Package 'telegram.bot'

February 12, 2018

Type Package

Title Develop a 'Telegram Bot' with R

Version 1.0.0

Maintainer Ernest Benedito <ebeneditos@gmail.com>

Description

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. It is built on top of the pure API implementation, being an extension of the 'telegram' package, an R wrapper around the 'Telegram Bot API' <http://core.telegram.org/bots/api>.

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

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

Imports R6, httr

Depends R (>= 3.2.3), telegram (>= 0.6.0)

Suggests covr, testthat

License LGPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

NeedsCompilation no

Author Ernest Benedito [aut, cre]

Repository CRAN

Date/Publication 2018-02-12 10:29:51 UTC

R topics documented:

`add_error_handler` *add_error_handler*

Description

Registers an error handler in the `Dispatcher`.

Usage

```
add_error_handler(callback)
```

Arguments

`callback`	A function that takes `(Bot, Update)` as arguments.

add_handler *add_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 = 1)
```

Arguments

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

Details

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.

Bot *Bot*

Description

This object represents a Telegram Bot. It inherits from `TGBot`. Thus, it has implemented all the API methods from that class. It also features the `get_updates` method, which allows the use of long polling; and the `set_webhook`, `delete_webhook` and `get_webhook_info` methods, which allow to manage webhooks.

Usage

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

Arguments

`token`	Bot's unique authentication.
`base_url`	Telegram Bot API service URL.
`base_file_url`	Telegram Bot API file URL.

Format

An R6Class object.

Details

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

Examples

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

## End(Not run)
```

CallbackQueryHandler *CallbackQueryHandler*

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_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_updates*

Description

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

Usage

```
clean_updates()
```

CommandHandler *CommandHandler*

Description

Handler class to handle Telegram commands.

Usage

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

Arguments

command	The command or list 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 Filters 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.

Format

An R6Class object.

---

delete_webhook *delete_webhook*

Description

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

Usage

```
delete_webhook()
```

Dispatcher *Dispatcher*

Description

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

Usage

```
Dispatcher(bot)
```

Arguments

bot	The bot object that should be passed to the handlers.

Format

An R6Class object.

Methods

add_handler Registers a handler in the Dispatcher.

add_error_handler Registers an error handler in the Dispatcher.

---

effective_chat *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 *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 *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()
```

`Filters` *Filters*

Description

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

Usage

```
Filters
```

Format

A `list` with filtering functions.

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, Filters$video)

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

## End(Not run)
```

`get_updates` *get_updates*

Description

Use this method to receive incoming updates using long polling.

Usage

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

Arguments

`offset`	(Optional). Identifier of the first update to be returned 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 get_updates, 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.
3. To take full advantage of this library take a look at `Updater`.

`get_webhook_info` *get_webhook_info*

Description

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

Usage

```
get_webhook_info()
```

Details

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

`Handler` *Handler*

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)
```

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.

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.

Examples

```
# 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')
```

`handle_update` *handle_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.

MessageHandler *MessageHandler*

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 Filters$all for no filtering. See Filters 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, Filters$all)

## End(Not run)
```

set_webhook *set_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

```
set_webhook(url = NULL, certificate = NULL, max_connections = 40,
  allowed_updates = 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.

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>`.

`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 = 10, clean = FALSE, allowed_updates = NULL,
  verbose = FALSE)
```

Arguments

timeout	(Optional). Passed to get_updates. 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 get_updates.
verbose	(Optional). If TRUE, prints status of the polling. Default is FALSE.

Examples

```
## Not run:
# Start polling example
updater <- Updater(token = 'TOKEN')

updater$start_polling(verbose = TRUE)

## End(Not run)
```

---

stop_polling *stop_polling*

Description

Stops the polling. Requires no parameters.

Usage

```
stop_polling()
```

Examples

```
## Not run:
# Supperassign the updater
updater <<- Updater(token = 'TOKEN')

# Example of a 'kill' command
kill <- function(bot, update){
  bot$sendMessage(chat_id = update$message$chat_id,
                  text = "Bye!")
  # Clean 'kill' update
  bot$get_updates(offset = update$update_id + 1)
  # Stop the updater polling
  updater$stop_polling()
}

updater$dispatcher$add_handler(CommandHandler('kill', kill))

updater$start_polling(verbose = T) # Send '/kill' to the bot

## End(Not run)
```

`telegram.bot` *telegram.bot*

Description

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. It is built on top of the pure API implementation, being an extension of the `telegram` package, an R wrapper around the Telegram Bot API.

Details

In this page you can learn how to build a Bot quickly with this package.

Main Classes

`Updater` Package main class. This class, which employs the class `Dispatcher`, provides a front-end to class `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.

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

`Handler` The base class for all update handlers.

`Bot` This object represents a Telegram Bot.

`Update` *Update*

Description

This object represents an incoming Update.

Usage

```
Update(data)
```

Arguments

`data`	Data of the update.

Format

An `R6Class` object.

Methods

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

`effective_chat` To get the chat that this update was sent in, 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` *Updater*

Description

Package main class. This class, which employs the class `Dispatcher`, provides a front-end to class `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. The dispatcher supports `Handler` classes for different kinds of data: Updates from Telegram, basic text commands and even arbitrary types.

Usage

```
Updater(token = NULL, bot = NULL)
```

Arguments

`token`	(Optional). The bot's token given by the @BotFather.
`bot`	(Optional). A pre-initialized `Bot` instance.

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')

## End(Not run)
```

Index