHATACTION_UPLOAD_DOCUMENT (in module telegram.constants), 154
CHATACTION_UPLOAD_PHOTO (in module telegram.constants), 154
CHATACTION_UPLOAD_VIDEO (in module telegram.constants), 154
CHATACTION_UPLOAD_VIDEO_NOTE (in module telegram.constants), 154
ChatLocation (class in telegram), 146
ChatMember (class in telegram), 147
CHATMEMBER_ADMINISTRATOR (in module telegram.constants), 154
CHATMEMBER_CREATOR (in module telegram.constants), 154
CHATMEMBER_KICKED (in module telegram.constants), 154
CHATMEMBER_LEFT (in module telegram.constants), 155
CHATMEMBER_MEMBER (in module tele-
country_code (telegram.PersonalDetails attribute), 276

createNewStickerSet() (telegram.Bot method), 86

createDeepLinkedUrl() (in module telegram.utils.helpers), 284

createNewStickerSet() (telegram.Bot method), 86

CREATOR (telegram.ChatMember attribute), 149

Credentials (class in telegram), 274
credentials (telegram.PassportData attribute), 278
currency (telegram.Invoice attribute), 262
currency (telegram.PreCheckoutQuery attribute), 267
currency (telegram.SuccessfulPayment attribute), 264
custom_title (telegram.ChatMember attribute), 147

date (telegram.Message attribute), 177
de_json() (telegram.Update class method), 207
decode_conversations_from_json() (in module telegram.utils.types), 289
decode_user_chat_data_from_json() (in module telegram.utils.types), 284
decrypted_credentials (telegram.PassportData attribute), 278
decrypted_data (telegram.EncryptedCredentials attribute), 282
decrypted_data (telegram.PassportData attribute), 278
decrypted_secret (telegram.EncryptedCredentials attribute), 283
DEFAULT_FALSE (in module telegram.utils.types), 283
DEFAULT_NONE (in module telegram.utils.types), 283

defaults (class in telegram), 36
defaultValue (class in telegram.utils.types), 283
DelayQueue (class in telegram), 33
delete() (telegram.Message method), 184
deleteChatPhoto() (telegram.ext.filters.Filters method), 159
delete_chat_photo() (telegram.Bot method), 88
delete_chat_photo() (telegram.ext.filters.Filters attribute), 21
delete_chat_photo() (telegram.Message attribute), 179
delete_chat_photo() (telegram.Bot method), 88
deleteChatPhoto() (telegram.Bot method), 89
deleteMessage() (telegram.Bot method), 88
delete_message() (telegram.Bot method), 88
delete_message() (telegram.CallbackQuery method), 135
delete_sticker_from_set() (telegram.Bot method), 89
delete_webhook() (telegram.Bot method), 89
deleteChatStickerSet() (telegram.Bot method), 87
deleteChatStickerSet() (telegram.Bot method), 87
deleteMessage() (telegram.Bot method), 87
deleteStickerFromSet() (telegram.Bot method), 87
deleteWebhook() (telegram.Bot method), 88
description (telegram.BotCommand attribute), 133
description (telegram.Chat attribute), 139
description (telegram.Game attribute), 268
description (telegram.InlineQueryResultArticle attribute), 226
description (telegram.InlineQueryResultCachedDocument attribute), 230
description (telegram.InlineQueryResultCachedPhoto attribute), 235
description (telegram.InlineQueryResultCachedVideo attribute), 237
Dispatcher (class in telegram), 158
DICE (telegram.Dice attribute), 159
dice (telegram.ext.filters.Filters attribute), 15
dice (telegram.Message attribute), 180
DICE_ALL_EMOJI (in module telegram.constants), 155
DICE_BASKETBALL (in module telegram.constants), 155
DICE_DARTS (in module telegram.constants), 155
DICE_DICE (in module telegram.constants), 155
DICE_FOOTBALL (in module telegram.constants), 155
DICE_SLOT_MACHINE (in module telegram.constants), 155
DictPersistence (class in telegram.ext), 75
disable_content_type_detection (telegram.InputMediaDocument attribute), 170
disable_notification (telegram.ext.Defaults attribute), 36
disable_web_page_preview (telegram.ext.Defaults attribute), 36
disable_web_page_preview (telegram.InputTextMessageContent attribute), 257
dispatch_error (telegram.ext.Dispatcher method), 10
Dispatcher (class in telegram.ext), 8
dispatcher (telegram.ext.CallbackContext attribute), 36
dispatcher (telegram.ext.Updater attribute), 6
DispatcherHandlerStop (class in telegram.ext), 12
distance (telegram.ProximityAlertTriggered attribute), 201
doc (telegram.ext.filters.Filters attribute), 16
Document (class in telegram), 159
document (telegram.ext.filters.Filters attribute), 15
document (telegram.Message attribute), 178
document_file_id (telegram.InlineQueryResultCachedDocument attribute), 230
document_no (telegram.IdDocumentData attribute), 276
document_url (telegram.InlineQueryResultDocument attribute), 241
docs (telegram.ext.filters.Filters attribute), 16
done (telegram.utils.promise.Promise attribute), 288
download () (telegram.File method), 161
download () (telegram.utils.request.Request method), 289
download_as_bytearray () (telegram.File method), 162
driver_license (telegram.SecureData attribute), 275
duration (telegram.Animation attribute), 78
duration (telegram.Audio attribute), 79
duration (telegram.InputMediaAnimation attribute), 167
duration (telegram.InputMediaAudio attribute), 168
duration (telegram.InputMediaVideo attribute), 172
duration (telegram.Video attribute), 215
duration (telegram.VideoNote attribute), 217
duration (telegram.Voice attribute), 218

edit_caption () (telegram.Message method), 184
edit_date (telegram.Message attribute), 177
edit_live_location () (telegram.Message method), 185
edit_media () (telegram.Message method), 185
edit_message_caption () (telegram.Bot method), 90
edit_message_caption () (telegram.CallbackQuery method), 135
edit_message_live_location () (telegram.Bot method), 91
edit_message_live_location () (telegram.CallbackQuery method), 136
edit_message_media () (telegram.Bot method), 92
edit_message_media () (telegram.CallbackQuery method), 136
edit_message_reply_markup () (telegram.Bot method), 92
edit_message_reply_markup () (telegram.CallbackQuery method), 136
edit_message_text () (telegram.Bot method), 93
edit_message_text () (telegram.CallbackQuery method), 137
edit_reply_markup () (telegram.Message method), 185
edit_text () (telegram.Message method), 185
edited_channel_post (telegram.ext.filters.Filters attribute), 22
edited_channel_post (telegram.Update attribute), 206
edited_message (telegram.ext.filters.Filters attribute), 22
edited_message (telegram.Update attribute), 206
edited_updates (telegram.ext.MessageHandler attribute), 53
editMessageCaption () (telegram.Bot method), 89
editMessageLiveLocation () (telegram.Bot method), 90
editMessageMedia () (telegram.Bot method), 90
filter() (telegram.ext.filters.InvertedFilter method), 26
filter() (telegram.ext.filters.MergedFilter method), 26
filter() (telegram.ext.filters.MessageFilter method), 25
filter() (telegram.ext.filters.UpdateFilter method), 26
filter() (telegram.ext.filters.XORFilter method), 26
Filters (class in telegram.ext.filters), 12
Filters (telegram.ext.CommandHandler attribute), 47
Filters (telegram.ext.MessageHandler attribute), 52
Filters (telegram.ext.PrefixHandler attribute), 59
Filters.caption_entity (class in telegram.ext.filters), 12
Filters.caption_regex (class in telegram.ext.filters), 13
Filters.chat (class in telegram.ext.filters), 13
Filters.entity (class in telegram.ext.filters), 17
Filters.language (class in telegram.ext.filters), 18
Filters.regex (class in telegram.ext.filters), 18
Filters.sender_chat (class in telegram.ext.filters), 19
Filters.via_bot (class in telegram.ext.filters), 22
FIND_LOCATION (class in telegram.ext.filters), 23
forward_signature (telegram.Message attribute), 179
forward_text (telegram.LoginUrl attribute), 175
forwarded (telegram.ext.filters.Filters attribute), 18
forwardMessage() (telegram.Bot method), 94
foursquare_id (telegram.InlineQueryResultVenue attribute), 251
foursquare_id (telegram.InputVenueMessageContent attribute), 259
foursquare_id (telegram.Venue attribute), 214
foursquare_type (telegram.InlineQueryResultVenue attribute), 251
foursquare_type (telegram.InputVenueMessageContent attribute), 259
foursquare_type (telegram.Venue attribute), 214
from_button() (telegram.InlineKeyboardMarkup class method), 164
from_button() (telegram.ReplyKeyboardMarkup class method), 203
from_column() (telegram.InlineKeyboardMarkup class method), 165
from_column() (telegram.ReplyKeyboardMarkup class method), 204
from_row() (telegram.InlineKeyboardMarkup class method), 165
from_row() (telegram.ReplyKeyboardMarkup class method), 204
from_timestamp() (in module telegram.utils.helpers), 285
from_user (telegram.CallbackQuery attribute), 134
from_user (telegram.ChosenInlineResult attribute), 260
from_user (telegram.InlineQuery attribute), 224
from_user (telegram.Message attribute), 176
from_user (telegram.PreCheckoutQuery attribute), 267
from_user (telegram.ShippingQuery attribute), 265
full_name (telegram.User attribute), 209

G
Game (class in telegram), 268
game (telegram.ext.filters.Filters attribute), 18
game (telegram.Message attribute), 178
game_short_name (telegram.CallbackQuery attribute), 134
game_short_name (telegram.InlineQueryResultGame attribute), 243
GameHighScore (class in telegram), 270
gender (telegram.PersonalDetails attribute), 270
get_administrators() (telegram.Chat method), 141
get_big_file() (telegram.ChatPhoto method), 152
get_bot_data() (telegram.ext.BasePersistence method), 71
get_bot_data() (telegram.ext.DictPersistence method), 76
get_bot_data() (telegram.ext.PicklePersistence method), 74
get_chat() (telegram.Bot method), 96
get_chat_administrators() (telegram.Bot method), 96
get_chat_data() (telegram.ext.BasePersistence method), 71
get_chat_data() (telegram.ext.DictPersistence method), 76
get_chat_data() (telegram.ext.PicklePersistence method), 74
get_chat_member() (telegram.Bot method), 96
get_chat_members_count() (telegram.Bot method), 97
get_conversations() (telegram.ext.BasePersistence method), 72
get_conversations() (telegram.ext.DictPersistence method), 76
get_conversations() (telegram.ext.PicklePersistence method), 74
get_file() (telegram.Animation method), 79
get_file() (telegram.Audio method), 80
get_file() (telegram.Bot method), 97
get_file() (telegram.Document method), 160
get_file() (telegram.PassportFile method), 279
get_file() (telegram.PhotoSize method), 197
get_file() (telegram.Sticker method), 221
get_file() (telegram.Video method), 216
get_file() (telegram.VideoNote method), 217
get_file() (telegram.Voice method), 218
get_game_high_scores() (telegram.Bot method), 97
get_game_high_scores() (telegram.CallbackQuery method), 137
get_game_high_scores() (telegram.Message method), 186
get_instance() (telegram.ext.Dispatcher class method), 10
get_jobs_by_name() (telegram.ext.JobQueue method), 28
get_me() (telegram.Bot method), 98
get_member() (telegram.Chat method), 141
get_members_count() (telegram.Chat method), 141
get_my_commands() (telegram.Bot method), 98
get_profile_photos() (telegram.Bot method), 209
get_signal_name() (in module telegram.utils.helpers), 285
get_small_file() (telegram.ChatPhoto method), 152
get_sticker_set() (telegram.Bot method), 98
get_updates() (telegram.Bot method), 99
get_user_data() (telegram.ext.BasePersistence method), 72
get_user_data() (telegram.ext.DictPersistence method), 76
get_user_data() (telegram.ext.PicklePersistence method), 74
get_user_profile_photos() (telegram.Bot method), 99
get_webhook_info() (telegram.Bot method), 100
get_chat() (telegram.Bot method), 95
get_chatAdministrators() (telegram.Bot method), 95
getChatMember() (telegram.Bot method), 95
getChatMembersCount() (telegram.Bot method), 95
gFile() (telegram.Bot method), 95
gGameHighScores() (telegram.Bot method), 95
gMe() (telegram.Bot method), 95
gMyCommands() (telegram.Bot method), 95
gStickerSet() (telegram.Bot method), 95
gUpdates() (telegram.Bot method), 95
gUserProfilePhotos() (telegram.Bot method), 95
getWebhookInfo() (telegram.Bot method), 96
gif (telegram.ext.filters.Filters attribute), 16
gif_duration (telegram.InlineQueryResultGif attribute), 244
gif_file_id (telegram.InlineQueryResultCachedGif attribute), 231
gif_height (telegram.InlineQueryResultGif attribute), 244
gif_url (telegram.InlineQueryResultGif attribute), 243
gif_width (telegram.InlineQueryResultGif attribute), 243
google_place_id (telegram.InlineQueryResultVenue attribute), 251
google_place_id (telegram.InputVenueMessageContent attribute), 259
google_place_id (telegram.Venue attribute), 214
google_place_type (telegram.InlineQueryResultVenue attribute), 251
google_place_type (telegram.InputVenueMessageContent attribute), 259
google_place_type (telegram.Venue attribute), 214
GROUP (telegram.Chat attribute), 141
group (telegram.ext.filters.Filters attribute), 14, 18
group_chat_created (telegram.Message attribute), 179
groups (telegram.ext.Dispatcher attribute), 10
groups (telegram.ext.filters.Filters attribute), 14
Index
InlineQueryResultCachedSticker (class in telegram), 236
InlineQueryResultCachedVideo (class in telegram), 236
InlineQueryResultCachedVoice (class in telegram), 238
InlineQueryResultContact (class in telegram), 239
InlineQueryResultDocument (class in telegram), 241
InlineQueryResultGame (class in telegram), 242
InlineQueryResultGif (class in telegram), 243
InlineQueryResultLocation (class in telegram), 245
InlineQueryResultMpeg4Gif (class in telegram), 247
InlineQueryResultPhoto (class in telegram), 249
InlineQueryResultVenue (class in telegram), 251
InlineQueryResultVideo (class in telegram), 253
InlineQueryResultVoice (class in telegram), 255
input_file_content (telegram.InputFile attribute), 165
input_message_content (telegram.InlineQueryResultDocument attribute), 242
input_message_content (telegram.InlineQueryResultGif attribute), 244
input_message_content (telegram.InlineQueryResultLocation attribute), 246
input_message_content (telegram.InlineQueryResultMpeg4Gif attribute), 248
input_message_content (telegram.InlineQueryResultPhoto attribute), 250
input_message_content (telegram.InlineQueryResultVenue attribute), 252
input_message_content (telegram.InlineQueryResultVideo attribute), 254
input_message_content (telegram.InlineQueryResultVoice attribute), 256
InputContactMessageContent (class in telegram), 259
InputFile (class in telegram), 165
InputLocationMessageContent (class in telegram), 257
InputMedia (class in telegram), 166
InputMediaAnimation (class in telegram), 166
InputMediaAudio (class in telegram), 168
InputMediaDocument (class in telegram), 169
InputMediaPhoto (class in telegram), 170
InputMediaVideo (class in telegram), 171
InputMessageContent (class in telegram), 256
InputTextMessageContent (class in telegram), 256
InputVenueMessageContent (class in telegram), 258
insert_bot () (telegram.ext.BasePersistence method), 72
internal_passport (telegram.SecureData attribute), 275
InvalidToken, 160
InvertedFilter (class in telegram.ext.filters), 26
invite_link (telegram.Chat attribute), 139
Invoice (class in telegram), 261
invoice (telegram.ext.filters.Filters attribute), 18
invoice (telegram.Message attribute), 179
invoice_payload (telegram.PreCheckoutQuery attribute), 267
invoice_payload (telegram.ShippingQuery attribute), 266
invoice_payload (telegram.SuccessfulPayment attribute), 264
ip_address (telegram.WebhookInfo attribute), 219
is_animated (telegram.Sticker attribute), 220
is_animated (telegram.StickerSet attribute), 222
is_anonymous (telegram.ChatMember attribute), 147
is_anonymous (telegram.Poll attribute), 198
is_bot (telegram.User attribute), 208
is_closed (telegram.Poll attribute), 198
is_image () (telegram.InputFile static method), 166
is_local_file() (in module telegram.utils.helpers), 285
is_member (telegram.ChatMember attribute), 148
ITALIC (telegram.MessageEntity attribute), 196

J
Job (class in telegram.ext), 27
job (telegram.ext.CallbackContext attribute), 36
job (telegram.ext.Job attribute), 27
job_queue (telegram.ext.CallbackContext attribute), 36
job_queue (telegram.ext.Dispatcher attribute), 8
job_queue (telegram.ext.Job attribute), 27
job_queue (telegram.ext.Updater attribute), 5
JobQueue (class in telegram.ext), 28
jobs () (telegram.ext.JobMethod method), 28
jpg (telegram.ext.filters.Filters attribute), 16
JSONDict (in module telegram.utils.types), 289

K
keyboard (telegram.ReplyKeyboardMarkup attribute), 202
KeyboardButton (class in telegram), 173
KeyboardButtonPollType (class in telegram), 173
kick_chat_member() (telegram.Bot method), 100
kick_member() (telegram.Chat method), 141
kickChatMember () (telegram.Bot method), 100
KICKED (telegram.ChatMember attribute), 149
kwargs (telegram.utils.promise.Promise attribute), 288

L
label (telegram.LabeledPrice attribute), 261
LabeledPrice (class in telegram), 261
language (telegram.MessageEntity attribute), 195
language_code (telegram.User attribute), 208
last_error_date (telegram.WebhookInfo attribute), 219
last_error_message (telegram.WebhookInfo attribute), 219
last_name (telegram.Bot attribute), 101
last_name (telegram.Chat attribute), 139
last_name (telegram.Contact attribute), 157
last_name (telegram.InlineQuery attribute), 157
last_name (telegram.InlineQueryResultContact attribute), 240
last_name (telegram.InputContactMessageContent attribute), 260
last_name (telegram.PersonalDetails attribute), 276
last_name (telegram.User attribute), 208
last_name_native (telegram.PersonalDetails attribute), 277
latitude (telegram.InlineQueryResultLocation attribute), 245
latitude (telegram.InlineQueryResultVenue attribute), 251
latitude (telegram.InputLocationMessageContent attribute), 257
latitude (telegram.InputVenueMessageContent attribute), 258
latitude (telegram.Location attribute), 174
leave () (telegram.Chat method), 142
leave_chat () (telegram.Bot method), 101
leaveChat () (telegram.Bot method), 101
LEFT (telegram.ChatMember attribute), 149
left_chat_member (telegram.ext.filters.Filters attribute), 21
left_chat_member (telegram.Message attribute), 178
length (telegram.MessageEntity attribute), 195
length (telegram.VideoNote attribute), 217
link (telegram.Bot attribute), 101
link (telegram.Chat attribute), 142
link (telegram.Message attribute), 186
link (telegram.User attribute), 209
linked_chat_id (telegram.Chat attribute), 140
live_period (telegram.InlineQueryResultLocation attribute), 246
live_period (telegram.InputLocationMessageContent attribute), 258
live_period (telegram.Location attribute), 174
Location (class in telegram), 174
location (telegram.Chat attribute), 140
location (telegram.ChatLocation attribute), 146
location (telegram.ChosenInlineResult attribute), 260
location (telegram.ext.filters.Filters attribute), 18
location (telegram.InlineQuery attribute), 224
location (telegram.Message attribute), 178
location (telegram.Venue attribute), 214
log_out () (telegram.Bot method), 101
login_url (telegram.InlineKeyboardButton attribute), 163
LoginUrl (class in telegram), 175
logOut () (telegram.Bot method), 101
longitude (telegram.InlineQueryResultLocation attribute), 245
longitude (telegram.InlineQueryResultVenue attribute), 251
longitude (telegram.InputLocationMessageContent attribute), 257
longitude (telegram.InputVenueMessageContent attribute), 258
longitude (telegram.Location attribute), 174
map_to_parent (telegram.ext.ConversationHandler

Index
NetworkError (telegram.PersonalDetails attribute), 277
migrate (telegram.ext.filters.Filters attribute), 21
migrate_from_chat_id (telegram.Message attribute), 179
migrate_to_chat_id (telegram.Message attribute), 179
mime_type (telegram.Animation attribute), 78
mime_type (telegram.Audio attribute), 79
mime_type (telegram.Document attribute), 159
mime_type (telegram.ext.filters.Filters attribute), 16
mime_type (telegram.InlineQueryResultDocument attribute), 241
mime_type (telegram.InlineQueryResultVideo attribute), 253
mime_type (telegram.Video attribute), 216
mime_type (telegram.Voice attribute), 218
MOUTH (telegram.MaskPosition attribute), 223
mp3 (telegram.ext.filters.Filters attribute), 17
mpeg4_duration (telegram.InlineQueryResultMpeg4Gif attribute), 247
mpeg4_file_id (telegram.InlineQueryResultCachedMpeg4Gif attribute), 233
mpeg4_height (telegram.InlineQueryResultMpeg4Gif attribute), 247
mpeg4_url (telegram.InlineQueryResultMpeg4Gif attribute), 247
mpeg4_width (telegram.InlineQueryResultMpeg4Gif attribute), 247

N
name (telegram.Bot attribute), 101
name (telegram.ext.ConversationHandler attribute), 45
name (telegram.ext.DelayQueue attribute), 33
name (telegram.ext.filters.BaseFilter attribute), 25
name (telegram.ext.filters.MessageFilter attribute), 25
name (telegram.ext.filters.UpdateFilter attribute), 26
name (telegram.ext.Job attribute), 27
name (telegram.OrderInfo attribute), 263
name (telegram.StickerSet attribute), 222
name (telegram.User attribute), 210
NetworkError, 160
new_chat_members (telegram.ext.filters.Filters attribute), 21
new_chat_members (telegram.Message attribute), 178
new_chat_photo (telegram.ext.filters.Filters attribute), 21
new_chat_photo (telegram.Message attribute), 179
new_chat_title (telegram.Message attribute), 21
new_chat_title (telegram.Message attribute), 179
next_t (telegram.ext.Job attribute), 28
nonce (telegram.Credentials attribute), 274

O
offset (telegram.InlineQuery attribute), 224
offset (telegram.MessageEntity attribute), 195
on_flush (telegram.ext.PicklePersistence attribute), 73
one_time_keyboard (telegram.ReplyKeyboardMarkup attribute), 203
open_period (telegram.Poll attribute), 199
option_ids (telegram.PollAnswer attribute), 200
options (telegram.Poll attribute), 198
order_info (telegram.PreCheckoutQuery attribute), 267
order_info (telegram.SuccessfulPayment attribute), 265
OrderInfo (class in telegram), 263

P
parse_caption_entities () (telegram.Message method), 186
parse_caption_entity () (telegram.Message method), 187
parse_entities () (telegram.Message method), 187
parse_entity () (telegram.Message method), 187
parse_explanation_entities () (telegram.Poll method), 199
parse_explanation_entity () (telegram.Poll method), 200
parse_file_input() (in module telegram.utils.helpers), 286
parse_mode (telegram.ext.Defaults attribute), 36
parse_mode (telegram.InlineQueryResultAudio attribute), 227
parse_mode (telegram.InlineQueryResultCachedAudio attribute), 229
parse_mode (telegram.InlineQueryResultCachedDocument attribute), 230
parse_mode (telegram.InlineQueryResultCachedGif attribute), 232
parse_mode (telegram.InlineQueryResultCachedMpeg4Gif attribute), 233
parse_mode (telegram.InlineQueryResultCachedPhoto attribute), 235
parse_mode (telegram.InlineQueryResultCachedVideo attribute), 237
parse_mode (telegram.InlineQueryResultDocument attribute), 238
parse_mode (telegram.InlineQueryResultDocument attribute), 241
parse_mode (telegram.InlineQueryResultGif attribute), 244
parse_mode (telegram.InlineQueryResultMpeg4Gif attribute), 248
parse_mode (telegram.InlineQueryResultPhoto attribute), 250
parse_mode (telegram.InlineQueryResultVideo attribute), 253
parse_mode (telegram.InlineQueryResultVoice attribute), 255
parse_mode (telegram.InputMediaAnimation attribute), 166
parse_mode (telegram.InputMediaAudio attribute), 168
parse_mode (telegram.InputMediaDocument attribute), 169
parse_mode (telegram.InputTextMessageContent attribute), 257
parse_text_entities() (telegram.Game method), 269
parse_text_entity() (telegram.Game method), 269
ParseMode (class in telegram), 196
PARSEMODE_HTML (in module telegram.constants), 156
PARSEMODE_MARKDOWN (in module telegram.constants), 156
PARSEMODE_MARKDOWN_V2 (in module telegram.constants), 156
pass_args (telegram.ext.CommandHandler attribute), 47
pass_args (telegram.ext.PrefixHandler attribute), 59
pass_args (telegram.ext.StringCommandHandler attribute), 66
pass_chat_data (telegram.ext.CallbackQueryHandler attribute), 40
pass_chat_data (telegram.ext.ChosenInlineResultHandler attribute), 43
pass_chat_data (telegram.ext.CommandHandler attribute), 48
pass_chat_data (telegram.ext.Handler attribute), 38
pass_chat_data (telegram.ext.InlineQueryHandler attribute), 50
pass_chat_data (telegram.ext.MessageHandler attribute), 52
pass_chat_data (telegram.ext.PollAnswerHandler attribute), 54
pass_chat_data (telegram.ext.PollHandler attribute), 56
pass_chat_data (telegram.ext.PollAnswerHandler attribute), 55
pass_chat_data (telegram.ext.PollHandler attribute), 56
pass_chat_data (telegram.ext.PollHandler attribute), 55
pass_chat_data (telegram.ext.PollHandler attribute), 54
pass_chat_data (telegram.ext.ShippingQueryHandler attribute), 64
pass_chat_data (telegram.ext.StringCommandHandler attribute), 66
pass_chat_data (telegram.ext.StringRegexHandler attribute), 68
pass_chat_data (telegram.ext.MessageHandler attribute), 62
pass_groupdict (telegram.ext.CallbackQueryHandler attribute), 40
pass_groupdict (telegram.ext.InlineQueryHandler attribute), 50
pass_groupdict (telegram.ext.RegexHandler attribute), 62
pass_groupdict (telegram.ext.StringRegexHandler attribute), 67
passGroups (class in telegram), 196
pass_groups (telegram.ext.CallbackQueryHandler attribute), 40
pass_groups (telegram.ext.InlineQueryHandler attribute), 50
pass_groups (telegram.ext.RegexHandler attribute), 62
pass_groups (telegram.ext.StringRegexHandler attribute), 67
pass_job_queue (telegram.ext.CallbackQueryHandler attribute), 40
pass_job_queue (telegram.ext.ChosenInlineResultHandler attribute), 42
pass_job_queue (telegram.ext.CommandHandler attribute), 47
pass_job_queue (telegram.ext.Handler attribute), 38
pass_job_queue (telegram.ext.InlineQueryHandler attribute), 50
pass_job_queue (telegram.ext.MessageHandler attribute), 52
pass_job_queue (telegram.ext.PollAnswerHandler attribute), 54
pass_job_queue (telegram.ext.PollHandler attribute), 56
pass_job_queue (telegram.ext.PollAnswerHandler attribute), 55
pass_job_queue (telegram.ext.PollHandler attribute), 54
pass_job_queue (telegram.ext.PollHandler attribute), 53
pass_job_queue (telegram.ext.ShippingQueryHandler attribute), 64
pass_job_queue (telegram.ext.StringCommandHandler attribute), 66
pass_job_queue (telegram.ext.StringRegexHandler attribute), 68
pass_job_queue (telegram.ext.TypeHandler attribute), 69
pass_update_queue (telegram.ext.CallbackQueryHandler attribute), 40
pass_update_queue (telegram.ext.CallbackQueryHandler attribute), 42
pass_update_queue (telegram.ext.CommandHandler attribute), 47
pass_update_queue (telegram.ext.CommandHandler attribute), 38
pass_update_queue (telegram.ext.InlineQueryHandler attribute), 50
pass_update_queue (telegram.ext.InlineQueryHandler attribute), 52
pass_update_queue (telegram.ext.InlineQueryHandler attribute), 54
pass_update_queue (telegram.ext.InlineQueryHandler attribute), 56
pass_update_queue (telegram.ext.InlineQueryHandler attribute), 57
pass_update_queue (telegram.ext.InlineQueryHandler attribute), 59
pass_update_queue (telegram.ext.InlineQueryHandler attribute), 62
pass_update_queue (telegram.ext.CommandHandler attribute), 64
pass_update_queue (telegram.ext.CommandHandler attribute), 66
pass_update_queue (telegram.ext.CommandHandler attribute), 68
pass_update_queue (telegram.ext.TypeHandler attribute), 69
pass_user_data (telegram.ext.CallbackQueryHandler attribute), 40
pass_user_data (telegram.ext.CallbackQueryHandler attribute), 43
pass_user_data (telegram.ext.CommandHandler attribute), 47
pass_user_data (telegram.ext.Handler attribute), 38
pass_user_data (telegram.ext.InlineQueryHandler attribute), 50
pass_user_data (telegram.ext.InlineQueryHandler attribute), 52
pass_user_data (telegram.ext.MessageHandler attribute), 55
pass_user_data (telegram.ext.PollAnswerHandler attribute), 56
pass_user_data (telegram.ext.PollHandler attribute), 56
pass_user_data (telegram.ext.PollHandler attribute), 57
pass_user_data (telegram.ext.PollHandler attribute), 59
pass_user_data (telegram.ext.PollHandler attribute), 62
pass_user_data (telegram.ext.PollHandler attribute), 64
pass_user_data (telegram.ext.PreCheckoutQueryHandler attribute), 58
pass_user_data (telegram.ext.PrefixHandler attribute), 60
pass_user_data (telegram.ext.RegexHandler attribute), 62
pass_user_data (telegram.ext.ShippingQueryHandler attribute), 64
passport (telegram.SecureData attribute), 274
passport_data (telegram.filters.Filters attribute), 18
passport_data (telegram.Message attribute), 180
passport_registration (telegram.SecureData attribute), 275
PersistData (class in telegram), 273
PassportDataError (class in telegram), 270
PassportDataError (class in telegram), 271
PassportDataErrorFile (class in telegram), 274
PassportDataErrorFiles (class in telegram), 272
PassportDataErrorFrontSide (class in telegram), 272
PassportDataErrorReverseSide (class in telegram), 271
PassportFile (class in telegram), 278
pattern (telegram.ext.CallbackQueryHandler attribute), 40
pattern (telegram.ext.InlineQueryHandler attribute), 50
pattern (telegram.ext.RegexHandler attribute), 62
pattern (telegram.ext.StringRegexHandler attribute), 67
day (telegram.InlineKeyboardButton attribute), 163
day (telegram.InlineKeyboardButton attribute), 163
pending_update_count (telegram.WebhookInfo attribute), 219
per_chat (telegram.ext.ConversationHandler attribute), 45
per_message (telegram.ext.ConversationHandler attribute), 45
per_user (telegram.ext.ConversationHandler attribute), 45
perform (telegram.Audio attribute), 79
perform (telegram.InlineQueryResultAudio attribute), 227
perform (telegram.InputMediaAudio attribute), 168
permissions (telegram.Chat attribute), 139
persistence (telegram.Dispatcher attribute), 9
persistence (telegram.Updater attribute), 6
persistent (telegram.ext.ConversationHandler attribute), 45
personal_details (telegram.SecureData attribute), 274
PersonalDetails (class in telegram), 276
phone_number (telegram.Contact attribute), 157
phone_number (telegram.EncryptedContact attribute), 280
phone_number (telegram.InlineQueryResultContact attribute), 239
phone_number (telegram.InputContactMessageContent attribute), 259
PHONE_NUMBER (telegram.MessageEntity attribute), 196
phone_number (telegram.OrderInfo attribute), 263
photo (telegram.Chat attribute), 139
photo (telegram.ext.filters.Filters attribute), 18
photo (telegram.Game attribute), 268
photo (telegram.Message attribute), 178
photo_file_id (telegram.InlineQueryResultCachedPhoto attribute), 234
photo_height (telegram.InlineQueryResultPhoto attribute), 249
photo_url (telegram.InlineQueryResultPhoto attribute), 249
photo_width (telegram.InlineQueryResultPhoto attribute), 249
photos (telegram.UserProfilePhotos attribute), 214
PhotoSize (class in telegram), 197
PicklePersistence (class in telegram.ext), 73
pin() (telegram.Message method), 188
pin_chat_message() (telegram.Bot method), 101
pin_message() (telegram.CallbackQuery method), 137
pin_message() (telegram.Chat method), 142
pin_message() (telegram.User method), 210
pinChatMessage() (telegram.Bot method), 101
pinned_message (telegram.Chat attribute), 139
pinned_message (telegram.ext.filters.Filters attribute), 21
pinned_message (telegram.Message attribute), 179
point (telegram.MaskPosition attribute), 222
Poll (class in telegram), 198
poll (telegram.ext.filters.Filters attribute), 18
poll (telegram.Message attribute), 180
poll (telegram.Update attribute), 207
poll_answer (telegram.Update attribute), 207
poll_id (telegram.PollAnswer attribute), 200
POLL QUIZ (in module telegram.constants), 157
POLL_REGULAR (in module telegram.constants), 157
PollAnswer (class in telegram), 200
PollAnswerHandler (class in telegram.ext), 54
PollHandler (class in telegram.ext), 56
PollOption (class in telegram), 201
pooled_function (telegram.utils.promise.Promise attribute), 287
position (telegram.GameHighScore attribute), 270
post() (telegram.utils.request.Request method), 289
post_code (telegram.ResidentialAddress attribute), 277
post_code (telegram.ShippingAddress attribute), 263
PRE (telegram.MessageEntity attribute), 196
pre_checkout_query (telegram.Update attribute), 206
PreCheckoutQuery (class in telegram), 266
PreCheckoutQueryHandler (class in telegram.ext), 57
prefix (telegram.ext.PrefixHandler attribute), 61
PrefixHandler (class in telegram.ext), 59
prices (telegram.ShippingOption attribute), 264
PRIVATE (telegram.Chat attribute), 141
private (telegram.ext.filters.Filters attribute), 14, 18
process_update() (telegram.ext.Dispatcher method), 10
Promise (class in telegram.utils.promise), 287
promote_chat_member() (telegram.Bot method), 102
promoteChatMember() (telegram.Bot method), 102
provider_payment_charge_id (telegram.SuccessfulPayment attribute), 265
proximity_alert_radius (telegram.InlineQueryResultLocation attribute), 246
proximity_alert_radius (telegram.InputLocationMessageContent attribute), 258
proximity_alert_radius (telegram.Location attribute), 174
proximity_alert_triggered (telegram.ext.filters.Filters attribute), 21
proximity_alert_triggered (telegram.Message attribute), 180
ProximityAlertTriggered (class in telegram), 201
py (telegram.ext.filters.Filters attribute), 17
query (telegram.ChosenInlineResult attribute), 260
query (telegram.InlineQuery attribute), 224
question (telegram.Poll attribute), 198
QUIZ (telegram.Poll attribute), 199
quote (telegram.ext.Defaults attribute), 37
Q

RECORD_AUDIO (telegram.ChatAction attribute), 146
RECORD_VIDEO (telegram.ChatAction attribute), 146
RECORD_VIDEO_NOTE (telegram.ChatAction attribute), 146
RegExHandler (class in telegram.ext), 62
REGULAR (telegram.Poll attribute), 199
remove_bot_ids() (telegram.ext.filters.Filters.via_bot method), 24
ReplyMarkup (class in telegram), 205
Request (class in telegram.utils.request), 288
request_contact (telegram.KeyboardButton attribute), 173
request_location (telegram.KeyboardButton attribute), 173
request_poll (telegram.KeyboardButton attribute), 173
request_write_access (telegram>LoginUrl attribute), 175
residence_country_code (telegram.PersonalDetails attribute), 276
ResidentialAddress (class in telegram), 277
resize_keyboard (telegram.ReplyKeyboardMarkup attribute), 203
restrict_chat_member () (telegram.Bot method), 103
restrictChatMember () (telegram.Bot method), 103
RESTRICTED (telegram.ChatMember attribute), 149
result () (telegram.utils.promise.Promise method), 288
result_id (telegram.ChosenInlineResult attribute), 260
retrieval (telegram.utils.request.Request method), 289
RetryAfter, 160
reverse_side (telegram.EncryptedPassportElement attribute), 280
run () (telegram.ext.DelayQueue method), 34
run () (telegram.ext.Job method), 28
run () (telegram.utils.promise.Promise method), 288
run_async (telegram.ext.CallbackQueryHandler attribute), 40
run_async (telegram.ext.ChosenInlineResultHandler attribute), 43
run_async (telegram.ext.CommandHandler attribute), 48
run_async (telegram.ext.Defaults attribute), 37
run_async (telegram.ext.Handler attribute), 38
run_async (telegram.ext.InlineQueryHandler attribute), 50
run_async (telegram.ext.MessageHandler attribute), 53
run_async (telegram.ext.PollAnswerHandler attribute), 55
run_async (telegram.ext.PollHandler attribute), 56
run_async (telegram.ext.PreCheckoutQueryHandler attribute), 58
run_async (telegram.ext.PrefixHandler attribute), 60
run_async (telegram.ext.RegexHandler attribute), 62
run_async (telegram.ext.ShippingQueryHandler attribute), 64
run_async (telegram.ext.StringCommandHandler
attribute), 66
run_async (telegram.ext.StringRegexHandler attribute), 68
run_async (telegram.ext.TypeHandler attribute), 69
run_async () (telegram.ext.Dispatcher method), 11
run_custom () (telegram.ext.JobQueue method), 28
run_daily () (telegram.ext.JobQueue method), 29
run_monthly () (telegram.ext.JobQueue method), 29
run_once () (telegram.ext.JobQueue method), 30
run_repeating () (telegram.ext.JobQueue method), 30
running (telegram.ext.Dispatcher attribute), 11
running (telegram.ext.Updater attribute), 6

S
scale (telegram.MaskPosition attribute), 223
schedule_remover () (telegram.ext.Job method), 28
scheduler (telegram.ext.JobQueue attribute), 28
score (telegram.GameHighScore attribute), 270
secret (telegram.DataCredentials attribute), 274
secret (telegram.EncryptedCredentials attribute), 282
secret (telegram.FileCredentials attribute), 275
secure_data (telegram.Credentials attribute), 274
SecureData (class in telegram), 274
selective (telegram.ForceReply attribute), 162
selective (telegram.ReplyKeyboardMarkup attribute), 203
selective (telegram.ReplyKeyboardRemove attribute), 202
selfie (telegram.EncryptedPassportElement attribute), 281
send_action () (telegram.Chat method), 142
send_action () (telegram.User method), 210
send_animated_image () (telegram.Bot method), 106
send_animated_image () (telegram.Chat method), 142
send_animated_image () (telegram.User method), 210
send_audio () (telegram.Bot method), 108
send_audio () (telegram.Chat method), 142
send_audio () (telegram.User method), 210
send_chat_action () (telegram.Bot method), 109
send_chat_action () (telegram.Chat method), 142
send_chat_action () (telegram.User method), 210
send_contact () (telegram.Bot method), 109
send_contact () (telegram.Chat method), 142
send_contact () (telegram.User method), 211
send_copy () (telegram.Chat method), 143
send_copy () (telegram.User method), 211
send_dice () (telegram.Bot method), 110
send_dice () (telegram.Chat method), 143
send_dice () (telegram.User method), 211
send_document () (telegram.Bot method), 111
send_document () (telegram.Chat method), 143
send_document () (telegram.User method), 211
send_game() (telegram.Bot method), 112
send_game() (telegram.Chat method), 143
send_invoice() (telegram.User method), 211
send_invoice() (telegram.Chat method), 143
send_invoice() (telegram.User method), 211
send_location() (telegram.Bot method), 114
send_location() (telegram.User method), 143
send_media_group() (telegram.Bot method), 115
send_media_group() (telegram.Chat method), 143
send_media_group() (telegram.User method), 212
send_message() (telegram.Bot method), 115
send_message() (telegram.Chat method), 144
send_message() (telegram.User method), 212
send_photo() (telegram.Bot method), 116
send_photo() (telegram.Chat method), 144
send_photo() (telegram.User method), 212
send_poll() (telegram.Bot method), 117
send_poll() (telegram.Chat method), 144
send_poll() (telegram.User method), 212
send_sticker() (telegram.Bot method), 118
send_sticker() (telegram.Chat method), 144
send_sticker() (telegram.User method), 212
send_venue() (telegram.Bot method), 119
send_venue() (telegram.Chat method), 144
send_venue() (telegram.User method), 212
send_video() (telegram.Bot method), 120
send_video() (telegram.Chat method), 144
send_video() (telegram.User method), 213
send_video_note() (telegram.Bot method), 122
send_video_note() (telegram.Chat method), 145
send_video_note() (telegram.User method), 213
send_video_note() (telegram.Bot method), 122
send_video() (telegram.Bot method), 122
send_voice() (telegram.Bot method), 145
send_voice() (telegram.Chat method), 145
send_voice() (telegram.User method), 213
sendAnimation() (telegram.Bot method), 104
sendAudio() (telegram.Bot method), 104
sendChatAction() (telegram.Bot method), 104
sendContact() (telegram.Bot method), 104
sendDice() (telegram.Bot method), 104
sendDocument() (telegram.Bot method), 104
sender_chat (telegram.Message attribute), 176
sendGame() (telegram.Bot method), 104
sendInvoice() (telegram.Bot method), 105
sendMediaGroup() (telegram.Bot method), 105
sendMessage() (telegram.Bot method), 105
sendMediaGroup() (telegram.Bot method), 105
sendPhoto() (telegram.Bot method), 105
sendPhoto() (telegram.Bot method), 105
sendPoll() (telegram.Bot method), 105
sendSticker() (telegram.Bot method), 106
sendVenue() (telegram.Bot method), 106
sendVideo() (telegram.Bot method), 106
sendVideoNote() (telegram.Bot method), 106
sendVoice() (telegram.Bot method), 106
SERVICE_CHAT_ID (in module telegram.constants), 153
set_administrator_custom_title() (telegram.Chat method), 145
set_bot() (telegram.ext.BasePersistence method), 72
set_chat_administrator_custom_title() (telegram.Bot method), 124
set_chat_description() (telegram.Bot method), 125
set_chat_permissions() (telegram.Bot method), 125
set_chat_photo() (telegram.Bot method), 125
set_chat_sticker_set() (telegram.Bot method), 126
set_chat_title() (telegram.Bot method), 126
set_dispatcher() (telegram.ext.JobQueue method), 31
set_game_score() (telegram.Bot method), 127
set_game_score() (telegram.CallbackQuery method), 137
set_game_score() (telegram.Message method), 192
set_my_commands() (telegram.Bot method), 127
set_name (telegram.Sticker attribute), 221
set_passport_data_errors() (telegram.Bot method), 127
set_permissions() (telegram.Chat method), 145
set_sticker_position_in_set() (telegram.Bot method), 128
set_sticker_set_thumb() (telegram.Bot method), 128
set_webhook() (telegram.Bot method), 129
setChatAdministratorCustomTitle() (telegram.Bot method), 123
setChatDescription() (telegram.Bot method), 124
setChatPermissions() (telegram.Bot method), 124
setChatPhoto() (telegram.Bot method), 124
setChatStickerSet() (telegram.Bot method), 124
setChatTitle() (telegram.Bot method), 124
setGameScore() (telegram.Bot method), 124
setMyCommands() (telegram.Bot method), 124
setPassportDataErrors() (telegram.Bot method), 124
setStickerPositionInSet() (telegram.Bot method), 124
setStickerSetThumb() (telegram.Bot method), 124
setWebhook() (telegram.Bot method), 124
shipping_address (telegram.OrderInfo attribute), 263
shipping_address (telegram.ShippingQuery attribute), 266
shipping_option_id (telegram.PreCheckoutQuery attribute), 267
shipping_option_id (telegram.SuccessfulPayment attribute), 265
shipping_query (telegram.Update attribute), 206
ShippingAddress (class in telegram), 262
ShippingOption (class in telegram), 264
ShippingQuery (class in telegram), 265
ShippingQueryHandler (class in telegram.ext), 64
single_file (telegram.ext.PicklePersistence attribute), 73
SLOT_MACHINE (telegram.Dice attribute), 159
slot_machine (telegram.ext.filters.Filters attribute), 15
slow_mode_delay (telegram.Chat attribute), 139
SLT (in module telegram.utils.types), 289
small_file_id (telegram.ChatPhoto attribute), 151
small_file_unique_id (telegram.ChatPhoto attribute), 151
source (telegram.PassportElementError attribute), 270
start() (telegram.ext.Dispatcher method), 11
start() (telegram.ext.JobQueue method), 32
start() (telegram.ext.MessageQueue method), 33
start_polling() (telegram.ext.Updater method), 7
start_webhook() (telegram.ext.Updater method), 7
state (telegram.ext.DispatcherHandlerStop attribute), 12
state (telegram.ResidentialAddress attribute), 277
date (telegram.ShippingAddress attribute), 262
states (telegram.ext.ConversationHandler attribute), 44
status (telegram.ChatMember attribute), 147
status_update (telegram.ext.filters.Filters attribute), 20
Sticker (class in telegram), 220
sticker (telegram.ext.filters.Filters attribute), 21
sticker (telegram.Message attribute), 178
STICKER_CENTRE (in module telegram.constants), 157
STICKER_EYES (in module telegram.constants), 157
sticker_file_id (telegram.InlineQueryResultCachedSticker attribute), 236
STICKER_FOREHEAD (in module telegram.constants), 157
STICKER_MOUTH (in module telegram.constants), 157
sticker_set_name (telegram.Chat attribute), 139
stickers (telegram.StickerSet attribute), 222
StickerSet (class in telegram), 222
stop() (telegram.DelayQueue method), 34
stop() (telegram.Dispatcher method), 11
stop() (telegram.JobQueue method), 32
stop() (telegram.MessageQueue method), 33
stop() (telegram.Updater method), 8
stop_live_location() (telegram.Message method), 192
stop_message_live_location() (telegram.Bot method), 130
stop_message_live_location() (telegram.CallbackQuery method), 138
stop_poll() (telegram.Bot method), 131
stop_poll() (telegram.Message method), 193
stopMessageLiveLocation() (telegram.Bot method), 130
stopPoll() (telegram.Bot method), 130
store_bot_data (telegram.ext.BasePersistence attribute), 71
store_bot_data (telegram.ext.DictPersistence attribute), 75
store_bot_data (telegram.ext.PicklePersistence attribute), 73
store_chat_data (telegram.ext.BasePersistence attribute), 71
store_chat_data (telegram.ext.DictPersistence attribute), 75
store_chat_data (telegram.ext.PicklePersistence attribute), 73
store_user_data (telegram.ext.BasePersistence attribute), 71
store_user_data (telegram.ext.DictPersistence attribute), 75
store_user_data (telegram.ext.PicklePersistence attribute), 73
street_line1 (telegram.ResidentialAddress attribute), 277
street_line1 (telegram.ShippingAddress attribute), 262
street_line2 (telegram.ResidentialAddress attribute), 277
street_line2 (telegram.ShippingAddress attribute), 263
strict (telegram.ext.TypeHandler attribute), 69
STRIKETHROUGH (telegram.MessageEntity attribute), 196
StringCommandHandler (class in telegram.ext), 65
StringRegexHandler (class in telegram.ext), 67
successful_payment (telegram.ext.filters.Filters attribute), 21
successful_payment (telegram.Message attribute), 179
SuccessfulPayment (class in telegram), 264
super_group (telegram.ext.filters.Filters.sender_chat attribute), 20
SUPERGROUP (telegram.Chat attribute), 141
supergroup (telegram.ext.filters.Filters attribute), 14
supergroup_chat_created (telegram.Message attribute), 179
SUPPORTED_WEBHOOK_PORTS (in module telegram.constants), 152
supports_inline_queries (telegram.Bot attribute), 131
supports_inline_queries (telegram.User attribute), 209
supports_streaming (telegram.InputMediaVideo attribute), 172
svg (telegram.ext.filters.Filters attribute), 17
switch_inline_query (telegram.InlineKeyboardButton attribute), 163
switch_inline_query_current_chat (telegram.InlineKeyboardButton attribute), 163

T
targz (telegram.ext.filters.Filters attribute), 17
telegram.constants (module), 152
telegram.error (module), 160
telegram.ext.filters (module), 12
telegram.utils.helpers (module), 283
telegram.utils.types (module), 289
telegram_payment_charge_id (telegram.SuccessfulPayment attribute), 265
TelegramError, 160
TelegramObject (class in telegram), 205
temporary_registration (telegram.SecureData attribute), 275
text (telegram.ext.filters.Filters attribute), 16, 21
text (telegram.Game attribute), 268
text (telegram.InlineKeyboardButton attribute), 163
text (telegram.Message attribute), 173
text (telegram.Message attribute), 177
text (telegram.PollOption attribute), 201
text_entities (telegram.Game attribute), 268
text_html (telegram.Message attribute), 193
text_html_urled (telegram.Message attribute), 193
TEXT_LINK (telegram.MessageEntity attribute), 196
text_markdown (telegram.Message attribute), 193
text_markdown_urled (telegram.Message attribute), 194
text_markdown_v2 (telegram.Message attribute), 194
text_markdown_v2_urled (telegram.Message attribute), 194
TEXT_MENTION (telegram.MessageEntity attribute), 196
thumb (telegram.Animation attribute), 78
thumb (telegram.Audio attribute), 79
thumb (telegram.Document attribute), 159
thumb (telegram.InputMediaAnimation attribute), 167
thumb (telegram.InputMediaAudio attribute), 168
thumb (telegram.InputMediaDocument attribute), 170
thumb (telegram.InputMediaVideo attribute), 172
thumb (telegram.Sticker attribute), 220
thumb (telegram.StickerSet attribute), 222
thumb (telegram.Video attribute), 215
thumb (telegram.VideoNote attribute), 217

thumb_height (telegram.InlineQueryResultArticle attribute), 226
thumb_height (telegram.InlineQueryResultContact attribute), 240
thumb_height (telegram.InlineQueryResultDocument attribute), 242
thumb_height (telegram.InlineQueryResultLocation attribute), 246
thumb_height (telegram.InlineQueryResultVenue attribute), 252
thumb_mime_type (telegram.InlineQueryResultGif attribute), 244
thumb_mime_type (telegram.InlineQueryResultMpeg4Gif attribute), 247
thumb_url (telegram.InlineQueryResultArticle attribute), 226
thumb_url (telegram.InlineQueryResultContact attribute), 240
thumb_url (telegram.InlineQueryResultDocument attribute), 242
thumb_url (telegram.InlineQueryResultGif attribute), 244
thumb_url (telegram.InlineQueryResultLocation attribute), 246
thumb_url (telegram.InlineQueryResultMpeg4Gif attribute), 247
thumb_url (telegram.InlineQueryResultPhoto attribute), 249
thumb_url (telegram.InlineQueryResultVenue attribute), 252
thumb_url (telegram.InlineQueryResultVideo attribute), 253
thumb_width (telegram.InlineQueryResultArticle attribute), 226
thumb_width (telegram.InlineQueryResultContact attribute), 240
thumb_width (telegram.InlineQueryResultDocument attribute), 242
thumb_width (telegram.InlineQueryResultLocation attribute), 246
thumb_width (telegram.InlineQueryResultVenue attribute), 252
time_limit (telegram.ext.DelayQueue attribute), 33
TimedOut, 160
TIMEOUT (telegram.ext.ConversationHandler attribute), 46
timeout (telegram.ext.Defaults attribute), 37
title (telegram.Audio attribute), 79
title (telegram.Chat attribute), 139
title (telegram.Game attribute), 268
title (telegram.InlineQueryResultArticle attribute), 226
title (telegram.InlineQueryResultAudio attribute), 227
title (telegram.InlineQueryResultCachedDocument attribute), 230
title (telegram.InlineQueryResultCachedGif attribute), 232
title (telegram.InlineQueryResultCachedMpeg4Gif attribute), 233
title (telegram.InlineQueryResultCachedPhoto attribute), 234
title (telegram.InlineQueryResultCachedVideo attribute), 237
title (telegram.InlineQueryResultCachedVoice attribute), 238
title (telegram.InlineQueryResultDocument attribute), 241
title (telegram.InlineQueryResultGif attribute), 244
title (telegram.InlineQueryResultLocation attribute), 245
title (telegram.InlineQueryResultMpeg4Gif attribute), 248
title (telegram.InlineQueryResultPhoto attribute), 249
title (telegram.InlineQueryResultVenue attribute), 251
title (telegram.InlineQueryResultVideo attribute), 253
title (telegram.InlineQueryResultVoice attribute), 255
title (telegram.InputMediaAudio attribute), 168
title (telegram.InputVenueMessageContent attribute), 259
title (telegram.Invoice attribute), 261
title (telegram.ShippingOption attribute), 264
title (telegram.StickerSet attribute), 222
title (telegram.Venue attribute), 214
to_float_timestamp() (in module telegram.util.helpers), 286
to_json() (telegram.TelegramObject method), 205
to_timestamp() (in module telegram.util.helpers), 287
total_amount (telegram.Invoice attribute), 262
total_amount (telegram.PreCheckoutQuery attribute), 267
total_amount (telegram.SuccessfulPayment attribute), 264
total_count (telegram.UserProfilePhotos attribute), 214
total_voter_count (telegram.Poll attribute), 198
translation (telegram.EncryptedPassportElementType attribute), 281
traveler (telegram.ProximityAlertTriggered attribute), 201
txt (telegram.filters.Filters attribute), 17
type (telegram.Chat attribute), 138
type (telegram.EncryptedPassportElementType attribute), 280
type (telegram.ext.TypeHandler attribute), 69
type (telegram.InlineQueryResult attribute), 225
type (telegram.InlineQueryResultArticle attribute), 225
type (telegram.InlineQueryResultAudio attribute), 227
type (telegram.InlineQueryResultCachedAudio attribute), 228
type (telegram.InlineQueryResultCachedDocument attribute), 230
type (telegram.InlineQueryResultCachedGif attribute), 231
type (telegram.InlineQueryResultCachedMpeg4Gif attribute), 233
type (telegram.InlineQueryResultCachedPhoto attribute), 234
type (telegram.InlineQueryResultCachedSticker attribute), 236
type (telegram.InlineQueryResultCachedVideo attribute), 237
type (telegram.InlineQueryResultCachedVoice attribute), 238
type (telegram.InlineQueryResultContact attribute), 238
type (telegram.InlineQueryResultDocument attribute), 239
type (telegram.InlineQueryResultGame attribute), 241
type (telegram.InlineQueryResultGif attribute), 243
type (telegram.InlineQueryResultLocation attribute), 245
type (telegram.InlineQueryResultMpeg4Gif attribute), 247
type (telegram.InlineQueryResultPhoto attribute), 249
type (telegram.InlineQueryResultVenue attribute), 251
type (telegram.InlineQueryResultVideo attribute), 253
type (telegram.InlineQueryResultVoice attribute), 255
type (telegram.InputMediaAnimation attribute), 166
type (telegram.InputMediaAudio attribute), 168
type (telegram.InputMediaDocument attribute), 169
type (telegram.InputMediaPhoto attribute), 170
type (telegram.InputMediaVideo attribute), 171
type (telegram.KeyboardButtonPollType attribute), 174
type (telegram.MessageEntity attribute), 195
type (telegram.PassportElementError attribute), 270
type (telegram.PassportElementErrorDataField attribute), 273
type (telegram.PassportElementErrorFile attribute), 271
type (telegram.PassportElementErrorFiles attribute), 272
type (telegram.PassportElementErrorFrontSide attribute), 272
type (telegram.PassportElementErrorReverseSide attribute), 271
type (telegram.Poll attribute), 198
TypeHandler (class in telegram.ext), 69
update_bot_data() (telegram.Bot method), 72
update_bot_data() (telegram.ext.DictPersistence method), 73
update_bot_data() (telegram.ext.PicklePersistence method), 77
update_user_data() (telegram.ext.BasePersistence method), 73
update_user_data() (telegram.ext.DictPersistence method), 77
update_user_data() (telegram.ext.PicklePersistence method), 77
UpdateFilter (class in telegram.ext.filters), 25
Updater (class in telegram.ext), 5
UPLOAD_AUDIO (telegram.ChatAction attribute), 146
UPLOAD_DOCUMENT (telegram.ChatAction attribute), 146
UPLOAD_PHOTO (telegram.ChatAction attribute), 146
UPLOAD sticker_file() (telegram.Bot method), 133
UPLOAD_VIDEO (telegram.ChatAction attribute), 146
UPLOAD_VIDEO_NOTE (telegram.ChatAction attribute), 146
uploadStickerFile() (telegram.Bot method), 132
user (class in telegram), 208
user (telegram.ChatMember attribute), 147
user (telegram.GameHighScore attribute), 270
user (telegram.MessageEntity attribute), 195
user (telegram.PollAnswer attribute), 200
user_data (telegram.ext.CallbackContext attribute), 35
user_data (telegram.ext.DictPersistence attribute), 77
user_data (telegram.ext.Dispatcher attribute), 9
user_data_json (telegram.ext.DictPersistence attribute), 77
user_id (telegram.Contact attribute), 157
user_ids (telegram.ext.filters.Filters.user attribute), 22
user_sig_handler (telegram.ext.Updater attribute), 5
username (telegram.Bot attribute), 133
username (telegram.Chat attribute), 139
username (telegram.User attribute), 208
usernames (telegram.ext.filters.Filters.chat attribute), 13
usernames (telegram.ext.filters.Filters.sender_chat attribute), 19
usernames (telegram.ext.filters.Filters.user attribute), 22
usernames (telegram.ext.filters.Filters.via_bot attribute), 23
UserProfilePhotos (class in telegram), 213
utility_bill (telegram.SecureData attribute), 275
V
value (telegram.Dice attribute), 158 
value (telegram.utils.helpers.DefaultValue attribute), 283 
vcard (telegram.Contact attribute), 158 
vcard (telegram.InlineQueryResultContact attribute), 240 
vcard (telegram.InputContactMessageContent attribute), 260 
Venue (class in telegram), 214 
venue (telegram.ext.filters.Filters attribute), 23 
venue (telegram.Message attribute), 178 
via_bot (telegram.Message attribute), 180 
Video (class in telegram), 215 
video (telegram.ext.filters.Filters attribute), 16, 24 
video (telegram.Message attribute), 178 
video_duration (telegram.InlineQueryResultVideo attribute), 254 
video_file_id (telegram.InlineQueryResultCachedVideo attribute), 237 
video_height (telegram.InlineQueryResultVideo attribute), 254 
video_note (telegram.ext.filters.Filters attribute), 24 
video_note (telegram.Message attribute), 178 
video_url (telegram.InlineQueryResultVideo attribute), 253 
video_width (telegram.InlineQueryResultVideo attribute), 254 
VideoNote (class in telegram), 216 
Voice (class in telegram), 218 
voice (telegram.ext.filters.Filters attribute), 24 
voice (telegram.Message attribute), 178 
voice_duration (telegram.InlineQueryResultVoice attribute), 255 
voice_file_id (telegram.InlineQueryResultCachedVoice attribute), 238 
voice_url (telegram.InlineQueryResultVoice attribute), 255 
voter_count (telegram.PollOption attribute), 201 

W
WAITING (telegram.ext.ConversationHandler attribute), 46 
watcher (telegram.ProximityAlertTriggered attribute), 201 
wav (telegram.ext.filters.Filters attribute), 17 
WebhookInfo (class in telegram), 219 
width (telegram.Animation attribute), 78 
width (telegram.InputMediaAnimation attribute), 167 
width (telegram.InputMediaVideo attribute), 172 
width (telegram.PhotoSize attribute), 197 
width (telegram.Sticker attribute), 220 
width (telegram.Video attribute), 215 

workers (telegram.ext.Dispatcher attribute), 8 

X
x_shift (telegram.MaskPosition attribute), 223 
xml (telegram.ext.filters.Filters attribute), 17 
XORFilter (class in telegram.ext.filters), 26 

Y
y_shift (telegram.MaskPosition attribute), 223 

Z
zip (telegram.ext.filters.Filters attribute), 17