Package 'telegram.bot'

September 17, 2025

Type Package

Title Develop a 'Telegram Bot' with R

Version 3.0.1

Description Provides a pure interface for the 'Telegram Bot API' <http://core.telegram.org/bots/api>. In addition to the pure API implementation, it features a number of tools to make the development of 'Telegram' bots with R easy and straightforward, providing an easy-to-use interface that takes some work off the programmer.

URL https://github.com/ebeneditos/telegram.bot

BugReports https://github.com/ebeneditos/telegram.bot/issues

Depends R (>= 3.1.0)

Imports curl, httpuv, httr, jsonlite, openssl, R6

Suggests covr, devtools, knitr, promises, rmarkdown, testthat

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.3

VignetteBuilder knitr

NeedsCompilation no

Author Ernest Benedito [aut, cre],
Chris Stefano [ctb]

Maintainer Ernest Benedito <ebeneditos@gmail.com>

Repository CRAN

Date/Publication 2025-09-17 05:40:02 UTC

Contents

`+.TelegramObject` *Constructing an Updater*

Description

With + you can add any kind of `Handler` to an `Updater`'s `Dispatcher` (or directly to a `Dispatcher`).

Usage

```
## S3 method for class 'TelegramObject'
e1 + e2
```

Arguments

`e1`	An object of class `Updater` or `Dispatcher`.
`e2`	An object of class `Handler`.

Details

See `add_handler` for further information.

Examples

```
## Not run:
# You can chain multiple handlers
start <- function(bot, update) {
  bot$sendMessage(
    chat_id = update$message$chat_id,
    text = sprintf(
      "Hello %s!",
      update$message$from$first_name
```

```
    )
  )
}
echo <- function(bot, update) {
  bot$sendMessage(
    chat_id = update$message$chat_id,
    text = update$message$text
  )
}

updater <- Updater("TOKEN") + CommandHandler("start", start) +
  MessageHandler(echo, MessageFilters$text)

# And keep adding...
caps <- function(bot, update, args) {
  if (length(args > 0L)) {
    text_caps <- toupper(paste(args, collapse = " "))
    bot$sendMessage(
      chat_id = update$message$chat_id,
      text = text_caps
    )
  }
}

updater <- updater + CommandHandler("caps", caps, pass_args = TRUE)

# Give it a try!
updater$start_polling()
# Send '/start' to the bot, '/caps foo' or just a simple text

## End(Not run)
```

`add_error_handler` *Add an error handler*

Description

Registers an error handler in the `Dispatcher`.

Usage

```
add_error_handler(callback)
```

Arguments

`callback`	A function that takes `(bot, error)` as arguments.

Details

You can also use `add_handler` to register error handlers if the handler is of type `ErrorHandler`.

Examples

```
## Not run:
updater <- Updater(token = "TOKEN")

# Create error callback
error_callback <- function(bot, error) {
  warning(simpleWarning(conditionMessage(error), call = "Updates polling"))
}

# Register it to the updater's dispatcher
updater$dispatcher$add_error_handler(error_callback)
# or
updater$dispatcher$add_handler(ErrorHandler(error_callback))
# or
updater <- updater + ErrorHandler(error_callback)

## End(Not run)
```

---

add_handler *Add a handler*

Description

Register a handler. A handler must be an instance of a subclass of Handler. All handlers are organized in groups with a numeric value. The default group is 1. All groups will be evaluated for handling an update, but only 0 or 1 handler per group will be used.

Usage

```
add_handler(handler, group = 1L)
```

Arguments

handler	A Handler instance.
group	The group identifier, must be higher or equal to 1. Default is 1.

Details

You can use the add (+) operator instead.

The priority/order of handlers is determined as follows:

1. Priority of the group (lower group number = higher priority)
2. The first handler in a group which should handle an update will be used. Other handlers from the group will not be used. The order in which handlers were added to the group defines the priority (the first handler added in a group has the highest priority).

answerCallbackQuery *Send answers to callback queries*

Description

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. On success, TRUE is returned.

Usage

```
answerCallbackQuery(
  callback_query_id,
  text = NULL,
  show_alert = FALSE,
  url = NULL,
  cache_time = NULL
)
```

Arguments

callback_query_id	Unique identifier for the query to be answered.
text	(Optional). Text of the notification. If not specified, nothing will be shown to the user, 0-200 characters.
show_alert	(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	(Optional). URL that will be opened by the user's client.
cache_time	(Optional). The maximum amount of time in seconds that the result of the callback query may be cached client-side. Telegram apps will support caching starting in version 3.14. Defaults to 0.

Details

You can also use it's snake_case equivalent answer_callback_query.

answerInlineQuery *Send answers to an inline query*

Description

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

Usage

```
answerInlineQuery(
  inline_query_id,
  results,
  cache_time = 300L,
  is_personal = NULL,
  next_offset = NULL,
  switch_pm_text = NULL,
  switch_pm_parameter = NULL
)
```

Arguments

inline_query_id	Unique identifier for the answered query.
results	A list of InlineQueryResult for the inline query.
cache_time	(Optional). The maximum amount of time in seconds that the result of the inline query may be cached on the server.
is_personal	(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	(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	(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	(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. *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 auth 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.

Details

To enable this option, send the /setinline command to @BotFather and provide the placeholder text that the user will see in the input field after typing your bot's name.

You can also use it's snake_case equivalent answer_inline_query.

`BaseFilter` *The base of all filters*

Description

Base class for all Message Filters.

Usage

```
BaseFilter(filter)

as.BaseFilter(x, ...)

is.BaseFilter(x)
```

Arguments

`filter`	If you want to create your own filters you can call this generator passing by a `filter` function that takes a `message` as input and returns a boolean: `TRUE` if the message should be handled, `FALSE` otherwise.
`x`	Object to be coerced or tested.
`...`	Further arguments passed to or from other methods.

Details

See `filtersLogic` to know more about combining filter functions.

Examples

```
## Not run:
# Create a filter function
text_or_command <- function(message) !is.null(message$text)

# Make it an instance of BaseFilter with its generator:
text_or_command <- BaseFilter(filter = text_or_command)

# Or by coercing it with as.BaseFilter:
text_or_command <- as.BaseFilter(function(message) !is.null(message$text))

## End(Not run)
```

Bot *Creating a Bot*

Description

This object represents a Telegram Bot.

Usage

```
Bot(token, base_url = NULL, base_file_url = NULL, request_config = NULL)

is.Bot(x)
```

Arguments

token	The bot's token given by the *BotFather*.
base_url	(Optional). Telegram Bot API service URL.
base_file_url	(Optional). Telegram Bot API file URL.
request_config	(Optional). Additional configuration settings to be passed to the bot's POST requests. See the config parameter from ?httr::POST for further details. The request_config settings are very useful for the advanced users who would like to control the default timeouts and/or control the proxy used for HTTP communication.
x	Object to be tested.

Format

An R6Class object.

Details

To take full advantage of this library take a look at Updater.

You can also use its methods snake_case equivalent.

API Methods

answerCallbackQuery Send answers to callback queries

answerInlineQuery Send answers to an inline query

deleteMessage Delete a message

deleteWebhook Remove webhook integration

editMessageText Edit a text message

editMessageCaption Edit a caption

editMessageReplyMarkup Edit the reply markup of a message

forwardMessage Forward messages of any kind

getFile Prepare a file for downloading

getMe Check your bot's information

getUpdates Receive incoming updates

getUserProfilePhotos Get a user's profile photos

getWebhookInfo Get current webhook status

leaveChat Leave a chat

sendAnimation Send animation files

sendAudio Send audio files

sendChatAction Send a chat action

sendDocument Send general files

sendLocation Send point on the map

sendMessage Send text messages

sendPhoto Send image files

sendSticker Send a sticker

sendVideo Send a video

sendVideoNote Send video messages

sendVoice Send voice files

setWebhook Set a webhook

Other Methods

clean_updates Clean any pending updates

set_token Change your bot's auth token

Examples

```
## Not run:
bot <- Bot(token = "TOKEN")

# In case you want to set a proxy (see ?httr:use_proxy)
bot <- Bot(
  token = "TOKEN",
  request_config = httr::use_proxy(...)
)

## End(Not run)
```

bot_token *Get a token from environment*

Description

Obtain token from system variables (in `.Renviron`) set according to the naming convention `R_TELEGRAM_BOT_X` where `X` is the bot's name.

Usage

```
bot_token(bot_name)
```

Arguments

`bot_name`	The bot's name.

Examples

```
## Not run:
# Open the `.Renviron` file
file.edit(path.expand(file.path("~", ".Renviron")))
# Add the line (uncomment and replace <bot-token> by your bot TOKEN):
# R_TELEGRAM_BOT_RTelegramBot=<bot-token>
# Save and restart R

bot_token("RTelegramBot")

## End(Not run)
```

CallbackQueryHandler *Handling callback queries*

Description

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

Usage

```
CallbackQueryHandler(callback, pattern = NULL)
```

Arguments

`callback`	The callback function for this handler. See `Handler` for information about this function.
`pattern`	(Optional). Regex pattern to test.

Format

An R6Class object.

`check_update` *Check an update*

Description

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

Usage

```
check_update(update)
```

Arguments

`update`	The update to be tested.

`clean_updates` *Clean any pending updates*

Description

Use this method to clean any pending updates on Telegram servers. Requires no parameters.

Usage

```
clean_updates()
```

CommandHandler *Handling commands*

Description

Handler class to handle Telegram commands.

Usage

```
CommandHandler(
  command,
  callback,
  filters = NULL,
  pass_args = FALSE,
  username = NULL
)
```

Arguments

command	The command or vector of commands this handler should listen for.
callback	The callback function for this handler. See Handler for information about this function.
filters	(Optional). Only allow updates with these filters. See MessageFilters for a full list of all available filters.
pass_args	(Optional). Determines whether the handler should be passed args, received as a vector, split on spaces.
username	(Optional). Bot's username, you can retrieve it from bot$getMe()$username. If this parameter is passed, then the CommandHandler will also listen to the command /command@username, as bot commands are often called this way.

Format

An R6Class object.

Examples

```
## Not run:

# Initialize bot
bot <- Bot("TOKEN")
username <- bot$getMe()$username
updater <- Updater(bot = bot)

# Add a command
start <- function(bot, update) {
  bot$sendMessage(
    chat_id = update$message$chat_id,
```

```
    text = "Hi, I am a bot!"
  )
}

updater <- updater + CommandHandler("start", start, username = username)

## End(Not run)
```

`deleteMessage` *Delete a message*

Description

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

Usage

```
deleteMessage(chat_id, message_id)
```

Arguments

`chat_id`	Unique identifier for the target chat or username of the target channel.
`message_id`	Identifier of the message to delete.

Details

You can also use it's snake_case equivalent `delete_message`.

`deleteWebhook` *Remove webhook integration*

Description

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

Usage

```
deleteWebhook()
```

Details

You can also use it's snake_case equivalent `delete_webhook`.

Dispatcher *The dispatcher of all updates*

Description

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

Usage

```
Dispatcher(bot)

is.Dispatcher(x)
```

Arguments

bot	The bot object that should be passed to the handlers.
x	Object to be tested.

Format

An R6Class object.

Methods

add_handler Registers a handler in the Dispatcher.

add_error_handler Registers an error handler in the Dispatcher.

editMessageCaption *Edit a caption*

Description

Use this method to edit captions of messages.

Usage

```
editMessageCaption(
  chat_id = NULL,
  message_id = NULL,
  inline_message_id = NULL,
  caption = NULL,
  parse_mode = NULL,
  reply_markup = NULL
)
```

Arguments

chat_id	(Optional). Unique identifier for the target chat or username of the target channel.
message_id	(Optional). Required if inline_message_id is not specified. Identifier of the sent message.
inline_message_id	(Optional). Required if chat_id and message_id are not specified. Identifier of the inline message.
caption	(Optional). New caption of the message.
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.
reply_markup	(Optional). A Reply Markup parameter object, it can be either:

- ReplyKeyboardMarkup
- InlineKeyboardMarkup
- ReplyKeyboardRemove
- ForceReply

Details

You can also use it's snake_case equivalent edit_message_caption.

---

editMessageReplyMarkup

Edit a reply markup

Description

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

Usage

```
editMessageReplyMarkup(
  chat_id = NULL,
  message_id = NULL,
  inline_message_id = NULL,
  reply_markup = NULL
)
```

Arguments

chat_id	(Optional). Unique identifier for the target chat or username of the target channel.
message_id	(Optional). Required if inline_message_id is not specified. Identifier of the sent message.
inline_message_id	(Optional). Required if chat_id and message_id are not specified. Identifier of the inline message.
reply_markup	(Optional). A Reply Markup parameter object, it can be either: • ReplyKeyboardMarkup • InlineKeyboardMarkup • ReplyKeyboardRemove • ForceReply

Details

You can also use it's snake_case equivalent `edit_message_reply_markup`.

---

editMessageText *Edit a text message*

Description

Use this method to edit text messages.

Usage

```
editMessageText(
  chat_id = NULL,
  message_id = NULL,
  inline_message_id = NULL,
  text,
  parse_mode = NULL,
  disable_web_page_preview = NULL,
  reply_markup = NULL
)
```

Arguments

chat_id	(Optional). Unique identifier for the target chat or username of the target channel.
message_id	(Optional). Required if inline_message_id is not specified. Identifier of the sent message.

`inline_message_id`
: (Optional). Required if chat_id and message_id are not specified. Identifier of the inline message.

`text`
: New text of the message.

`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.

`disable_web_page_preview`
: (Optional). Disables link previews for links in this message.

`reply_markup`
: (Optional). A Reply Markup parameter object, it can be either:

- `ReplyKeyboardMarkup`
- `InlineKeyboardMarkup`
- `ReplyKeyboardRemove`
- `ForceReply`

Details

You can also use it's snake_case equivalent `edit_message_text`.

`effective_chat` *Get the effective chat*

Description

The chat that this update was sent in, no matter what kind of update this is. Will be `None` for `inline_query`, `chosen_inline_result`, `callback_query` from inline messages, `shipping_query` and `pre_checkout_query`.

Usage

```
effective_chat()
```

`effective_message` *Get the effective message*

Description

The message included in this update, no matter what kind of update this is. Will be `None` for `inline_query`, `chosen_inline_result`, `callback_query` from inline messages, `shipping_query` and `pre_checkout_query`.

Usage

```
effective_message()
```

effective_user *Get the effective user*

Description

The user that sent this update, no matter what kind of update this is. Will be NULL for channel_post.

Usage

```
effective_user()
```

ErrorHandler *Handling errors*

Description

Handler class to handle errors in the Dispatcher.

Usage

```
ErrorHandler(callback)

is.ErrorHandler(x)
```

Arguments

callback	A function that takes (bot, error) as arguments.
x	Object to be tested.

Format

An R6Class object.

Examples

```
## Not run:
updater <- Updater(token = "TOKEN")

# Create error callback
error_callback <- function(bot, error) {
  warning(simpleWarning(conditionMessage(error), call = "Updates polling"))
}

# Register it to the updater's dispatcher
updater$dispatcher$add_handler(ErrorHandler(error_callback))
# or
```

```
updater <- updater + ErrorHandler(error_callback)

## End(Not run)
```

`filtersLogic` *Combining filters*

Description

Creates a function which returns the corresponding logical operation between what `f` and `g` return.

Usage

```
## S3 method for class 'BaseFilter'
!f

## S3 method for class 'BaseFilter'
f & g

## S3 method for class 'BaseFilter'
f | g
```

Arguments

`f, g`	Arbitrary `BaseFilter` class functions.

Details

See `BaseFilter` and `MessageFilters` for further details.

Examples

```
not_command <- !MessageFilters$command
text_and_reply <- MessageFilters$text & MessageFilters$reply
audio_or_video <- MessageFilters$audio | MessageFilters$video
```

`ForceReply` *Display a reply*

Description

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').

Usage

```
ForceReply(force_reply = TRUE, selective = NULL)
```

Arguments

force_reply	Shows reply interface to the user, as if they manually selected the bot's message and tapped 'Reply'. Defaults to TRUE.
selective	(Optional). Use this parameter if you want to show the keyboard to specific users only.

Examples

```
## Not run:
# Initialize bot
bot <- Bot(token = "TOKEN")
chat_id <- "CHAT_ID"

# Set input parameters
text <- "Don't forget to send me the answer!"

# Send reply message
bot$sendMessage(chat_id, text, reply_markup = ForceReply())

## End(Not run)
```

---

forwardMessage *Forward messages of any kind*

Description

Use this method to forward messages of any kind.

Usage

```
forwardMessage(chat_id, from_chat_id, message_id, disable_notification = FALSE)
```

Arguments

chat_id	Unique identifier for the target chat or username of the target channel.
from_chat_id	Unique identifier for the chat where the original message was sent.
message_id	Message identifier in the chat specified in from_chat_id.
disable_notification	(Optional). Sends the message silently. Users will receive a notification with no sound.

Details

You can also use it's snake_case equivalent forward_message.

from_chat_id *Get an update's chat ID*

Description

Get the id from the Update's effective chat.

Usage

```
from_chat_id()
```

from_user_id *Get an update's user ID*

Description

Get the id from the Update's effective user.

Usage

```
from_user_id()
```

getFile *Prepare a file for downloading*

Description

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. It is guaranteed that the link will be valid for at least 1 hour. When the link expires, a new one can be requested by calling getFile again.

Usage

```
getFile(file_id, destfile = NULL, ...)
```

Arguments

file_id	The file identifier.
destfile	(Optional). If you want to save the file, pass by a character string with the name where the downloaded file is saved. See the destfile parameter from ?curl::curl_download for further details.
...	(Optional). Additional parameters to be passed to curl_download. It is not used if destfile is NULL.

Details

You can also use it's snake_case equivalent `get_file`.

Examples

```
## Not run:
bot <- Bot(token = bot_token("RTelegramBot"))
chat_id <- user_id("Me")

photos <- bot$getUserProfilePhotos(chat_id = chat_id)

# Download user profile photo
file_id <- photos$photos[[1L]][[1L]]$file_id
bot$getFile(file_id, destfile = "photo.jpg")

## End(Not run)
```

`getMe` *Check your bot's information*

Description

A simple method for testing your bot's auth token. Requires no parameters.

Usage

```
getMe()
```

Details

You can also use it's snake_case equivalent `get_me`.

`getUpdates` *Receive incoming updates*

Description

Use this method to receive incoming updates. It returns a list of `Update` objects.

Usage

```
getUpdates(offset = NULL, limit = 100L, timeout = 0L, allowed_updates = NULL)
```

Arguments

`offset`	(Optional). Identifier of the first update to be returned.
`limit`	(Optional). Limits the number of updates to be retrieved. Values between 1-100 are accepted. Defaults to 100.
`timeout`	(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`	(Optional). String or vector of strings with the types of updates you want your bot to receive. For example, specify `c("message", "edited_channel_post", "callback_query")` to only receive updates of these types. See Update for a complete list of available update types. Specify an empty string 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 getUpdates, so unwanted updates may be received for a short period of time.

Details

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 or use `Bot` method `clean_updates`.

3. To take full advantage of this library take a look at `Updater`.

You can also use it's snake_case equivalent `get_updates`.

Examples

```
## Not run:
bot <- Bot(token = bot_token("RTelegramBot"))

updates <- bot$getUpdates()

## End(Not run)
```

`getUserProfilePhotos` *Get a user's profile photos*

Description

Use this method to get a list of profile pictures for a user.

Usage

```
getUserProfilePhotos(user_id, offset = NULL, limit = 100L)
```

Arguments

user_id	Unique identifier of the target user.
offset	(Optional). Sequential number of the first photo to be returned. By default, all photos are returned.
limit	(Optional). Limits the number of photos to be retrieved. Values between 1-100 are accepted. Defaults to 100.

Details

You can also use it's snake_case equivalent `get_user_profile_photos`.

See `getFile` to know how to download files.

Examples

```
## Not run:
bot <- Bot(token = bot_token("RTelegramBot"))
chat_id <- user_id("Me")

photos <- bot$getUserProfilePhotos(chat_id = chat_id)

## End(Not run)
```

`getWebhookInfo` *Get current webhook status*

Description

Use this method to get current webhook status. Requires no parameters.

Usage

```
getWebhookInfo()
```

Details

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

You can also use it's snake_case equivalent `get_webhook_info`.

Handler *The base of all handlers*

Description

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

Usage

```
Handler(
  callback,
  check_update = NULL,
  handle_update = NULL,
  handlername = NULL
)

is.Handler(x)
```

Arguments

callback	The callback function for this handler. Its inputs will be (bot, update), where bot is a Bot instance and update an Update class.
check_update	Function that will override the default check_update method. Use it if you want to create your own Handler.
handle_update	Function that will override the default handle_update method. Use it if you want to create your own Handler.
handlername	Name of the customized class, which will inherit from Handler. If NULL (default) it will create a Handler class.
x	Object to be tested.

Format

An R6Class object.

Methods

check_update Called to determine if an update should be handled by this handler instance.

handle_update Called if it was determined that an update should indeed be handled by this instance.

Sub-classes

MessageHandler To handle Telegram messages.

CommandHandler To handle Telegram commands.

CallbackQueryHandler To handle Telegram callback queries.

ErrorHandler To handle errors while polling for updates.

Examples

```
## Not run:
# Example of a Handler
callback_method <- function(bot, update) {
  chat_id <- update$effective_chat()$id
  bot$sendMessage(chat_id = chat_id, text = "Hello")
}

hello_handler <- Handler(callback_method)

# Customizing Handler
check_update <- function(update) {
  TRUE
}

handle_update <- function(update, dispatcher) {
  self$callback(dispatcher$bot, update)
}

foo_handler <- Handler(callback_method,
  check_update = check_update,
  handle_update = handle_update,
  handlername = "FooHandler"
)

## End(Not run)
```

---

handle_update *Handle an update*

Description

This method is called if it was determined that an update should indeed be handled by this instance. It should also be overridden (see Handler).

Usage

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

Arguments

update	The update to be handled.
dispatcher	The dispatcher to collect optional arguments.

Details

In most cases `self$callback(dispatcher$bot, update)` can be called, possibly along with optional arguments.

`InlineKeyboardButton` *Create an inline keyboard button*

Description

This object represents one button of an inline keyboard. You **must** use exactly one of the optional fields. If all optional fields are NULL, by defect it will generate `callback_data` with same data as in `text`.

Usage

```
InlineKeyboardButton(
  text,
  url = NULL,
  callback_data = NULL,
  switch_inline_query = NULL,
  switch_inline_query_current_chat = NULL
)

is.InlineKeyboardButton(x)
```

Arguments

`text`	Label text on the button.
`url`	(Optional). HTTP url to be opened when button is pressed.
`callback_data`	(Optional). Data to be sent in a callback query to the bot when button is pressed, 1-64 bytes.
`switch_inline_query`	(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.
`switch_inline_query_current_chat`	(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.
`x`	Object to be tested.

Details

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

`InlineKeyboardMarkup` *Create an inline keyboard markup*

Description

This object represents an inline keyboard that appears right next to the message it belongs to.

Usage

```
InlineKeyboardMarkup(inline_keyboard)
```

Arguments

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

Details

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

Examples

```
## Not run:
# Initialize bot
bot <- Bot(token = "TOKEN")
chat_id <- "CHAT_ID"

# Create Inline Keyboard
text <- "Could you type their phone number, please?"
IKM <- InlineKeyboardMarkup(
  inline_keyboard = list(
    list(
      InlineKeyboardButton(1),
      InlineKeyboardButton(2),
      InlineKeyboardButton(3)
    ),
    list(
      InlineKeyboardButton(4),
      InlineKeyboardButton(5),
      InlineKeyboardButton(6)
    ),
    list(
      InlineKeyboardButton(7),
      InlineKeyboardButton(8),
      InlineKeyboardButton(9)
    ),
    list(
```

```
      InlineKeyboardButton("*"),
      InlineKeyboardButton(0),
      InlineKeyboardButton("#")
    )
  )
)

# Send Inline Keyboard
bot$sendMessage(chat_id, text, reply_markup = IKM)

## End(Not run)
```

`InlineQueryResult` *The base of inline query results*

Description

Baseclass for the InlineQueryResult* classes.

Usage

```
InlineQueryResult(type, id, ...)

is.InlineQueryResult(x)
```

Arguments

`type`	Type of the result. See the documentation for a list of supported types.
`id`	Unique identifier for this result, 1-64 Bytes.
`...`	Additional parameters for the selected type. See the documentation for the description of the parameters depending on the `InlineQueryResult` type.
`x`	Object to be tested.

Examples

```
## Not run:
document_url <- paste0(
  "https://github.com/ebeneditos/telegram.bot/raw/gh-pages/docs/",
  "telegram.bot.pdf"
)

result <- InlineQueryResult(
  type = "document",
  id = 1,
  title = "Documentation",
  document_url = document_url,
  mime_type = "application/pdf"
)

## End(Not run)
```

KeyboardButton *Create a keyboard button*

Description

This object represents one button of the reply keyboard. Optional fields are mutually exclusive.

Usage

```
KeyboardButton(text, request_contact = NULL, request_location = NULL)

is.KeyboardButton(x)
```

Arguments

text	Text of the button. If none of the optional fields are used, it will be sent as a message when the button is pressed.
request_contact	(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	(Optional). If TRUE, the user's current location will be sent when the button is pressed. Available in private chats only.
x	Object to be tested.

Details

Note: request_contact and request_location options will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

leaveChat *Leave a chat*

Description

Use this method for your bot to leave a group, supergroup or channel.

Usage

```
leaveChat(chat_id)
```

Arguments

chat_id	Unique identifier for the target chat or username of the target channel.

Details

You can also use it's snake_case equivalent leave_chat.

`MessageFilters` *Filter message updates*

Description

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

Usage

```
MessageFilters
```

Format

A `list` with filtering functions.

Details

See `BaseFilter` and `filtersLogic` for advanced filters.

Functions

- `all`: All Messages.
- `text`: Text Messages.
- `command`: Messages starting with `/`.
- `reply`: Messages that are a reply to another message.
- `audio`: Messages that contain audio.
- `document`: Messages that contain document.
- `photo`: Messages that contain photo.
- `sticker`: Messages that contain sticker.
- `video`: Messages that contain video.
- `voice`: Messages that contain voice.
- `contact`: Messages that contain contact.
- `location`: Messages that contain location.
- `venue`: Messages that are forwarded.
- `game`: Messages that contain game.

Examples

```
## Not run:
# Use to filter all video messages
video_handler <- MessageHandler(callback_method, MessageFilters$video)

# To filter all contacts, etc.
contact_handler <- MessageHandler(callback_method, MessageFilters$contact)

## End(Not run)
```

MessageHandler *Handling messages*

Description

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

Usage

```
MessageHandler(callback, filters = NULL)
```

Arguments

callback	The callback function for this handler. See Handler for information about this function.
filters	(Optional). Only allow updates with these filters. Use NULL (default) or MessageFilters$all for no filtering. See MessageFilters for a full list of all available filters.

Format

An R6Class object.

Examples

```
## Not run:
callback_method <- function(bot, update) {
  chat_id <- update$message$chat_id
  bot$sendMessage(chat_id = chat_id, text = "Hello")
}

# No filtering
message_handler <- MessageHandler(callback_method, MessageFilters$all)

## End(Not run)
```

ReplyKeyboardMarkup *Create a keyboard markup*

Description

This object represents a custom keyboard with reply options.

Usage

```
ReplyKeyboardMarkup(
  keyboard,
  resize_keyboard = NULL,
  one_time_keyboard = NULL,
  selective = NULL
)
```

Arguments

keyboard	List of button rows, each represented by a list of KeyboardButton objects.
resize_keyboard	(Optional). Requests clients to resize the keyboard vertically for optimal fit. Defaults to FALSE, in which case the custom keyboard is always of the same height as the app's standard keyboard.
one_time_keyboard	(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	(Optional). Use this parameter if you want to show the keyboard to specific users only.

Examples

```
## Not run:
# Initialize bot
bot <- Bot(token = "TOKEN")
chat_id <- "CHAT_ID"

# Create Custom Keyboard
text <- "Aren't those custom keyboards cool?"
RKM <- ReplyKeyboardMarkup(
  keyboard = list(
    list(KeyboardButton("Yes, they certainly are!")),
    list(KeyboardButton("I'm not quite sure")),
    list(KeyboardButton("No..."))
  ),
  resize_keyboard = FALSE,
  one_time_keyboard = TRUE
)

# Send Custom Keyboard
bot$sendMessage(chat_id, text, reply_markup = RKM)

## End(Not run)
```

ReplyKeyboardRemove *Remove a keyboard*

Description

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 ReplyKeyboardMarkup).

Usage

```
ReplyKeyboardRemove(remove_keyboard = TRUE, selective = NULL)
```

Arguments

remove_keyboard	Requests clients to remove the custom keyboard. (user will not be able to summon this keyboard; if you want to hide the keyboard from sight but keep it accessible, use one_time_keyboard in ReplyKeyboardMarkup). Defaults to TRUE.
selective	(Optional). Use this parameter if you want to show the keyboard to specific users only.

Examples

```
## Not run:
# Initialize bot
bot <- Bot(token = "TOKEN")
chat_id <- "CHAT_ID"

# Create Custom Keyboard
text <- "Don't forget to send me the answer!"
RKM <- ReplyKeyboardMarkup(
  keyboard = list(
    list(KeyboardButton("Yes, they certainly are!")),
    list(KeyboardButton("I'm not quite sure")),
    list(KeyboardButton("No..."))
  ),
  resize_keyboard = FALSE,
  one_time_keyboard = FALSE
)

# Send Custom Keyboard
bot$sendMessage(chat_id, text, reply_markup = RKM)

# Remove Keyboard
bot$sendMessage(
  chat_id,
```

```
  "Okay, thanks!",
  reply_markup = ReplyKeyboardRemove()
)

## End(Not run)
```

`running` *Retrieve the status of the Webhook.*

Description

Returns `TRUE` when listening for updates.

Usage

```
running()
```

`sendAnimation` *Send animation files*

Description

Use this method to send animation files (GIF or H.264/MPEG-4 AVC video without sound).

Usage

```
sendAnimation(
  chat_id,
  animation,
  duration = NULL,
  width = NULL,
  height = NULL,
  caption = NULL,
  parse_mode = NULL,
  disable_notification = FALSE,
  reply_to_message_id = NULL,
  reply_markup = NULL
)
```

Arguments

chat_id	Unique identifier for the target chat or username of the target channel.
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 local file by passing a file path.
duration	(Optional). Duration of sent audio in seconds.
width	(Optional). Video width.
height	(Optional). Video height.
caption	(Optional). Animation caption, 0-1024 characters.
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.
disable_notification	(Optional). Sends the message silently. Users will receive a notification with no sound.
reply_to_message_id	(Optional). If the message is a reply, ID of the original message.
reply_markup	(Optional). A Reply Markup parameter object, it can be either: • ReplyKeyboardMarkup • InlineKeyboardMarkup • ReplyKeyboardRemove • ForceReply

Details

You can also use it's snake_case equivalent send_animation.

Examples

```
## Not run:
bot <- Bot(token = bot_token("RTelegramBot"))
chat_id <- user_id("Me")
animation_url <- "http://techslides.com/demos/sample-videos/small.mp4"

bot$sendAnimation(
  chat_id = chat_id,
  animation = animation_url
)

## End(Not run)
```

sendAudio *Send audio files*

Description

Use this method to send audio files, if you want Telegram clients to display them in the music player. Your audio must be in the .mp3 format. On success, the sent Message is returned. Bots can currently send audio files of up to 50 MB in size, this limit may be changed in the future. For sending voice messages, use the sendVoice method instead.

Usage

```
sendAudio(
  chat_id,
  audio,
  duration = NULL,
  performer = NULL,
  title = NULL,
  caption = NULL,
  disable_notification = FALSE,
  reply_to_message_id = NULL,
  reply_markup = NULL,
  parse_mode = NULL
)
```

Arguments

chat_id	Unique identifier for the target chat or username of the target channel.
audio	Audio file to send. Pass a file_id as String to send an audio that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get an audio from the Internet, or upload a local audio file by passing a file path.
duration	(Optional). Duration of sent audio in seconds.
performer	(Optional). Performer.
title	(Optional). Track name.
caption	(Optional). Audio caption, 0-1024 characters.
disable_notification	(Optional). Sends the message silently. Users will receive a notification with no sound.
reply_to_message_id	(Optional). If the message is a reply, ID of the original message.
reply_markup	(Optional). A Reply Markup parameter object, it can be either: • ReplyKeyboardMarkup • InlineKeyboardMarkup

- ReplyKeyboardRemove
- ForceReply

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.

Details

You can also use it's snake_case equivalent send_audio.

Examples

```
## Not run:
bot <- Bot(token = bot_token("RTelegramBot"))
chat_id <- user_id("Me")
audio_url <- "http://www.largesound.com/ashborytour/sound/brobob.mp3"

bot$sendAudio(
  chat_id = chat_id,
  audio = audio_url
)

## End(Not run)
```

---

sendChatAction *Send a chat action*

Description

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).

Usage

```
sendChatAction(chat_id, action)
```

Arguments

chat_id Unique identifier for the target chat or username of the target channel.

action Type of action to broadcast. Choose one, depending on what the user is about to receive:

- typing for text messages
- upload_photo for photos
- upload_video for videos
- record_video for video recording
- upload_audio for audio files
- record_audio for audio file recording

upload_document for general files
find_location for location data
upload_video_note for video notes
record_video_note for video note recording

Details

You can also use it's snake_case equivalent send_chat_action.

Examples

```
## Not run:
bot <- Bot(token = bot_token("RTelegramBot"))
chat_id <- user_id("Me")

bot$sendChatAction(
  chat_id = chat_id,
  action = "typing"
)

## End(Not run)
```

---

sendDocument *Send general files*

Description

Use this method to send general files.

Usage

```
sendDocument(
  chat_id,
  document,
  filename = NULL,
  caption = NULL,
  disable_notification = FALSE,
  reply_to_message_id = NULL,
  reply_markup = NULL,
  parse_mode = NULL
)
```

Arguments

chat_id	Unique identifier for the target chat or username of the target channel.
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 local file by passing a file path

filename (Optional). File name that shows in telegram message.

caption (Optional). Document caption, 0-1024 characters.

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

reply_to_message_id
(Optional). If the message is a reply, ID of the original message.

reply_markup (Optional). A Reply Markup parameter object, it can be either:

- ReplyKeyboardMarkup
- InlineKeyboardMarkup
- ReplyKeyboardRemove
- ForceReply

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.

Details

You can also use it's snake_case equivalent send_document.

Examples

```
## Not run:
bot <- Bot(token = bot_token("RTelegramBot"))
chat_id <- user_id("Me")
document_url <- paste0(
  "https://github.com/ebeneditos/telegram.bot/raw/gh-pages/docs/",
  "telegram.bot.pdf"
)

bot$sendDocument(
  chat_id = chat_id,
  document = document_url
)

## End(Not run)
```

---

sendLocation *Send point on the map*

Description

Use this method to send point on the map.

Usage

```
sendLocation(
  chat_id,
  latitude,
  longitude,
  disable_notification = FALSE,
  reply_to_message_id = NULL,
  reply_markup = NULL
)
```

Arguments

chat_id	Unique identifier for the target chat or username of the target channel.
latitude	Latitude of location.
longitude	Longitude of location.
disable_notification	(Optional). Sends the message silently. Users will receive a notification with no sound.
reply_to_message_id	(Optional). If the message is a reply, ID of the original message.
reply_markup	(Optional). A Reply Markup parameter object, it can be either: • ReplyKeyboardMarkup • InlineKeyboardMarkup • ReplyKeyboardRemove • ForceReply

Details

You can also use it's snake_case equivalent send_location.

Examples

```
## Not run:
bot <- Bot(token = bot_token("RTelegramBot"))
chat_id <- user_id("Me")

bot$sendLocation(
  chat_id = chat_id,
  latitude = 51.521727,
  longitude = -0.117255
)

## End(Not run)
```

`sendMessage` *Send text messages*

Description

Use this method to send text messages.

Usage

```
sendMessage(
  chat_id,
  text,
  parse_mode = NULL,
  disable_web_page_preview = NULL,
  disable_notification = FALSE,
  reply_to_message_id = NULL,
  reply_markup = NULL
)
```

Arguments

`chat_id`	Unique identifier for the target chat or username of the target channel.
`text`	Text of the message to be sent.
`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.
`disable_web_page_preview`	(Optional). Disables link previews for links in this message.
`disable_notification`	(Optional). Sends the message silently. Users will receive a notification with no sound.
`reply_to_message_id`	(Optional). If the message is a reply, ID of the original message.
`reply_markup`	(Optional). A Reply Markup parameter object, it can be either: • `ReplyKeyboardMarkup` • `InlineKeyboardMarkup` • `ReplyKeyboardRemove` • `ForceReply`

Details

You can also use it's snake_case equivalent `send_message`.

Examples

```
## Not run:
bot <- Bot(token = bot_token("RTelegramBot"))
chat_id <- user_id("Me")

bot$sendMessage(
  chat_id = chat_id,
  text = "foo *bold* _italic_",
  parse_mode = "Markdown"
)

## End(Not run)
```

---

sendPhoto *Send image files*

Description

Use this method to send photos.

Usage

```
sendPhoto(
  chat_id,
  photo,
  caption = NULL,
  disable_notification = FALSE,
  reply_to_message_id = NULL,
  reply_markup = NULL,
  parse_mode = NULL
)
```

Arguments

chat_id	Unique identifier for the target chat or username of the target channel.
photo	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 local photo by passing a file path.
caption	(Optional). Photo caption (may also be used when re-sending photos by file_id), 0-1024 characters.
disable_notification	(Optional). Sends the message silently. Users will receive a notification with no sound.
reply_to_message_id	(Optional). If the message is a reply, ID of the original message.
reply_markup	(Optional). A Reply Markup parameter object, it can be either:

- ReplyKeyboardMarkup
- InlineKeyboardMarkup
- ReplyKeyboardRemove
- ForceReply

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.

Details

You can also use it's snake_case equivalent send_photo.

Examples

```
## Not run:
bot <- Bot(token = bot_token("RTelegramBot"))
chat_id <- user_id("Me")
photo_url <- "https://telegram.org/img/t_logo.png"

bot$sendPhoto(
  chat_id = chat_id,
  photo = photo_url,
  caption = "Telegram Logo"
)

## End(Not run)
```

---

sendSticker *Send a sticker*

Description

Use this method to send .webp stickers.

Usage

```
sendSticker(
  chat_id,
  sticker,
  disable_notification = FALSE,
  reply_to_message_id = NULL,
  reply_markup = NULL
)
```

Arguments

chat_id	Unique identifier for the target chat or username of the target channel.
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 local one by passing a file path.
disable_notification	(Optional). Sends the message silently. Users will receive a notification with no sound.
reply_to_message_id	(Optional). If the message is a reply, ID of the original message.
reply_markup	(Optional). A Reply Markup parameter object, it can be either:

- ReplyKeyboardMarkup
- InlineKeyboardMarkup
- ReplyKeyboardRemove
- ForceReply

Details

You can also use it's snake_case equivalent send_sticker.

Examples

```
## Not run:
bot <- Bot(token = bot_token("RTelegramBot"))
chat_id <- user_id("Me")
sticker_url <- "https://www.gstatic.com/webp/gallery/1.webp"

bot$sendSticker(
  chat_id = chat_id,
  sticker = sticker_url
)

## End(Not run)
```

---

sendVideo *Send a video*

Description

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

Usage

```
sendVideo(
  chat_id,
  video,
  duration = NULL,
  caption = NULL,
  disable_notification = FALSE,
  reply_to_message_id = NULL,
  reply_markup = NULL,
  width = NULL,
  height = NULL,
  parse_mode = NULL,
  supports_streaming = NULL
)
```

Arguments

chat_id	Unique identifier for the target chat or username of the target channel.
video	Video file to send. Pass a file_id as String to send a video that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a video from the Internet, or upload a local video file by passing a file path.
duration	(Optional). Duration of sent audio in seconds.
caption	(Optional). Video caption, 0-1024 characters.
disable_notification	(Optional). Sends the message silently. Users will receive a notification with no sound.
reply_to_message_id	(Optional). If the message is a reply, ID of the original message.
reply_markup	(Optional). A Reply Markup parameter object, it can be either: • ReplyKeyboardMarkup • InlineKeyboardMarkup • ReplyKeyboardRemove • ForceReply
width	(Optional). Video width.
height	(Optional). Video height.
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.
supports_streaming	(Optional). Pass TRUE, if the uploaded video is suitable for streaming.

Details

You can also use it's snake_case equivalent send_video.

Examples

```
## Not run:
bot <- Bot(token = bot_token("RTelegramBot"))
chat_id <- user_id("Me")
video_url <- "http://techslides.com/demos/sample-videos/small.mp4"

bot$sendVideo(
  chat_id = chat_id,
  video = video_url
)

## End(Not run)
```

---

sendVideoNote *Send video messages*

Description

Use this method to send video messages.

Usage

```
sendVideoNote(
  chat_id,
  video_note,
  duration = NULL,
  length = NULL,
  disable_notification = FALSE,
  reply_to_message_id = NULL,
  reply_markup = NULL
)
```

Arguments

chat_id	Unique identifier for the target chat or username of the target channel.
video_note	Video note file to send. Pass a file_id as String to send a video note that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a video note from the Internet, or upload a local video note file by passing a file path.
duration	(Optional). Duration of sent audio in seconds.
length	(Optional). Video width and height.
disable_notification	(Optional). Sends the message silently. Users will receive a notification with no sound.
reply_to_message_id	(Optional). If the message is a reply, ID of the original message.

reply_markup (Optional). A Reply Markup parameter object, it can be either:

- ReplyKeyboardMarkup
- InlineKeyboardMarkup
- ReplyKeyboardRemove
- ForceReply

Details

You can also use it's snake_case equivalent send_video_note.

Examples

```
## Not run:
bot <- Bot(token = bot_token("RTelegramBot"))
chat_id <- user_id("Me")
video_note_url <- "http://techslides.com/demos/sample-videos/small.mp4"

bot$sendVideoNote(
  chat_id = chat_id,
  video_note = video_note_url
)

## End(Not run)
```

---

sendVoice *Send voice files*

Description

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 with sendAudio or sendDocument).

Usage

```
sendVoice(
  chat_id,
  voice,
  duration = NULL,
  caption = NULL,
  disable_notification = FALSE,
  reply_to_message_id = NULL,
  reply_markup = NULL,
  parse_mode = NULL
)
```

Arguments

chat_id	Unique identifier for the target chat or username of the target channel.
voice	Voice file to send. Pass a file_id as String to send a voice file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a voice file from the Internet, or upload a local voice file file by passing a file path.
duration	(Optional). Duration of sent audio in seconds.
caption	(Optional). Voice message caption, 0-1024 characters.
disable_notification	(Optional). Sends the message silently. Users will receive a notification with no sound.
reply_to_message_id	(Optional). If the message is a reply, ID of the original message.
reply_markup	(Optional). A Reply Markup parameter object, it can be either: • ReplyKeyboardMarkup • InlineKeyboardMarkup • ReplyKeyboardRemove • ForceReply
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.

Details

You can also use it's snake_case equivalent send_voice.

Examples

```
## Not run:
bot <- Bot(token = bot_token("RTelegramBot"))
chat_id <- user_id("Me")
ogg_url <- "https://upload.wikimedia.org/wikipedia/commons/c/c8/Example.ogg"

bot$sendVoice(
  chat_id = chat_id,
  voice = ogg_url
)

## End(Not run)
```

`setWebhook` *Set a webhook*

Description

Use this method to specify a url and receive incoming updates via an outgoing webhook. Whenever there is an update for the bot, we will send an HTTPS POST request to the specified url, containing a JSON-serialized Update.

Usage

```
setWebhook(
  url = NULL,
  certificate = NULL,
  max_connections = 40L,
  allowed_updates = NULL,
  ip_address = NULL,
  drop_pending_updates = FALSE,
  secret_token = NULL
)
```

Arguments

`url`	HTTPS url to send updates to. Use an empty string to remove webhook integration.
`certificate`	(Optional). Upload your public key certificate so that the root certificate in use can be checked. See Telegram's self-signed guide for details.
`max_connections`	(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`	(Optional). String or vector of strings with the types of updates you want your bot to receive. For example, specify `c("message", "edited_channel_post", "callback_query")` to only receive updates of these types. See Update for a complete list of available update types. Specify an empty string 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.
`ip_address`	(Optional). The fixed IP address which will be used to send webhook requests instead of the IP address resolved through DNS.
`drop_pending_updates`	(Optional). Pass True to drop all pending updates.

secret_token (Optional). A secret token to be sent in a header X-Telegram-Bot-Api-Secret-Token in every webhook request, 1-256 characters. Only characters A-Z, a-z, 0-9, _ and - are allowed. The header is useful to ensure that the request comes from a webhook set by you.

Details

If you'd like to make sure that the webhook request comes from Telegram, we recommend using a secret path in the URL, e.g. https://www.example.com/<token>.

You can also use it's snake_case equivalent set_webhook.

---

set_token *Change your bot's auth token*

Description

Use this method to change your bot's auth token.

Usage

```
set_token(token)
```

Arguments

token The bot's token given by the *BotFather*.

---

start_polling *Start polling*

Description

Starts polling updates from Telegram. You can stop the polling either by using the the interrupt R command in the session menu or with the stop_polling method.

Usage

```
start_polling(
  timeout = 10L,
  clean = FALSE,
  allowed_updates = NULL,
  verbose = FALSE
)
```

Arguments

timeout	(Optional). Passed to getUpdates. Default is 10.
clean	(Optional). Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is FALSE.
allowed_updates	(Optional). Passed to getUpdates.
verbose	(Optional). If TRUE, prints status of the polling. Default is FALSE.

Examples

```
## Not run:
# Start polling example
start <- function(bot, update) {
  bot$sendMessage(
    chat_id = update$message$chat_id,
    text = sprintf(
      "Hello %s!",
      update$message$from$first_name
    )
  )
}

updater <- Updater("TOKEN") + CommandHandler("start", start)

updater$start_polling(verbose = TRUE)

## End(Not run)
```

---

start_server *Start the webhook server.*

Description

Starts the webhook for updates from Telegram. You can stop listening either by using the RStudio's interrupt R command in the session menu or with the stop_server method.

Usage

```
start_server(host = "127.0.0.1", port = 5001, clean = FALSE, blocking = TRUE)
```

Arguments

host	a string that is a valid IPv4 or IPv6 address that is owned by this server, which the application will listen on. "0.0.0.0" represents all IPv4 addresses and "::/0" represents all IPv6 addresses. Default is "127.0.0.1".
port	a number or integer that indicates the server port that should be listened on. Note that on most Unix-like systems including Linux and Mac OS X, port numbers smaller than 1025 require root privileges. Default is 5001.

clean	(Optional). Whether to clean any pending updates on Telegram servers before actually starting to poll. Default is FALSE.
blocking	(Optional). Determines whether the method blocks whilst listening for updates from Telegram. Default is TRUE.

Examples

```
## Not run:
# Start webhook example
start <- function(bot, update) {
  bot$sendMessage(
    chat_id = update$message$chat_id,
    text = sprintf(
      "Hello %s!",
      update$message$from$first_name
    )
  )
}

webhook <- Webhook("https://example.com/webhook", "TOKEN") + CommandHandler("start", start)

webhook$start_server()

## End(Not run)
```

---

stop_polling *Stop polling*

Description

Stops the polling. Requires no parameters.

Usage

```
stop_polling()
```

Examples

```
## Not run:
# Example of a 'kill' command
kill <- function(bot, update) {
  bot$sendMessage(
    chat_id = update$message$chat_id,
    text = "Bye!"
  )
  # Clean 'kill' update
  bot$getUpdates(offset = update$update_id + 1)
  # Stop the updater polling
  updater$stop_polling()
```

```
}

updater <<- updater + CommandHandler("kill", kill)

updater$start_polling(verbose = TRUE) # Send '/kill' to the bot

## End(Not run)
```

`stop_server` *Stop the webhook server.*

Description

Stops listening on the webhook. Requires no parameters.

Usage

```
stop_server()
```

Examples

```
## Not run:
# Example of a 'kill' command
kill <- function(bot, update) {
  bot$sendMessage(
    chat_id = update$message$chat_id,
    text = "Bye!"
  )
  # Stop the webhook
  webhook$stop_server()
}

webhook <- Webhook("https://example.com/webhook", "TOKEN") + CommandHandler("start", start)

webhook$start_server()

## End(Not run)
```

`TelegramObject` *The base of telegram.bot objects*

Description

Base class for most telegram objects.

Usage

```
is.TelegramObject(x)
```

Arguments

x	Object to be tested.

Format

An R6Class generator object.

Methods

Public methods:

- TelegramObject$clone()

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
TelegramObject$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Update *Represent an update*

Description

This object represents an incoming Update.

Usage

```
Update(data)

is.Update(x)
```

Arguments

data	Data of the update.
x	Object to be tested.

Format

An R6Class object.

Methods

from_chat_id To get the id from the update's effective chat.

from_user_id To get the id from the update's effective user.

effective_chat To get the chat that this update was sent in, no matter what kind of update this is.

effective_user To get the user that sent this update, no matter what kind of update this is.

effective_message To get the message included in this update, no matter what kind of update this is.

Updater *Building a Telegram Bot with Update Polling*

Description

This class, which employs the class Dispatcher, provides a front-end to class Bot to the programmer, so you can focus on coding the bot. Its purpose is to receive the updates from Telegram and to deliver them to said dispatcher. The dispatcher supports Handler classes for different kinds of data: Updates from Telegram, basic text commands and even arbitrary types. See add (+) to learn more about building your Updater.

Usage

```
Updater(
  token = NULL,
  base_url = NULL,
  base_file_url = NULL,
  request_config = NULL,
  bot = NULL
)

is.Updater(x)
```

Arguments

token	(Optional). The bot's token given by the *BotFather*.
base_url	(Optional). Telegram Bot API service URL.
base_file_url	(Optional). Telegram Bot API file URL.
request_config	(Optional). Additional configuration settings to be passed to the bot's POST requests. See the config parameter from ?httr::POST for further details. The request_config settings are very useful for the advanced users who would like to control the default timeouts and/or control the proxy used for HTTP communication.
bot	(Optional). A pre-initialized Bot instance.
x	Object to be tested.

Format

An R6Class object.

Details

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

Methods

`start_polling` Starts polling updates from Telegram.

`stop_polling` Stops the polling.

References

Bots: An introduction for developers and Telegram Bot API

Examples

```
## Not run:
updater <- Updater(token = "TOKEN")

# In case you want to set a proxy (see ?httr:use_proxy)
updater <- Updater(
  token = "TOKEN",
  request_config = httr::use_proxy(...)
)

# Add a handler
start <- function(bot, update) {
  bot$sendMessage(
    chat_id = update$message$chat_id,
    text = sprintf(
      "Hello %s!",
      update$message$from$first_name
    )
  )
}
updater <- updater + CommandHandler("start", start)

# Start polling
updater$start_polling(verbose = TRUE) # Send '/start' to the bot

## End(Not run)
```

user_id *Get a user from environment*

Description

Obtain Telegram user id from system variables (in `.Renviron`) set according to the naming convention `R_TELEGRAM_USER_X` where `X` is the user's name.

Usage

```
user_id(user_name)
```

Arguments

`user_name`	The user's name.

Examples

```
## Not run:
# Open the `.Renviron` file
file.edit(path.expand(file.path("~", ".Renviron")))
# Add the line (uncomment and replace <user-id> by your Telegram user ID):
# R_TELEGRAM_USER_Me=<user-id>
# Save and restart R

user_id("Me")

## End(Not run)
```

Webhook *Building a Telegram Bot with a Webhook*

Description

This class, which employs the class `Dispatcher`, provides a front-end to class `Bot` to the programmer, so you can focus on coding the bot. Its purpose is to receive updates via webhook from Telegram and to deliver them to said dispatcher. The dispatcher supports `Handler` classes for different kinds of data: Updates from Telegram, basic text commands and even arbitrary types. See `add` (+) to learn more about building your `Webhook`.

Usage

```
Webhook(
  webhook_url,
  token = NULL,
  base_url = NULL,
  base_file_url = NULL,
  request_config = NULL,
  certificate = NULL,
  max_connections = NULL,
  allowed_updates = NULL,
  ip_address = NULL,
  drop_pending_updates = FALSE,
  verbose = FALSE,
  bot = NULL
)

is.Webhook(x)
```

Arguments

webhook_url	Webhook HTTPS url to send updates to. The url is conventionally suffixed with the /webhook path. **Note**: The url must be publicly accessible, since Telegram will need to make HTTP POST requests to the end-point for each update. For example, if you are deploying to Heroku, you can use the app's hostname, such as https://[name of app].herokuapp.com/webhook, or a custom hostname for a domain that belongs to you, such as https://app.yourcustomdomain.com/webhook.
token	(Optional). The bot's token given by the *BotFather*.
base_url	(Optional). Telegram Bot API service URL.
base_file_url	(Optional). Telegram Bot API file URL.
request_config	(Optional). Additional configuration settings to be passed to the bot's POST requests. See the config parameter from httr::POST for further details. The request_config settings are very useful for the advanced users who would like to control the default timeouts and/or control the proxy used for HTTP communication.
certificate	(Optional). Upload your public key certificate so that the root certificate in use can be checked. See Telegram's self-signed guide for details.
max_connections	(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	(Optional). String or vector of strings with the types of updates you want your bot to receive. For example, specify c("message", "edited_channel_post", "callback_query") to only receive updates of these types. See Update for a

complete list of available update types. Specify an empty string 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.

`ip_address` (Optional). The fixed IP address which will be used to send webhook requests instead of the IP address resolved through DNS.

`drop_pending_updates`
(Optional). Pass True to drop all pending updates.

`verbose` (Optional). If `TRUE`, prints status of the polling. Default is `FALSE`.

`bot` (Optional). A pre-initialized `Bot` instance.

`x` Object to be tested.

Format

An `R6Class` object.

Details

You **must** supply the `webhook_url` and either a `bot` or a `token` argument.

The `webhook_url` must be publicly accessible, since Telegram will need to make HTTP `POST` requests to the end-point for each update.

Security Note: `Webhook` encapsulates generating a `secret_token` which is used to validate that the request comes from a webhook set by you.

Methods

`start_server` Starts listening for updates from Telegram.

`stop_server` Stops listening for updates.

`running` Returns `TRUE` when listening for updates.

References

Bots: An introduction for developers, Telegram Bot API and Marvin's Marvellous Guide to All Things Webhook

Examples

```
## Not run:
webhook <- Webhook("https://example.com/webhook", "TOKEN")

# In case you want to set a proxy
webhook <- Webhook(
  webhook_url = "https://example.com/webhook",
  token = "TOKEN",
  request_config = httr::use_proxy(...),
  verbose = TRUE
)
```

```
# Add a handler
start <- function(bot, update) {
  bot$sendMessage(
    chat_id = update$message$chat_id,
    text = sprintf(
      "Hello %s!",
      update$message$from$first_name
    )
  )
}
webhook <- webhook + CommandHandler("start", start)

# Start polling
webhook$start_server() # Send '/start' to the bot

## End(Not run)
```

Index