# tgintegration Documentation

*Release 0.1.5*

Joscha Götzer

May 09, 2018

# Contents

Contents:

# CHAPTER 1

# tgintegration

WORK IN PROGRESS. Take bugs with a grain of salt.  on top of Pyrogram.

- Free software: MIT license

## 1.1 Features

- Log into a Telegram user account and interact with bots
- Capable of sending messages and retrieving the bot's responses

## 1.2 Installation

All hail pip!

```
$ pip install tgintegration
```

## 1.3 Requirements

Same as Pyrogram:

- Python 3.4 or higher.
- A Telegram API key.

## 1.4 Usage

Suppose we want to write integration tests for @BotListBot by sending it a couple of messages and asserting that it responds the way it should. First, let's create a `BotIntegrationClient`:

```
from tgintegration import BotIntegrationClient

client = BotIntegrationClient(
    bot_under_test='@BotListBot',
    session_name='my_account',  # arbitrary file path to the Pyrogram session file
    api_id=API_ID,
    api_hash=API_HASH,
    max_wait_response=15,  # maximum timeout for bot responses
    min_wait_consecutive=2  # minimum time to wait for consecutive messages
)

client.start()
client.clear_chat()  # Let's start with a blank screen
```

Now let's send the `/start` command to the `bot_under_test` and "await" exactly three messages:

```
response = client.send_command_await("start", num_expected=3)

assert response.num_messages == 3
assert response.messages[0].sticker
```

The result should look like this:

Let's examine these buttons in the response...

```
second_message = response[1]

# Three buttons in the first row
assert len(second_message.reply_markup.inline_keyboard[0]) == 3
```

We can also find and press the inline keyboard buttons:

```
# Click the first button matching the pattern
examples = response.press_inline_button(pattern=r'.*Examples')

assert "Examples for contributing to the BotList" in examples.full_text
```

As the bot edits the message, `press_inline_button` automatically listens for `MessageEdited` updates and picks up on the edit, returning it as `Response`.

So what happens when we send an invalid query or the bot fails to respond?

```
try:
    # The following instruction will raise an `InvalidResponseError` after
    # `client.max_wait_response` seconds
    client.send_command_await("ayylmao")
except InvalidResponseError:
    print("Raised.")
```

The `BotIntegrationClient` is based off a regular Pyrogram `Client`, meaning that, in addition to the `*_await` methods, all normal calls still work:

```
client.send_message(client.bot_under_test, "Hello Pyrogram")
client.send_message_await("Hello Pyrogram")  # This automatically uses the bot_under_
↪test as the peer
client.send_voice_await("files/voice.ogg")
client.send_video_await("files/video.mp4")
```

### 1.4.1 Custom awaitable actions

The main logic for the timeout between sending a message and receiving a response from the user is handled in the `act_await_response` method:

```
def act_await_response(self, action: AwaitableAction) -> Response: ...
```

It expects an `AwaitableAction` which is a plan for a message to be sent, while the `BotIntegrationClient` just makes it easy and removes a lot of the boilerplate code to create these actions.

After executing the action, the client collects all incoming messages that match the `filters` and adds them to the response. Thus you can think of a `Response` object as a collection of messages returned by the peer in reaction to the executed `AwaitableAction`.

```
from tgintegration import AwaitableAction, Response
from pyrogram import Filters

peer = '@BotListBot'

action = AwaitableAction(
    func=client.send_message,
    kwargs=dict(
        chat_id=peer,
        text="**Hello World**",
        parse_mode='markdown'
    ),
    # Wait for messages only by the peer we're interacting with
    filters=Filters.user(peer) & Filters.incoming,
    # Time out and raise after 15 seconds
    max_wait=15
)

response = client.act_await_response(action)  # type: Response
```

## 1.5 Integrating with test frameworks

TODO

- py.test
- unittest

## 1.6 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

# CHAPTER 2

# Installation

## 2.1 Stable release

To install tgintegration, run this command in your terminal:

```
$ pip install tgintegration
```

This is the preferred method to install tgintegration, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

## 2.2 From sources

The sources for tgintegration can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/JosXa/tgintegration
```

Or download the tarball:

```
$ curl  -OL https://github.com/JosXa/tgintegration/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

# CHAPTER 3

# Usage

To use tgintegration in a project:

```
import tgintegration
```

CHAPTER 4

# tgintegration

## 4.1 TgIntegration package

### 4.1.1 Submodules

### 4.1.2 tgintegration.botintegrationclient module

### 4.1.3 tgintegration.interactionclient module

### 4.1.4 Module contents

# CHAPTER 5

# Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

## 5.1 Types of Contributions

### 5.1.1 Report Bugs

Report bugs at https://github.com/JosXa/tgintegration/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

### 5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

### 5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

### 5.1.4 Write Documentation

tgintegration could always use more documentation, whether as part of the official tgintegration docs, in docstrings, or even on the web in blog posts, articles, and such.

### 5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/JosXa/tgintegration/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

## 5.2 Get Started!

Ready to contribute? Here's how to set up *tgintegration* for local development.

1. Fork the *tgintegration* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/tgintegration.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv tgintegration
   $ cd tgintegration/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 tgintegration tests
   $ python setup.py test or py.test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

## 5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/JosXa/tgintegration/pull_requests and make sure that the tests pass for all supported Python versions.

## 5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_tgintegration
```

# CHAPTER 6

# Credits

## 6.1 Development Lead

- Joscha Götzer <joscha.goetzer@gmail.com>

## 6.2 Contributors

None yet. Why not be the first?

# CHAPTER 7

# History

## 7.1 0.1.0 (2018-04-30)

- First release on PyPI.

# CHAPTER 8

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

# Index