fbchat

Release 1.7.1

Taehoon Kim; Moreels Pieter-Jan; Mads Marquart

Jul 03, 2019

CONTENTS

Release v1.7.1. (*Installation*)

Facebook Chat (Messenger) for Python. This project was inspired by facebook-chat-api.

No XMPP or API key is needed. Just use your email and password.

Currently `fbchat` support Python 2.7, 3.4, 3.5 and 3.6:

`fbchat` works by emulating the browser. This means doing the exact same GET/POST requests and tricking Facebook into thinking it's accessing the website normally. Therefore, this API requires the credentials of a Facebook account.

Note: If you're having problems, please check the *FAQ*, before asking questions on Github

---

Warning: We are not responsible if your account gets banned for spammy activities, such as sending lots of messages to people you don't know, sending messages very quickly, sending spammy looking URLs, logging in and out very quickly... Be responsible Facebook citizens.

Note: Facebook now has an official API for chat bots, so if you're familiar with node.js, this might be what you're looking for.

If you're already familiar with the basics of how Facebook works internally, go to *Examples* to see example usage of `fbchat`

CHAPTER

ONE

OVERVIEW

1.1 Installation

1.1.1 Pip Install fbchat

To install fbchat, run this command:

```
$ pip install fbchat
```

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

1.1.2 Get the Source Code

fbchat is developed on GitHub, where the code is always available.

You can either clone the public repository:

```
$ git clone git://github.com/carpedm20/fbchat.git
```

Or, download a tarball:

```
$ curl -OL https://github.com/carpedm20/fbchat/tarball/master
# optionally, zipball is also available (for Windows users).
```

Once you have a copy of the source, you can embed it in your own Python package, or install it into your site-packages easily:

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

1.2 Introduction

`fbchat` uses your email and password to communicate with the Facebook server. That means that you should always store your password in a separate file, in case e.g. someone looks over your shoulder while you're writing code. You should also make sure that the file's access control is appropriately restrictive

1.2.1 Logging In

Simply create an instance of *`Client`*. If you have two factor authentication enabled, type the code in the terminal prompt (If you want to supply the code in another fashion, overwrite *`Client.on2FACode`*):

```
from fbchat import Client
from fbchat.models import *
client = Client('<email>', '<password>')
```

Replace `<email>` and `<password>` with your email and password respectively

---

Note: For ease of use then most of the code snippets in this document will assume you've already completed the login process Though the second line, `from fbchat.models import *`, is not strictly neccesary here, later code snippets will assume you've done this

---

If you want to change how verbose `fbchat` is, change the logging level (in *`Client`*)

Throughout your code, if you want to check whether you are still logged in, use *`Client.isLoggedIn`*. An example would be to login again if you've been logged out, using *`Client.login`*:

```
if not client.isLoggedIn():
    client.login('<email>', '<password>')
```

When you're done using the client, and want to securely logout, use *`Client.logout`*:

```
client.logout()
```

1.2.2 Threads

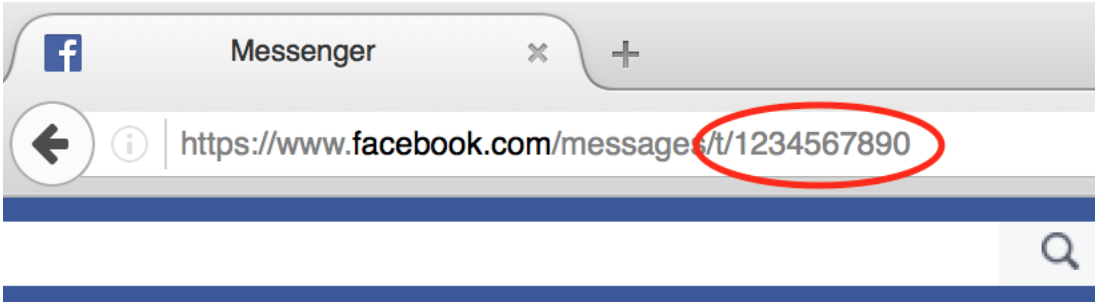
A thread can refer to two things: A Messenger group chat or a single Facebook user

`ThreadType` is an enumerator with two values: `USER` and `GROUP`. These will specify whether the thread is a single user chat or a group chat. This is required for many of `fbchat`'s functions, since Facebook differentiates between these two internally

Searching for group chats and finding their ID can be done via. *`Client.searchForGroups`*, and searching for users is possible via. *`Client.searchForUsers`*. See *Fetching Information*

You can get your own user ID by using *`Client.uid`*

Getting the ID of a group chat is fairly trivial otherwise, since you only need to navigate to https://www.facebook.com/messages/, click on the group you want to find the ID of, and then read the id from the address bar. The URL will look something like this: `https://www.facebook.com/messages/t/1234567890`, where `1234567890` would be the ID of the group. An image to illustrate this is shown below:



The same method can be applied to some user accounts, though if they've set a custom URL, then you'll just see that URL instead

Here's an snippet showing the usage of thread IDs and thread types, where `<user id>` and `<group id>` corresponds to the ID of a single user, and the ID of a group respectively:

```
client.send(Message(text='<message>'), thread_id='<user id>', thread_type=ThreadType.
↪USER)
client.send(Message(text='<message>'), thread_id='<group id>', thread_type=ThreadType.
↪GROUP)
```

Some functions (e.g. *Client.changeThreadColor*) don't require a thread type, so in these cases you just provide the thread ID:

```
client.changeThreadColor(ThreadColor.BILOBA_FLOWER, thread_id='<user id>')
client.changeThreadColor(ThreadColor.MESSENGER_BLUE, thread_id='<group id>')
```

1.2.3 Message IDs

Every message you send on Facebook has a unique ID, and every action you do in a thread, like changing a nickname or adding a person, has a unique ID too.

Some of `fbchat`'s functions require these ID's, like *Client.reactToMessage*, and some of then provide this ID, like *Client.sendMessage*. This snippet shows how to send a message, and then use the returned ID to react to that message with a emoji:

```
message_id = client.send(Message(text='message'), thread_id=thread_id, thread_
↪type=thread_type)
client.reactToMessage(message_id, MessageReaction.LOVE)
```

1.2.4 Interacting with Threads

`fbchat` provides multiple functions for interacting with threads

Most functionality works on all threads, though some things, like adding users to and removing users from a group chat, logically only works on group chats

The simplest way of using `fbchat` is to send a message. The following snippet will, as you've probably already figured out, send the message `test message` to your account:

```
message_id = client.send(Message(text='test message'), thread_id=client.uid, thread_
↪type=ThreadType.USER)
```

You can see a full example showing all the possible thread interactions with `fbchat` by going to *Examples*

1.2.5 Fetching Information

You can use `fbchat` to fetch basic information like user names, profile pictures, thread names and user IDs

You can retrieve a user's ID with *Client.searchForUsers*. The following snippet will search for users by their name, take the first (and most likely) user, and then get their user ID from the result:

```
users = client.searchForUsers('<name of user>')
user = users[0]
print("User's ID: {}".format(user.uid))
print("User's name: {}".format(user.name))
print("User's profile picture url: {}".format(user.photo))
print("User's main url: {}".format(user.url))
```

Since this uses Facebook's search functions, you don't have to specify the whole name, first names will usually be enough

You can see a full example showing all the possible ways to fetch information with `fbchat` by going to *Examples*

1.2.6 Sessions

`fbchat` provides functions to retrieve and set the session cookies. This will enable you to store the session cookies in a separate file, so that you don't have to login each time you start your script. Use *`Client.getSession`* to retrieve the cookies:

```
session_cookies = client.getSession()
```

Then you can use *`Client.setSession`*:

```
client.setSession(session_cookies)
```

Or you can set the `session_cookies` on your initial login. (If the session cookies are invalid, your email and password will be used to login instead):

```
client = Client('<email>', '<password>', session_cookies=session_cookies)
```

Warning: You session cookies can be just as valueable as you password, so store them with equal care

1.2.7 Listening & Events

To use the listening functions `fbchat` offers (like *`Client.listen`*), you have to define what should be executed when certain events happen. By default, (most) events will just be a `logging.info` statement, meaning it will simply print information to the console when an event happens

Note: You can identify the event methods by their `on` prefix, e.g. *`onMessage`*

The event actions can be changed by subclassing the *`Client`*, and then overwriting the event methods:

```
class CustomClient(Client):
    def onMessage(self, mid, author_id, message_object, thread_id, thread_type, ts,␣
↪metadata, msg, **kwargs):
        # Do something with message_object here
        pass

client = CustomClient('<email>', '<password>')
```

Notice: The following snippet is as equally valid as the previous one:

```
class CustomClient(Client):
    def onMessage(self, message_object, author_id, thread_id, thread_type, **kwargs):
        # Do something with message_object here
        pass

client = CustomClient('<email>', '<password>')
```

The change was in the parameters that our *onMessage* method took: `message_object` and `author_id` got swapped, and `mid`, `ts`, `metadata` and `msg` got removed, but the function still works, since we included `**kwargs`

Note: Therefore, for both backwards and forwards compatability, the API actually requires that you include `**kwargs` as your final argument.

View the *Examples* to see some more examples illustrating the event system

1.3 Examples

These are a few examples on how to use `fbchat`. Remember to swap out `<email>` and `<password>` for your email and password

1.3.1 Basic example

This will show basic usage of `fbchat`

```
# -*- coding: UTF-8 -*-

from fbchat import Client
from fbchat.models import *

client = Client("<email>", "<password>")

print("Own id: {}".format(client.uid))

client.send(Message(text="Hi me!"), thread_id=client.uid, thread_type=ThreadType.USER)

client.logout()
```

1.3.2 Interacting with Threads

This will interact with the thread in every way `fbchat` supports

```
# -*- coding: UTF-8 -*-

from fbchat import Client
from fbchat.models import *

client = Client("<email>", "<password>")

thread_id = "1234567890"
thread_type = ThreadType.GROUP

# Will send a message to the thread
client.send(Message(text="<message>"), thread_id=thread_id, thread_type=thread_type)

# Will send the default `like` emoji
client.send(
    Message(emoji_size=EmojiSize.LARGE), thread_id=thread_id, thread_type=thread_type
)
```

(continues on next page)

(continued from previous page)

```
# Will send the emoji ``
client.send(
    Message(text="", emoji_size=EmojiSize.LARGE),
    thread_id=thread_id,
    thread_type=thread_type,
)

# Will send the sticker with ID `767334476626295`
client.send(
    Message(sticker=Sticker("767334476626295")),
    thread_id=thread_id,
    thread_type=thread_type,
)

# Will send a message with a mention
client.send(
    Message(
        text="This is a @mention", mentions=[Mention(thread_id, offset=10, length=8)]
    ),
    thread_id=thread_id,
    thread_type=thread_type,
)

# Will send the image located at `<image path>`
client.sendLocalImage(
    "<image path>",
    message=Message(text="This is a local image"),
    thread_id=thread_id,
    thread_type=thread_type,
)

# Will download the image at the url `<image url>`, and then send it
client.sendRemoteImage(
    "<image url>",
    message=Message(text="This is a remote image"),
    thread_id=thread_id,
    thread_type=thread_type,
)


# Only do these actions if the thread is a group
if thread_type == ThreadType.GROUP:
    # Will remove the user with ID `<user id>` from the thread
    client.removeUserFromGroup("<user id>", thread_id=thread_id)

    # Will add the user with ID `<user id>` to the thread
    client.addUsersToGroup("<user id>", thread_id=thread_id)

    # Will add the users with IDs `<1st user id>`, `<2nd user id>` and `<3th user id>
↪` to the thread
    client.addUsersToGroup(
        ["<1st user id>", "<2nd user id>", "<3rd user id>"], thread_id=thread_id
    )


# Will change the nickname of the user `<user_id>` to `<new nickname>`
```

(continues on next page)

(continued from previous page)

```
client.changeNickname(
    "<new nickname>", "<user id>", thread_id=thread_id, thread_type=thread_type
)

# Will change the title of the thread to `<title>`
client.changeThreadTitle("<title>", thread_id=thread_id, thread_type=thread_type)

# Will set the typing status of the thread to `TYPING`
client.setTypingStatus(
    TypingStatus.TYPING, thread_id=thread_id, thread_type=thread_type
)

# Will change the thread color to `MESSENGER_BLUE`
client.changeThreadColor(ThreadColor.MESSENGER_BLUE, thread_id=thread_id)

# Will change the thread emoji to ``
client.changeThreadEmoji("", thread_id=thread_id)

# Will react to a message with a  emoji
client.reactToMessage("<message id>", MessageReaction.LOVE)
```

1.3.3 Fetching Information

This will show the different ways of fetching information about users and threads

```
# -*- coding: UTF-8 -*-

from fbchat import Client
from fbchat.models import *

client = Client("<email>", "<password>")

# Fetches a list of all users you're currently chatting with, as `User` objects
users = client.fetchAllUsers()

print("users' IDs: {}".format([user.uid for user in users]))
print("users' names: {}".format([user.name for user in users]))


# If we have a user id, we can use `fetchUserInfo` to fetch a `User` object
user = client.fetchUserInfo("<user id>")["<user id>"]
# We can also query both mutiple users together, which returns list of `User` objects
users = client.fetchUserInfo("<1st user id>", "<2nd user id>", "<3rd user id>")

print("user's name: {}".format(user.name))
print("users' names: {}".format([users[k].name for k in users]))


# `searchForUsers` searches for the user and gives us a list of the results,
# and then we just take the first one, aka. the most likely one:
user = client.searchForUsers("<name of user>")[0]

print("user ID: {}".format(user.uid))
print("user's name: {}".format(user.name))
print("user's photo: {}".format(user.photo))
```

(continues on next page)

(continued from previous page)

```
print("Is user client's friend: {}".format(user.is_friend))


# Fetches a list of the 20 top threads you're currently chatting with
threads = client.fetchThreadList()
# Fetches the next 10 threads
threads += client.fetchThreadList(offset=20, limit=10)

print("Threads: {}".format(threads))


# Gets the last 10 messages sent to the thread
messages = client.fetchThreadMessages(thread_id="<thread id>", limit=10)
# Since the message come in reversed order, reverse them
messages.reverse()

# Prints the content of all the messages
for message in messages:
    print(message.text)


# If we have a thread id, we can use `fetchThreadInfo` to fetch a `Thread` object
thread = client.fetchThreadInfo("<thread id>")["<thread id>"]
print("thread's name: {}".format(thread.name))
print("thread's type: {}".format(thread.type))


# `searchForThreads` searches works like `searchForUsers`, but gives us a list of␣
↪threads instead
thread = client.searchForThreads("<name of thread>")[0]
print("thread's name: {}".format(thread.name))
print("thread's type: {}".format(thread.type))


# Here should be an example of `getUnread`
```

1.3.4 Echobot

This will reply to any message with the same message

```
# -*- coding: UTF-8 -*-

from fbchat import log, Client

# Subclass fbchat.Client and override required methods
class EchoBot(Client):
    def onMessage(self, author_id, message_object, thread_id, thread_type, **kwargs):
        self.markAsDelivered(thread_id, message_object.uid)
        self.markAsRead(thread_id)

        log.info("{} from {} in {}".format(message_object, thread_id, thread_type.
↪name))

        # If you're not the author, echo
        if author_id != self.uid:
```

(continues on next page)

(continued from previous page)

```
            self.send(message_object, thread_id=thread_id, thread_type=thread_type)


client = EchoBot("<email>", "<password>")
client.listen()
```

1.3.5 Remove Bot

This will remove a user from a group if they write the message `Remove me!`

```
# -*- coding: UTF-8 -*-

from fbchat import log, Client
from fbchat.models import *


class RemoveBot(Client):
    def onMessage(self, author_id, message_object, thread_id, thread_type, **kwargs):
        # We can only kick people from group chats, so no need to try if it's a user␣
↪chat
        if message_object.text == "Remove me!" and thread_type == ThreadType.GROUP:
            log.info("{} will be removed from {}".format(author_id, thread_id))
            self.removeUserFromGroup(author_id, thread_id=thread_id)
        else:
            # Sends the data to the inherited onMessage, so that we can still see␣
↪when a message is recieved
            super(RemoveBot, self).onMessage(
                author_id=author_id,
                message_object=message_object,
                thread_id=thread_id,
                thread_type=thread_type,
                **kwargs
            )


client = RemoveBot("<email>", "<password>")
client.listen()
```

1.3.6 "Prevent changes"-Bot

This will prevent chat color, emoji, nicknames and chat name from being changed. It will also prevent people from being added and removed

```
# -*- coding: UTF-8 -*-

from fbchat import log, Client
from fbchat.models import *

# Change this to your group id
old_thread_id = "1234567890"

# Change these to match your liking
old_color = ThreadColor.MESSENGER_BLUE
```

(continues on next page)

(continued from previous page)

```
old_emoji = ""
old_title = "Old group chat name"
old_nicknames = {
    "12345678901": "User nr. 1's nickname",
    "12345678902": "User nr. 2's nickname",
    "12345678903": "User nr. 3's nickname",
    "12345678904": "User nr. 4's nickname",
}


class KeepBot(Client):
    def onColorChange(self, author_id, new_color, thread_id, thread_type, **kwargs):
        if old_thread_id == thread_id and old_color != new_color:
            log.info(
                "{} changed the thread color. It will be changed back".format(author_
↪id)
            )
            self.changeThreadColor(old_color, thread_id=thread_id)

    def onEmojiChange(self, author_id, new_emoji, thread_id, thread_type, **kwargs):
        if old_thread_id == thread_id and new_emoji != old_emoji:
            log.info(
                "{} changed the thread emoji. It will be changed back".format(author_
↪id)
            )
            self.changeThreadEmoji(old_emoji, thread_id=thread_id)

    def onPeopleAdded(self, added_ids, author_id, thread_id, **kwargs):
        if old_thread_id == thread_id and author_id != self.uid:
            log.info("{} got added. They will be removed".format(added_ids))
            for added_id in added_ids:
                self.removeUserFromGroup(added_id, thread_id=thread_id)

    def onPersonRemoved(self, removed_id, author_id, thread_id, **kwargs):
        # No point in trying to add ourself
        if (
            old_thread_id == thread_id
            and removed_id != self.uid
            and author_id != self.uid
        ):
            log.info("{} got removed. They will be re-added".format(removed_id))
            self.addUsersToGroup(removed_id, thread_id=thread_id)

    def onTitleChange(self, author_id, new_title, thread_id, thread_type, **kwargs):
        if old_thread_id == thread_id and old_title != new_title:
            log.info(
                "{} changed the thread title. It will be changed back".format(author_
↪id)
            )
            self.changeThreadTitle(
                old_title, thread_id=thread_id, thread_type=thread_type
            )

    def onNicknameChange(
        self, author_id, changed_for, new_nickname, thread_id, thread_type, **kwargs
    ):
        if (
```

(continues on next page)

(continued from previous page)

```
            old_thread_id == thread_id
            and changed_for in old_nicknames
            and old_nicknames[changed_for] != new_nickname
        ):
            log.info(
                "{} changed {}'s' nickname. It will be changed back".format(
                    author_id, changed_for
                )
            )
            self.changeNickname(
                old_nicknames[changed_for],
                changed_for,
                thread_id=thread_id,
                thread_type=thread_type,
            )


client = KeepBot("<email>", "<password>")
client.listen()
```

1.4 Testing

To use the tests, copy `tests/data.json` to `tests/my_data.json` or type the information manually in the terminal prompts.

- email: Your (or a test user's) email / phone number
- password: Your (or a test user's) password
- group_thread_id: A test group that will be used to test group functionality
- user_thread_id: A person that will be used to test kick/add functionality (This user should be in the group)

Please remember to test all supported python versions. If you've made any changes to the 2FA functionality, test it with a 2FA enabled account.

If you only want to execute specific tests, pass the function names in the command line (not including the `test_` prefix). Example:

```
$ python tests.py sendMessage sessions sendEmoji
```

Warning: Do not execute the full set of tests in too quick succession. This can get your account temporarily blocked for spam! (You should execute the script at max about 10 times a day)

1.5 Full API

If you are looking for information on a specific function, class, or method, this part of the documentation is for you.

1.5.1 Client

class fbchat.**Client**(*email*, *password*, *user_agent=None*, *max_tries=5*, *session_cookies=None*, *logging_level=20*)

A client for the Facebook Chat (Messenger).

This is the main class of `fbchat`, which contains all the methods you use to interact with Facebook. You can extend this class, and overwrite the `on` methods, to provide custom event handling (mainly useful while listening).

Initialize and log in the client.

Parameters

- **email** – Facebook `email`, `id` or `phone number`
- **password** – Facebook account password
- **user_agent** – Custom user agent to use when sending requests. If `None`, user agent will be chosen from a premade list
- **max_tries** (*int*) – Maximum number of times to try logging in
- **session_cookies** (*dict*) – Cookies from a previous session (Will default to login if these are invalid)
- **logging_level** (*int*) – Configures the logging level. Defaults to `logging.INFO`

Raises FBchatException on failed login

listening = False

Whether the client is listening. Used when creating an external event loop to determine when to stop listening

property ssl_verify

Verify ssl certificate, set to False to allow debugging with a proxy.

property uid

The ID of the client.

Can be used as `thread_id`. See *Threads* for more info.

graphql_requests(**queries*)

Parameters **queries** (*dict*) – Zero or more dictionaries

Raises FBchatException if request failed

Returns A tuple containing json graphql queries

Return type tuple

graphql_request(*query*)

Shorthand for `graphql_requests(query)[0]`

Raises FBchatException if request failed

isLoggedIn()

Sends a request to Facebook to check the login status

Returns True if the client is still logged in

Return type bool

getSession()

Retrieves session cookies

Returns A dictionay containing session cookies

Return type dict

setSession(*session_cookies*, *user_agent=None*)
Loads session cookies

Parameters **session_cookies** (*dict*) – A dictionay containing session cookies

Returns False if `session_cookies` does not contain proper cookies

Return type bool

login(*email*, *password*, *max_tries=5*, *user_agent=None*)
Uses `email` and `password` to login the user (If the user is already logged in, this will do a re-login)

Parameters

- **email** – Facebook `email` or `id` or `phone number`
- **password** – Facebook account password
- **max_tries** (*int*) – Maximum number of times to try logging in

Raises FBchatException on failed login

logout()
Safely logs out the client

Returns True if the action was successful

Return type bool

setDefaultThread(*thread_id*, *thread_type*)
Sets default thread to send messages to

Parameters

- **thread_id** – User/Group ID to default to. See *Threads*
- **thread_type** (`ThreadType`) – See *Threads*

resetDefaultThread()
Resets default thread

fetchThreads(*thread_location*, *before=None*, *after=None*, *limit=None*)
Get all threads in thread_location. Threads will be sorted from newest to oldest.

Parameters

- **thread_location** – ThreadLocation: INBOX, PENDING, ARCHIVED or OTHER
- **before** – Fetch only thread before this epoch (in ms) (default all threads)
- **after** – Fetch only thread after this epoch (in ms) (default all threads)
- **limit** – The max. amount of threads to fetch (default all threads)

Returns *Thread* objects

Return type list

Raises FBchatException if request failed

fetchAllUsersFromThreads(*threads*)
Get all users involved in threads.

Parameters **threads** – Thread: List of threads to check for users

Returns *User* objects

Return type list

Raises FBchatException if request failed

fetchAllUsers()
Gets all users the client is currently chatting with

Returns *User* objects

Return type list

Raises FBchatException if request failed

searchForUsers(*name*, *limit=10*)
Find and get user by his/her name

Parameters

- **name** – Name of the user
- **limit** – The max. amount of users to fetch

Returns *User* objects, ordered by relevance

Return type list

Raises FBchatException if request failed

searchForPages(*name*, *limit=10*)
Find and get page by its name

Parameters **name** – Name of the page

Returns *Page* objects, ordered by relevance

Return type list

Raises FBchatException if request failed

searchForGroups(*name*, *limit=10*)
Find and get group thread by its name

Parameters

- **name** – Name of the group thread
- **limit** – The max. amount of groups to fetch

Returns *Group* objects, ordered by relevance

Return type list

Raises FBchatException if request failed

searchForThreads(*name*, *limit=10*)
Find and get a thread by its name

Parameters

- **name** – Name of the thread
- **limit** – The max. amount of groups to fetch

Returns *User*, *Group* and *Page* objects, ordered by relevance

Return type list

Raises FBchatException if request failed

searchForMessageIDs(*query*, *offset=0*, *limit=5*, *thread_id=None*)
Find and get message IDs by query

Parameters

- **query** – Text to search for
- **offset** (*int*) – Number of messages to skip
- **limit** (*int*) – Max. number of messages to retrieve
- **thread_id** – User/Group ID to search in. See *Threads*

Returns Found Message IDs

Return type typing.Iterable

Raises FBchatException if request failed

searchForMessages(*query*, *offset=0*, *limit=5*, *thread_id=None*)
Find and get *Message* objects by query

> **Warning:** This method sends request for every found message ID.

Parameters

- **query** – Text to search for
- **offset** (*int*) – Number of messages to skip
- **limit** (*int*) – Max. number of messages to retrieve
- **thread_id** – User/Group ID to search in. See *Threads*

Returns Found *Message* objects

Return type typing.Iterable

Raises FBchatException if request failed

search(*query*, *fetch_messages=False*, *thread_limit=5*, *message_limit=5*)
Searches for messages in all threads

Parameters

- **query** – Text to search for
- **fetch_messages** – Whether to fetch *Message* objects or IDs only
- **thread_limit** (*int*) – Max. number of threads to retrieve
- **message_limit** (*int*) – Max. number of messages to retrieve

Returns Dictionary with thread IDs as keys and iterables to get messages as values

Return type typing.Dict[str, typing.Iterable]

Raises FBchatException if request failed

fetchUserInfo(**user_ids*)
Get users' info from IDs, unordered

> **Warning:** Sends two requests, to fetch all available info!

Parameters **user_ids** – One or more user ID(s) to query

Returns *User* objects, labeled by their ID

Return type dict

Raises FBchatException if request failed

fetchPageInfo(**page_ids*)
Get pages' info from IDs, unordered

> **Warning:** Sends two requests, to fetch all available info!

Parameters **page_ids** – One or more page ID(s) to query

Returns *Page* objects, labeled by their ID

Return type dict

Raises FBchatException if request failed

fetchGroupInfo(**group_ids*)
Get groups' info from IDs, unordered

Parameters **group_ids** – One or more group ID(s) to query

Returns *Group* objects, labeled by their ID

Return type dict

Raises FBchatException if request failed

fetchThreadInfo(**thread_ids*)
Get threads' info from IDs, unordered

> **Warning:** Sends two requests if users or pages are present, to fetch all available info!

Parameters **thread_ids** – One or more thread ID(s) to query

Returns *Thread* objects, labeled by their ID

Return type dict

Raises FBchatException if request failed

fetchThreadMessages(*thread_id=None*, *limit=20*, *before=None*)
Get the last messages in a thread

Parameters

- **thread_id** – User/Group ID to get messages from. See *Threads*
- **limit** (*int*) – Max. number of messages to retrieve
- **before** (*int*) – A timestamp, indicating from which point to retrieve messages

Returns *Message* objects

Return type list

Raises FBchatException if request failed

fetchThreadList(*offset=None*, *limit=20*, *thread_location=ThreadLocation.INBOX*, *before=None*)
Get thread list of your facebook account

Parameters

- **offset** – Deprecated. Do not use!
- **limit** (*int*) – Max. number of threads to retrieve. Capped at 20
- **thread_location** – ThreadLocation: INBOX, PENDING, ARCHIVED or OTHER
- **before** (*int*) – A timestamp (in milliseconds), indicating from which point to retrieve threads

Returns *Thread* objects

Return type list

Raises FBchatException if request failed

fetchUnread()
Get the unread thread list

Returns List of unread thread ids

Return type list

Raises FBchatException if request failed

fetchUnseen()
Get the unseen (new) thread list

Returns List of unseen thread ids

Return type list

Raises FBchatException if request failed

fetchImageUrl(*image_id*)
Fetches the url to the original image from an image attachment ID

Parameters **image_id** (*str*) – The image you want to fethc

Returns An url where you can download the original image

Return type str

Raises FBchatException if request failed

fetchMessageInfo(*mid*, *thread_id=None*)
Fetches *Message* object from the message id

Parameters

- **mid** – Message ID to fetch from
- **thread_id** – User/Group ID to get message info from. See *Threads*

Returns *Message* object

Return type *Message*

Raises FBchatException if request failed

fetchPollOptions(*poll_id*)
Fetches list of *PollOption* objects from the poll id

Parameters **poll_id** – Poll ID to fetch from

Return type list

Raises FBchatException if request failed

fetchPlanInfo(*plan_id*)
Fetches a *Plan* object from the plan id

Parameters **plan_id** – Plan ID to fetch from

Returns *Plan* object

Return type *Plan*

Raises FBchatException if request failed

getPhoneNumbers()
Fetches a list of user phone numbers.

Returns List of phone numbers

Return type list

getEmails()
Fetches a list of user emails.

Returns List of emails

Return type list

getUserActiveStatus(*user_id*)
Gets friend active status as an *ActiveStatus* object. Returns `None` if status isn't known.

Warning: Only works when listening.

Parameters **user_id** – ID of the user

Returns Given user active status

Return type *ActiveStatus*

send(*message*, *thread_id=None*, *thread_type=ThreadType.USER*)
Sends a message to a thread

Parameters

- **message** (Message) – Message to send
- **thread_id** – User/Group ID to send to. See *Threads*
- **thread_type** (ThreadType) – See *Threads*

Returns *Message ID* of the sent message

Raises FBchatException if request failed

sendMessage(*message*, *thread_id=None*, *thread_type=ThreadType.USER*)
Deprecated. Use *fbchat.Client.send* instead

sendEmoji(*emoji=None*, *size=EmojiSize.SMALL*, *thread_id=None*, *thread_type=ThreadType.USER*)
Deprecated. Use *fbchat.Client.send* instead

wave(*wave_first=True*, *thread_id=None*, *thread_type=None*)
Says hello with a wave to a thread!

Parameters

- **wave_first** – Whether to wave first or wave back
- **thread_id** – User/Group ID to send to. See *Threads*
- **thread_type** (ThreadType) – See *Threads*

Returns *Message ID* of the sent message

Raises FBchatException if request failed

quickReply(*quick_reply*, *payload=None*, *thread_id=None*, *thread_type=None*)
Replies to a chosen quick reply

Parameters

- **quick_reply** (QuickReply) – Quick reply to reply to
- **payload** – Optional answer to the quick reply
- **thread_id** – User/Group ID to send to. See *Threads*
- **thread_type** (ThreadType) – See *Threads*

Returns *Message ID* of the sent message

Raises FBchatException if request failed

unsend(*mid*)
Unsends a message (removes for everyone)

Parameters **mid** – *Message ID* of the message to unsend

sendLocation(*location*, *message=None*, *thread_id=None*, *thread_type=None*)
Sends a given location to a thread as the user's current location

Parameters

- **location** (LocationAttachment) – Location to send
- **message** (Message) – Additional message
- **thread_id** – User/Group ID to send to. See *Threads*
- **thread_type** (ThreadType) – See *Threads*

Returns *Message ID* of the sent message

Raises FBchatException if request failed

sendPinnedLocation(*location*, *message=None*, *thread_id=None*, *thread_type=None*)
Sends a given location to a thread as a pinned location

Parameters

- **location** (LocationAttachment) – Location to send
- **message** (Message) – Additional message
- **thread_id** – User/Group ID to send to. See *Threads*
- **thread_type** (ThreadType) – See *Threads*

Returns *Message ID* of the sent message

Raises FBchatException if request failed

sendRemoteFiles(*file_urls*, *message=None*, *thread_id=None*, *thread_type=ThreadType.USER*)
Sends files from URLs to a thread

Parameters

- **file_urls** – URLs of files to upload and send
- **message** – Additional message
- **thread_id** – User/Group ID to send to. See *Threads*
- **thread_type** (ThreadType) – See *Threads*

Returns *Message ID* of the sent files

Raises FBchatException if request failed

sendLocalFiles(*file_paths*, *message=None*, *thread_id=None*, *thread_type=ThreadType.USER*)
Sends local files to a thread

Parameters

- **file_paths** – Paths of files to upload and send
- **message** – Additional message
- **thread_id** – User/Group ID to send to. See *Threads*
- **thread_type** (ThreadType) – See *Threads*

Returns *Message ID* of the sent files

Raises FBchatException if request failed

sendRemoteVoiceClips(*clip_urls*, *message=None*, *thread_id=None*, *thread_type=ThreadType.USER*)
Sends voice clips from URLs to a thread

Parameters

- **clip_urls** – URLs of clips to upload and send
- **message** – Additional message
- **thread_id** – User/Group ID to send to. See *Threads*
- **thread_type** (ThreadType) – See *Threads*

Returns *Message ID* of the sent files

Raises FBchatException if request failed

sendLocalVoiceClips(*clip_paths*, *message=None*, *thread_id=None*, *thread_type=ThreadType.USER*)
Sends local voice clips to a thread

Parameters

- **clip_paths** – Paths of clips to upload and send
- **message** – Additional message
- **thread_id** – User/Group ID to send to. See *Threads*
- **thread_type** (ThreadType) – See *Threads*

Returns *Message ID* of the sent files

Raises FBchatException if request failed

sendImage(*image_id*, *message=None*, *thread_id=None*, *thread_type=ThreadType.USER*, *is_gif=False*)
Deprecated.

sendRemoteImage(*image_url*, *message=None*, *thread_id=None*, *thread_type=ThreadType.USER*)
Deprecated. Use *fbchat.Client.sendRemoteFiles* instead

sendLocalImage(*image_path*, *message=None*, *thread_id=None*, *thread_type=ThreadType.USER*)
Deprecated. Use *fbchat.Client.sendLocalFiles* instead

forwardAttachment(*attachment_id*, *thread_id=None*)
Forwards an attachment

Parameters

- **attachment_id** – Attachment ID to forward
- **thread_id** – User/Group ID to send to. See *Threads*

Raises FBchatException if request failed

createGroup(*message*, *user_ids*)
Creates a group with the given ids

Parameters

- **message** – The initial message
- **user_ids** – A list of users to create the group with.

Returns ID of the new group

Raises FBchatException if request failed

addUsersToGroup(*user_ids*, *thread_id=None*)
Adds users to a group.

Parameters

- **user_ids** (*list*) – One or more user IDs to add
- **thread_id** – Group ID to add people to. See *Threads*

Raises FBchatException if request failed

removeUserFromGroup(*user_id*, *thread_id=None*)
Removes users from a group.

Parameters

- **user_id** – User ID to remove
- **thread_id** – Group ID to remove people from. See *Threads*

Raises FBchatException if request failed

addGroupAdmins(*admin_ids*, *thread_id=None*)
Sets specifed users as group admins.

Parameters

- **admin_ids** – One or more user IDs to set admin
- **thread_id** – Group ID to remove people from. See *Threads*

Raises FBchatException if request failed

removeGroupAdmins(*admin_ids*, *thread_id=None*)
Removes admin status from specifed users.

Parameters

- **admin_ids** – One or more user IDs to remove admin

- **thread_id** – Group ID to remove people from. See *Threads*

Raises FBchatException if request failed

changeGroupApprovalMode(*require_admin_approval*, *thread_id=None*)
Changes group's approval mode

Parameters

- **require_admin_approval** – True or False
- **thread_id** – Group ID to remove people from. See *Threads*

Raises FBchatException if request failed

acceptUsersToGroup(*user_ids*, *thread_id=None*)
Accepts users to the group from the group's approval

Parameters

- **user_ids** – One or more user IDs to accept
- **thread_id** – Group ID to accept users to. See *Threads*

Raises FBchatException if request failed

denyUsersFromGroup(*user_ids*, *thread_id=None*)
Denies users from the group's approval

Parameters

- **user_ids** – One or more user IDs to deny
- **thread_id** – Group ID to deny users from. See *Threads*

Raises FBchatException if request failed

changeGroupImageRemote(*image_url*, *thread_id=None*)
Changes a thread image from a URL

Parameters

- **image_url** – URL of an image to upload and change
- **thread_id** – User/Group ID to change image. See *Threads*

Raises FBchatException if request failed

changeGroupImageLocal(*image_path*, *thread_id=None*)
Changes a thread image from a local path

Parameters

- **image_path** – Path of an image to upload and change
- **thread_id** – User/Group ID to change image. See *Threads*

Raises FBchatException if request failed

changeThreadTitle(*title*, *thread_id=None*, *thread_type=ThreadType.USER*)
Changes title of a thread. If this is executed on a user thread, this will change the nickname of that user, effectively changing the title

Parameters

- **title** – New group thread title
- **thread_id** – Group ID to change title of. See *Threads*

- **thread_type** (ThreadType) – See *Threads*

Raises FBchatException if request failed

changeNickname(*nickname*, *user_id*, *thread_id=None*, *thread_type=ThreadType.USER*)
Changes the nickname of a user in a thread

Parameters

- **nickname** – New nickname
- **user_id** – User that will have their nickname changed
- **thread_id** – User/Group ID to change color of. See *Threads*
- **thread_type** (ThreadType) – See *Threads*

Raises FBchatException if request failed

changeThreadColor(*color*, *thread_id=None*)
Changes thread color

Parameters

- **color** (ThreadColor) – New thread color
- **thread_id** – User/Group ID to change color of. See *Threads*

Raises FBchatException if request failed

changeThreadEmoji(*emoji*, *thread_id=None*)
Changes thread color

Trivia: While changing the emoji, the Facebook web client actually sends multiple different requests, though only this one is required to make the change

Parameters

- **color** – New thread emoji
- **thread_id** – User/Group ID to change emoji of. See *Threads*

Raises FBchatException if request failed

reactToMessage(*message_id*, *reaction*)
Reacts to a message, or removes reaction

Parameters

- **message_id** – *Message ID* to react to
- **reaction** (MessageReaction *or None*) – Reaction emoji to use, if None removes reaction

Raises FBchatException if request failed

createPlan(*plan*, *thread_id=None*)
Sets a plan

Parameters

- **plan** (Plan) – Plan to set
- **thread_id** – User/Group ID to send plan to. See *Threads*

Raises FBchatException if request failed

editPlan(*plan*, *new_plan*)
Edits a plan

Parameters

- **plan** (Plan) – Plan to edit
- **new_plan** – New plan

Raises FBchatException if request failed

deletePlan(*plan*)
Deletes a plan

Parameters **plan** – Plan to delete

Raises FBchatException if request failed

changePlanParticipation(*plan*, *take_part=True*)
Changes participation in a plan

Parameters

- **plan** – Plan to take part in or not
- **take_part** – Whether to take part in the plan

Raises FBchatException if request failed

eventReminder(*thread_id*, *time*, *title*, *location=''*, *location_id=''*)
Deprecated. Use *fbchat.Client.createPlan* instead

createPoll(*poll*, *thread_id=None*)
Creates poll in a group thread

Parameters

- **poll** (Poll) – Poll to create
- **thread_id** – User/Group ID to create poll in. See *Threads*

Raises FBchatException if request failed

updatePollVote(*poll_id*, *option_ids=[]*, *new_options=[]*)
Updates a poll vote

Parameters

- **poll_id** – ID of the poll to update vote
- **option_ids** – List of the option IDs to vote
- **new_options** – List of the new option names
- **thread_id** – User/Group ID to change status in. See *Threads*
- **thread_type** (ThreadType) – See *Threads*

Raises FBchatException if request failed

setTypingStatus(*status*, *thread_id=None*, *thread_type=None*)
Sets users typing status in a thread

Parameters

- **status** (TypingStatus) – Specify the typing status
- **thread_id** – User/Group ID to change status in. See *Threads*
- **thread_type** (ThreadType) – See *Threads*

Raises FBchatException if request failed

markAsDelivered(*thread_id*, *message_id*)
Mark a message as delivered

Parameters

- **thread_id** – User/Group ID to which the message belongs. See *Threads*
- **message_id** – Message ID to set as delivered. See *Threads*

Returns True

Raises FBchatException if request failed

markAsRead(*thread_ids=None*)
Mark threads as read All messages inside the threads will be marked as read

Parameters **thread_ids** – User/Group IDs to set as read. See *Threads*

Raises FBchatException if request failed

markAsUnread(*thread_ids=None*)
Mark threads as unread All messages inside the threads will be marked as unread

Parameters **thread_ids** – User/Group IDs to set as unread. See *Threads*

Raises FBchatException if request failed

markAsSeen()

Todo: Documenting this

---

friendConnect(*friend_id*)

Todo: Documenting this

---

removeFriend(*friend_id=None*)
Removes a specifed friend from your friend list

Parameters **friend_id** – The ID of the friend that you want to remove

Returns True

Raises FBchatException if request failed

blockUser(*user_id*)
Blocks messages from a specifed user

Parameters **user_id** – The ID of the user that you want to block

Returns True

Raises FBchatException if request failed

unblockUser(*user_id*)
Unblocks messages from a blocked user

Parameters **user_id** – The ID of the user that you want to unblock

Returns Whether the request was successful

Raises FBchatException if request failed

moveThreads(*location*, *thread_ids*)
Moves threads to specifed location

Parameters

- **location** – ThreadLocation: INBOX, PENDING, ARCHIVED or OTHER
- **thread_ids** – Thread IDs to move. See *Threads*

Returns True

Raises FBchatException if request failed

deleteThreads(*thread_ids*)
Deletes threads

Parameters **thread_ids** – Thread IDs to delete. See *Threads*

Returns True

Raises FBchatException if request failed

markAsSpam(*thread_id=None*)
Mark a thread as spam and delete it

Parameters **thread_id** – User/Group ID to mark as spam. See *Threads*

Returns True

Raises FBchatException if request failed

deleteMessages(*message_ids*)
Deletes specifed messages

Parameters **message_ids** – Message IDs to delete

Returns True

Raises FBchatException if request failed

muteThread(*mute_time=-1*, *thread_id=None*)
Mutes thread

Parameters

- **mute_time** – Mute time in seconds, leave blank to mute forever
- **thread_id** – User/Group ID to mute. See *Threads*

unmuteThread(*thread_id=None*)
Unmutes thread

Parameters **thread_id** – User/Group ID to unmute. See *Threads*

muteThreadReactions(*mute=True*, *thread_id=None*)
Mutes thread reactions

Parameters

- **mute** – Boolean. True to mute, False to unmute
- **thread_id** – User/Group ID to mute. See *Threads*

unmuteThreadReactions(*thread_id=None*)
Unmutes thread reactions

Parameters **thread_id** – User/Group ID to unmute. See *Threads*

muteThreadMentions(*mute=True*, *thread_id=None*)
Mutes thread mentions

Parameters

- **mute** – Boolean. True to mute, False to unmute
- **thread_id** – User/Group ID to mute. See *Threads*

unmuteThreadMentions(*thread_id=None*)
Unmutes thread mentions

Parameters **thread_id** – User/Group ID to unmute. See *Threads*

startListening()
Start listening from an external event loop

Raises FBchatException if request failed

doOneListen(*markAlive=None*)
Does one cycle of the listening loop. This method is useful if you want to control fbchat from an external event loop

Warning: `markAlive` parameter is deprecated, use *`Client.setActiveStatus`* or `markAlive` parameter in *`Client.listen`* instead.

Returns Whether the loop should keep running

Return type bool

stopListening()
Cleans up the variables from startListening

listen(*markAlive=None*)
Initializes and runs the listening loop continually

Parameters **markAlive** (*bool*) – Whether this should ping the Facebook server each time the loop runs

setActiveStatus(*markAlive*)
Changes client active status while listening

Parameters **markAlive** (*bool*) – Whether to show if client is active

onLoggingIn(*email=None*)
Called when the client is logging in

Parameters **email** – The email of the client

on2FACode()
Called when a 2FA code is needed to progress

onLoggedIn(*email=None*)
Called when the client is successfully logged in

Parameters **email** – The email of the client

onListening()
Called when the client is listening

onListenError(*exception=None*)
Called when an error was encountered while listening

Parameters **exception** – The exception that was encountered

Returns Whether the loop should keep running

onMessage(*mid=None*, *author_id=None*, *message=None*, *message_object=None*, *thread_id=None*, *thread_type=ThreadType.USER*, *ts=None*, *metadata=None*, *msg=None*)
Called when the client is listening, and somebody sends a message

Parameters

- **mid** – The message ID
- **author_id** – The ID of the author
- **message** – (deprecated. Use `message_object.text` instead)
- **message_object** (`Message`) – The message (As a *`Message`* object)
- **thread_id** – Thread ID that the message was sent to. See *Threads*
- **thread_type** (`ThreadType`) – Type of thread that the message was sent to. See *Threads*
- **ts** – The timestamp of the message
- **metadata** – Extra metadata about the message
- **msg** – A full set of the data recieved

onColorChange(*mid=None*, *author_id=None*, *new_color=None*, *thread_id=None*, *thread_type=ThreadType.USER*, *ts=None*, *metadata=None*, *msg=None*)
Called when the client is listening, and somebody changes a thread's color

Parameters

- **mid** – The action ID
- **author_id** – The ID of the person who changed the color
- **new_color** (`ThreadColor`) – The new color
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (`ThreadType`) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **metadata** – Extra metadata about the action
- **msg** – A full set of the data recieved

onEmojiChange(*mid=None*, *author_id=None*, *new_emoji=None*, *thread_id=None*, *thread_type=ThreadType.USER*, *ts=None*, *metadata=None*, *msg=None*)
Called when the client is listening, and somebody changes a thread's emoji

Parameters

- **mid** – The action ID
- **author_id** – The ID of the person who changed the emoji
- **new_emoji** – The new emoji
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (`ThreadType`) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **metadata** – Extra metadata about the action

- **msg** – A full set of the data recieved

onTitleChange(*mid=None*, *author_id=None*, *new_title=None*, *thread_id=None*, *thread_type=ThreadType.USER*, *ts=None*, *metadata=None*, *msg=None*)

Called when the client is listening, and somebody changes the title of a thread

Parameters

- **mid** – The action ID
- **author_id** – The ID of the person who changed the title
- **new_title** – The new title
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **metadata** – Extra metadata about the action
- **msg** – A full set of the data recieved

onImageChange(*mid=None*, *author_id=None*, *new_image=None*, *thread_id=None*, *thread_type=ThreadType.GROUP*, *ts=None*, *msg=None*)

Called when the client is listening, and somebody changes the image of a thread

Parameters

- **mid** – The action ID
- **author_id** – The ID of the person who changed the image
- **new_image** – The ID of the new image
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **msg** – A full set of the data recieved

onNicknameChange(*mid=None*, *author_id=None*, *changed_for=None*, *new_nickname=None*, *thread_id=None*, *thread_type=ThreadType.USER*, *ts=None*, *metadata=None*, *msg=None*)

Called when the client is listening, and somebody changes the nickname of a person

Parameters

- **mid** – The action ID
- **author_id** – The ID of the person who changed the nickname
- **changed_for** – The ID of the person whom got their nickname changed
- **new_nickname** – The new nickname
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **metadata** – Extra metadata about the action
- **msg** – A full set of the data recieved

onAdminAdded(*mid=None*, *added_id=None*, *author_id=None*, *thread_id=None*, *thread_type=ThreadType.GROUP*, *ts=None*, *msg=None*)
Called when the client is listening, and somebody adds an admin to a group thread

Parameters

- **mid** – The action ID
- **added_id** – The ID of the admin who got added
- **author_id** – The ID of the person who added the admins
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **msg** – A full set of the data recieved

onAdminRemoved(*mid=None*, *removed_id=None*, *author_id=None*, *thread_id=None*, *thread_type=ThreadType.GROUP*, *ts=None*, *msg=None*)
Called when the client is listening, and somebody removes an admin from a group thread

Parameters

- **mid** – The action ID
- **removed_id** – The ID of the admin who got removed
- **author_id** – The ID of the person who removed the admins
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **msg** – A full set of the data recieved

onApprovalModeChange(*mid=None*, *approval_mode=None*, *author_id=None*, *thread_id=None*, *thread_type=ThreadType.GROUP*, *ts=None*, *msg=None*)
Called when the client is listening, and somebody changes approval mode in a group thread

Parameters

- **mid** – The action ID
- **approval_mode** – True if approval mode is activated
- **author_id** – The ID of the person who changed approval mode
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **msg** – A full set of the data recieved

onMessageSeen(*seen_by=None*, *thread_id=None*, *thread_type=ThreadType.USER*, *seen_ts=None*, *ts=None*, *metadata=None*, *msg=None*)
Called when the client is listening, and somebody marks a message as seen

Parameters

- **seen_by** – The ID of the person who marked the message as seen
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **seen_ts** – A timestamp of when the person saw the message
- **ts** – A timestamp of the action

- **metadata** – Extra metadata about the action
- **msg** – A full set of the data recieved

onMessageDelivered(*msg_ids=None*, *delivered_for=None*, *thread_id=None*, *thread_type=ThreadType.USER*, *ts=None*, *metadata=None*, *msg=None*)
Called when the client is listening, and somebody marks messages as delivered

Parameters

- **msg_ids** – The messages that are marked as delivered
- **delivered_for** – The person that marked the messages as delivered
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **metadata** – Extra metadata about the action
- **msg** – A full set of the data recieved

onMarkedSeen(*threads=None*, *seen_ts=None*, *ts=None*, *metadata=None*, *msg=None*)
Called when the client is listening, and the client has successfully marked threads as seen

Parameters

- **threads** – The threads that were marked
- **author_id** – The ID of the person who changed the emoji
- **seen_ts** – A timestamp of when the threads were seen
- **ts** – A timestamp of the action
- **metadata** – Extra metadata about the action
- **msg** – A full set of the data recieved

onMessageUnsent(*mid=None*, *author_id=None*, *thread_id=None*, *thread_type=None*, *ts=None*, *msg=None*)
Called when the client is listening, and someone unsends (deletes for everyone) a message

Parameters

- **mid** – ID of the unsent message
- **author_id** – The ID of the person who unsent the message
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **msg** – A full set of the data recieved

onPeopleAdded(*mid=None*, *added_ids=None*, *author_id=None*, *thread_id=None*, *ts=None*, *msg=None*)
Called when the client is listening, and somebody adds people to a group thread

Parameters

- **mid** – The action ID
- **added_ids** – The IDs of the people who got added
- **author_id** – The ID of the person who added the people

- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **msg** – A full set of the data recieved

onPersonRemoved(*mid=None*, *removed_id=None*, *author_id=None*, *thread_id=None*, *ts=None*, *msg=None*)

Called when the client is listening, and somebody removes a person from a group thread

Parameters

- **mid** – The action ID
- **removed_id** – The ID of the person who got removed
- **author_id** – The ID of the person who removed the person
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **msg** – A full set of the data recieved

onFriendRequest(*from_id=None*, *msg=None*)

Called when the client is listening, and somebody sends a friend request

Parameters

- **from_id** – The ID of the person that sent the request
- **msg** – A full set of the data recieved

onInbox(*unseen=None*, *unread=None*, *recent_unread=None*, *msg=None*)

Todo: Documenting this

Parameters

- **unseen** – –
- **unread** – –
- **recent_unread** – –
- **msg** – A full set of the data recieved

onTyping(*author_id=None*, *status=None*, *thread_id=None*, *thread_type=None*, *msg=None*)

Called when the client is listening, and somebody starts or stops typing into a chat

Parameters

- **author_id** – The ID of the person who sent the action
- **status** – The typing status
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (`ThreadType`) – Type of thread that the action was sent to. See *Threads*
- **msg** – A full set of the data recieved

onGamePlayed(*mid=None*, *author_id=None*, *game_id=None*, *game_name=None*, *score=None*, *leaderboard=None*, *thread_id=None*, *thread_type=None*, *ts=None*, *metadata=None*, *msg=None*)

Called when the client is listening, and somebody plays a game

Parameters

- **mid** – The action ID
- **author_id** – The ID of the person who played the game
- **game_id** – The ID of the game
- **game_name** – Name of the game
- **score** – Score obtained in the game
- **leaderboard** – Actual leaderboard of the game in the thread
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **metadata** – Extra metadata about the action
- **msg** – A full set of the data recieved

onReactionAdded(*mid=None*, *reaction=None*, *author_id=None*, *thread_id=None*, *thread_type=None*, *ts=None*, *msg=None*)

Called when the client is listening, and somebody reacts to a message

Parameters

- **mid** – Message ID, that user reacted to
- **reaction** (MessageReaction) – Reaction
- **add_reaction** – Whether user added or removed reaction
- **author_id** – The ID of the person who reacted to the message
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **msg** – A full set of the data recieved

onReactionRemoved(*mid=None*, *author_id=None*, *thread_id=None*, *thread_type=None*, *ts=None*, *msg=None*)

Called when the client is listening, and somebody removes reaction from a message

Parameters

- **mid** – Message ID, that user reacted to
- **author_id** – The ID of the person who removed reaction
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **msg** – A full set of the data recieved

onBlock(*author_id=None*, *thread_id=None*, *thread_type=None*, *ts=None*, *msg=None*)
Called when the client is listening, and somebody blocks client

Parameters

- **author_id** – The ID of the person who blocked
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **msg** – A full set of the data recieved

onUnblock(*author_id=None*, *thread_id=None*, *thread_type=None*, *ts=None*, *msg=None*)
Called when the client is listening, and somebody blocks client

Parameters

- **author_id** – The ID of the person who unblocked
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **msg** – A full set of the data recieved

onLiveLocation(*mid=None*, *location=None*, *author_id=None*, *thread_id=None*, *thread_type=None*, *ts=None*, *msg=None*)
Called when the client is listening and somebody sends live location info

Parameters

- **mid** – The action ID
- **location** (LiveLocationAttachment) – Sent location info
- **author_id** – The ID of the person who sent location info
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **msg** – A full set of the data recieved

onCallStarted(*mid=None*, *caller_id=None*, *is_video_call=None*, *thread_id=None*, *thread_type=None*, *ts=None*, *metadata=None*, *msg=None*)

Todo: Make this work with private calls

Called when the client is listening, and somebody starts a call in a group

Parameters

- **mid** – The action ID
- **caller_id** – The ID of the person who started the call
- **is_video_call** – True if it's video call
- **thread_id** – Thread ID that the action was sent to. See *Threads*

- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **metadata** – Extra metadata about the action
- **msg** – A full set of the data recieved

onCallEnded(*mid=None*, *caller_id=None*, *is_video_call=None*, *call_duration=None*, *thread_id=None*, *thread_type=None*, *ts=None*, *metadata=None*, *msg=None*)

Todo: Make this work with private calls

Called when the client is listening, and somebody ends a call in a group

Parameters

- **mid** – The action ID
- **caller_id** – The ID of the person who ended the call
- **is_video_call** – True if it was video call
- **call_duration** – Call duration in seconds
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **metadata** – Extra metadata about the action
- **msg** – A full set of the data recieved

onUserJoinedCall(*mid=None*, *joined_id=None*, *is_video_call=None*, *thread_id=None*, *thread_type=None*, *ts=None*, *metadata=None*, *msg=None*)

Called when the client is listening, and somebody joins a group call

Parameters

- **mid** – The action ID
- **joined_id** – The ID of the person who joined the call
- **is_video_call** – True if it's video call
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **metadata** – Extra metadata about the action
- **msg** – A full set of the data recieved

onPollCreated(*mid=None*, *poll=None*, *author_id=None*, *thread_id=None*, *thread_type=None*, *ts=None*, *metadata=None*, *msg=None*)

Called when the client is listening, and somebody creates a group poll

Parameters

- **mid** – The action ID
- **poll** (Poll) – Created poll

- **author_id** – The ID of the person who created the poll
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **metadata** – Extra metadata about the action
- **msg** – A full set of the data recieved

onPollVoted(*mid=None*, *poll=None*, *added_options=None*, *removed_options=None*, *author_id=None*, *thread_id=None*, *thread_type=None*, *ts=None*, *metadata=None*, *msg=None*)

Called when the client is listening, and somebody votes in a group poll

Parameters

- **mid** – The action ID
- **poll** (Poll) – Poll, that user voted in
- **author_id** – The ID of the person who voted in the poll
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **metadata** – Extra metadata about the action
- **msg** – A full set of the data recieved

onPlanCreated(*mid=None*, *plan=None*, *author_id=None*, *thread_id=None*, *thread_type=None*, *ts=None*, *metadata=None*, *msg=None*)

Called when the client is listening, and somebody creates a plan

Parameters

- **mid** – The action ID
- **plan** (Plan) – Created plan
- **author_id** – The ID of the person who created the plan
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **metadata** – Extra metadata about the action
- **msg** – A full set of the data recieved

onPlanEnded(*mid=None*, *plan=None*, *thread_id=None*, *thread_type=None*, *ts=None*, *metadata=None*, *msg=None*)

Called when the client is listening, and a plan ends

Parameters

- **mid** – The action ID
- **plan** (Plan) – Ended plan
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*

- **ts** – A timestamp of the action
- **metadata** – Extra metadata about the action
- **msg** – A full set of the data recieved

onPlanEdited(*mid=None*, *plan=None*, *author_id=None*, *thread_id=None*, *thread_type=None*, *ts=None*, *metadata=None*, *msg=None*)

Called when the client is listening, and somebody edits a plan

Parameters

- **mid** – The action ID
- **plan** (Plan) – Edited plan
- **author_id** – The ID of the person who edited the plan
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **metadata** – Extra metadata about the action
- **msg** – A full set of the data recieved

onPlanDeleted(*mid=None*, *plan=None*, *author_id=None*, *thread_id=None*, *thread_type=None*, *ts=None*, *metadata=None*, *msg=None*)

Called when the client is listening, and somebody deletes a plan

Parameters

- **mid** – The action ID
- **plan** (Plan) – Deleted plan
- **author_id** – The ID of the person who deleted the plan
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action
- **metadata** – Extra metadata about the action
- **msg** – A full set of the data recieved

onPlanParticipation(*mid=None*, *plan=None*, *take_part=None*, *author_id=None*, *thread_id=None*, *thread_type=None*, *ts=None*, *metadata=None*, *msg=None*)

Called when the client is listening, and somebody takes part in a plan or not

Parameters

- **mid** – The action ID
- **plan** (Plan) – Plan
- **take_part** (*bool*) – Whether the person takes part in the plan or not
- **author_id** – The ID of the person who will participate in the plan or not
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (ThreadType) – Type of thread that the action was sent to. See *Threads*
- **ts** – A timestamp of the action

- **metadata** – Extra metadata about the action
- **msg** – A full set of the data recieved

onQprimer(*ts=None*, *msg=None*)
Called when the client just started listening

Parameters

- **ts** – A timestamp of the action
- **msg** – A full set of the data recieved

onChatTimestamp(*buddylist=None*, *msg=None*)
Called when the client receives chat online presence update

Parameters

- **buddylist** – A list of dicts with friend id and last seen timestamp
- **msg** – A full set of the data recieved

onBuddylistOverlay(*statuses=None*, *msg=None*)
Called when the client is listening and client receives information about friend active status

Parameters

- **statuses** (*dict*) – Dictionary with user IDs as keys and *ActiveStatus* as values
- **msg** – A full set of the data recieved

onUnknownMesssageType(*msg=None*)
Called when the client is listening, and some unknown data was recieved

Parameters **msg** – A full set of the data recieved

onMessageError(*exception=None*, *msg=None*)
Called when an error was encountered while parsing recieved data

Parameters

- **exception** – The exception that was encountered
- **msg** – A full set of the data recieved

1.5.2 Threads

class fbchat.**Thread**
Represents a Facebook thread

uid = None
The unique identifier of the thread. Can be used a `thread_id`. See *Threads* for more info

type = None
Specifies the type of thread. Can be used a `thread_type`. See *Threads* for more info

photo = None
A url to the thread's picture

name = None
The name of the thread

last_message_timestamp = None
Timestamp of last message

message_count = None
Number of messages in the thread

plan = None
Set *Plan*

class fbchat.**ThreadType**(*Enum*)
Used to specify what type of Facebook thread is being used. See *Threads* for more info

USER = 1

GROUP = 2

ROOM = 2

PAGE = 3

class fbchat.**Page**
Represents a Facebook page. Inherits *Thread*

url = None
The page's custom url

city = None
The name of the page's location city

likes = None
Amount of likes the page has

sub_title = None
Some extra information about the page

category = None
The page's category

class fbchat.**User**
Represents a Facebook user. Inherits *Thread*

url = None
The profile url

first_name = None
The users first name

last_name = None
The users last name

is_friend = None
Whether the user and the client are friends

gender = None
The user's gender

affinity = None
From 0 to 1. How close the client is to the user

nickname = None
The user's nickname

own_nickname = None
The clients nickname, as seen by the user

color = None
A *ThreadColor*. The message color

emoji = None
The default emoji

class fbchat.**Group**
Represents a Facebook group. Inherits *Thread*

participants = None
Unique list (set) of the group thread's participant user IDs

nicknames = None
A dict, containing user nicknames mapped to their IDs

color = None
A *ThreadColor*. The groups's message color

emoji = None
The groups's default emoji

1.5.3 Messages

class fbchat.**Message**(*text=None*, *mentions=NOTHING*, *emoji_size=None*, *sticker=None*, *attachments=NOTHING*, *quick_replies=NOTHING*, *reply_to_id=None*)
Represents a Facebook message

text = None
The actual message

mentions = None
A list of *Mention* objects

emoji_size = None
A *EmojiSize*. Size of a sent emoji

uid = None
The message ID

author = None
ID of the sender

timestamp = None
Timestamp of when the message was sent

is_read = None
Whether the message is read

read_by = None
A list of pepole IDs who read the message, works only with *fbchat.Client.fetchThreadMessages*

reactions = None
A dict with user's IDs as keys, and their *MessageReaction* as values

sticker = None
A *Sticker*

attachments = None
A list of attachments

quick_replies = None
A list of *QuickReply*

unsent = None
Whether the message is unsent (deleted for everyone)

reply_to_id = None
Message ID you want to reply to

replied_to = None
Replied message

forwarded = None
Whether the message was forwarded

classmethod formatMentions(*text*, **args*, ***kwargs*)
Like `str.format`, but takes tuples with a thread id and text instead.

Returns a *Message* object, with the formatted string and relevant mentions.

```
>>> Message.formatMentions("Hey {!r}! My name is {}", ("1234", "Peter"), (
↪"4321", "Michael"))
<Message (None): "Hey 'Peter'! My name is Michael", mentions=[<Mention 1234:␣
↪offset=4 length=7>, <Mention 4321: offset=24 length=7>] emoji_size=None␣
↪attachments=[]>
```

```
>>> Message.formatMentions("Hey {p}! My name is {}", ("1234", "Michael"), p=(
↪"4321", "Peter"))
<Message (None): 'Hey Peter! My name is Michael', mentions=[<Mention 4321:␣
↪offset=4 length=5>, <Mention 1234: offset=22 length=7>] emoji_size=None␣
↪attachments=[]>
```

class fbchat.**Mention**(*thread_id*, *offset=0*, *length=10*)
Represents a @mention

thread_id = None
The thread ID the mention is pointing at

offset = None
The character where the mention starts

length = None
The length of the mention

class fbchat.**EmojiSize**(*Enum*)
Used to specify the size of a sent emoji

LARGE = '369239383222810'

MEDIUM = '369239343222814'

SMALL = '369239263222822'

class fbchat.**MessageReaction**(*Enum*)
Used to specify a message reaction

LOVE = ''

SMILE = ''

WOW = ''

SAD = ''

ANGRY = ''

YES = ''

NO = ''

1.5.4 Exceptions

exception fbchat.**FBchatException**
Custom exception thrown by fbchat. All exceptions in the fbchat module inherits this

exception fbchat.**FBchatFacebookError**

fb_error_code = None
The error code that Facebook returned

fb_error_message = None
The error message that Facebook returned (In the user's own language)

request_status_code = None
The status code that was sent in the http response (eg. 404) (Usually only set if not successful, aka. not 200)

exception fbchat.**FBchatUserError**
Thrown by fbchat when wrong values are entered

1.5.5 Attachments

class fbchat.**Attachment**
Represents a Facebook attachment

uid = None
The attachment ID

class fbchat.**ShareAttachment**
Represents a shared item (eg. URL) that has been sent as a Facebook attachment

author = None
ID of the author of the shared post

url = None
Target URL

original_url = None
Original URL if Facebook redirects the URL

title = None
Title of the attachment

description = None
Description of the attachment

source = None
Name of the source

image_url = None
URL of the attachment image

original_image_url = None
URL of the original image if Facebook uses `safe_image`

image_width = None
Width of the image

image_height = None
Height of the image

attachments = None
List of additional attachments

class fbchat.**Sticker**
Represents a Facebook sticker that has been sent to a thread as an attachment

pack = None
The sticker-pack's ID

is_animated = None
Whether the sticker is animated

medium_sprite_image = None
URL to a medium spritemap

large_sprite_image = None
URL to a large spritemap

frames_per_row = None
The amount of frames present in the spritemap pr. row

frames_per_col = None
The amount of frames present in the spritemap pr. coloumn

frame_rate = None
The frame rate the spritemap is intended to be played in

url = None
URL to the sticker's image

width = None
Width of the sticker

height = None
Height of the sticker

label = None
The sticker's label/name

class fbchat.**LocationAttachment**
Represents a user location

Latitude and longitude OR address is provided by Facebook

latitude = None
Latitude of the location

longitude = None
Longitude of the location

image_url = None
URL of image showing the map of the location

image_width = None
Width of the image

image_height = None
Height of the image

url = None
URL to Bing maps with the location

class fbchat.**LiveLocationAttachment**
Represents a live user location

name = None
Name of the location

expiration_time = None
Timestamp when live location expires

is_expired = None
True if live location is expired

class fbchat.**FileAttachment**
Represents a file that has been sent as a Facebook attachment

url = None
Url where you can download the file

size = None
Size of the file in bytes

name = None
Name of the file

is_malicious = None
Whether Facebook determines that this file may be harmful

class fbchat.**AudioAttachment**
Represents an audio file that has been sent as a Facebook attachment

filename = None
Name of the file

url = None
Url of the audio file

duration = None
Duration of the audioclip in milliseconds

audio_type = None
Audio type

class fbchat.**ImageAttachment**
Represents an image that has been sent as a Facebook attachment

To retrieve the full image url, use: *fbchat.Client.fetchImageUrl*, and pass it the uid of the image attachment

original_extension = None
The extension of the original image (eg. 'png')

width = None
Width of original image

height = None
Height of original image

is_animated = None
Whether the image is animated

thumbnail_url = None
URL to a thumbnail of the image

preview_url = None
URL to a medium preview of the image

preview_width = None
Width of the medium preview image

preview_height = None
Height of the medium preview image

large_preview_url = None
URL to a large preview of the image

large_preview_width = None
Width of the large preview image

large_preview_height = None
Height of the large preview image

animated_preview_url = None
URL to an animated preview of the image (eg. for gifs)

animated_preview_width = None
Width of the animated preview image

animated_preview_height = None
Height of the animated preview image

class fbchat.**VideoAttachment**
Represents a video that has been sent as a Facebook attachment

size = None
Size of the original video in bytes

width = None
Width of original video

height = None
Height of original video

duration = None
Length of video in milliseconds

preview_url = None
URL to very compressed preview video

small_image_url = None
URL to a small preview image of the video

small_image_width = None
Width of the small preview image

small_image_height = None
Height of the small preview image

medium_image_url = None
URL to a medium preview image of the video

medium_image_width = None
Width of the medium preview image

medium_image_height = None
Height of the medium preview image

large_image_url = None
URL to a large preview image of the video

large_image_width = None
Width of the large preview image

large_image_height = None
Height of the large preview image

class fbchat.**ImageAttachment**
Represents an image that has been sent as a Facebook attachment

To retrieve the full image url, use: *fbchat.Client.fetchImageUrl*, and pass it the uid of the image attachment

original_extension = None
The extension of the original image (eg. 'png')

width = None
Width of original image

height = None
Height of original image

is_animated = None
Whether the image is animated

thumbnail_url = None
URL to a thumbnail of the image

preview_url = None
URL to a medium preview of the image

preview_width = None
Width of the medium preview image

preview_height = None
Height of the medium preview image

large_preview_url = None
URL to a large preview of the image

large_preview_width = None
Width of the large preview image

large_preview_height = None
Height of the large preview image

animated_preview_url = None
URL to an animated preview of the image (eg. for gifs)

animated_preview_width = None
Width of the animated preview image

animated_preview_height = None
Height of the animated preview image

1.5.6 Miscellaneous

class fbchat.**ThreadLocation**(*Enum*)
Used to specify where a thread is located (inbox, pending, archived, other).

INBOX = 'INBOX'

PENDING = 'PENDING'

ARCHIVED = 'ARCHIVED'

OTHER = 'OTHER'

class fbchat.**ThreadColor**(*Enum*)
Used to specify a thread colors

MESSENGER_BLUE = '#0084ff'

VIKING = '#44bec7'

GOLDEN_POPPY = '#ffc300'

RADICAL_RED = '#fa3c4c'

SHOCKING = '#d696bb'

PICTON_BLUE = '#6699cc'

FREE_SPEECH_GREEN = '#13cf13'

PUMPKIN = '#ff7e29'

LIGHT_CORAL = '#e68585'

MEDIUM_SLATE_BLUE = '#7646ff'

DEEP_SKY_BLUE = '#20cef5'

FERN = '#67b868'

CAMEO = '#d4a88c'

BRILLIANT_ROSE = '#ff5ca1'

BILOBA_FLOWER = '#a695c7'

TICKLE_ME_PINK = '#ff7ca8'

MALACHITE = '#1adb5b'

RUBY = '#f01d6a'

DARK_TANGERINE = '#ff9c19'

BRIGHT_TURQUOISE = '#0edcde'

class fbchat.**ActiveStatus**

active = None
Whether the user is active now

last_active = None
Timestamp when the user was last active

in_game = None
Whether the user is playing Messenger game now

class fbchat.**TypingStatus**(*Enum*)
Used to specify whether the user is typing or has stopped typing

STOPPED = 0

TYPING = 1

class fbchat.**QuickReply**(*payload=None*, *data=None*, *is_response=False*)
Represents a quick reply

payload = None
Payload of the quick reply

external_payload = None
External payload for responses

data = None
Additional data

is_response = None
Whether it's a response for a quick reply

class fbchat.**QuickReplyText**(*title=None*, *image_url=None*, ***kwargs*)
Represents a text quick reply

title = None
Title of the quick reply

image_url = None
URL of the quick reply image (optional)

class fbchat.**QuickReplyLocation**(***kwargs*)
Represents a location quick reply (Doesn't work on mobile)

class fbchat.**QuickReplyPhoneNumber**(*image_url=None*, ***kwargs*)
Represents a phone number quick reply (Doesn't work on mobile)

image_url = None
URL of the quick reply image (optional)

class fbchat.**QuickReplyEmail**(*image_url=None*, ***kwargs*)
Represents an email quick reply (Doesn't work on mobile)

image_url = None
URL of the quick reply image (optional)

class fbchat.**Poll**(*title*, *options*, *options_count=None*, *uid=None*)
Represents a poll

title = None
Title of the poll

options = None
List of *PollOption*, can be fetched with *fbchat.Client.fetchPollOptions*

options_count = None
Options count

uid = None
ID of the poll

class fbchat.**PollOption**(*text*, *vote=False*, *voters=None*, *votes_count=None*, *uid=None*)
Represents a poll option

text = None
Text of the poll option

vote = None
Whether vote when creating or client voted

voters = None
ID of the users who voted for this poll option

votes_count = None
Votes count

uid = None
ID of the poll option

class fbchat.**Plan**(*time*, *title*, *location=None*, *location_id=None*)
Represents a plan

uid = None
ID of the plan

time = None
Plan time (unix time stamp), only precise down to the minute

title = None
Plan title

location = None
Plan location name

location_id = None
Plan location ID

author_id = None
ID of the plan creator

guests = None
Dict of *User* IDs mapped to their *GuestStatus*

property going
List of the *User* IDs who will take part in the plan.

property declined
List of the *User* IDs who won't take part in the plan.

property invited
List of the *User* IDs who are invited to the plan.

class fbchat.**GuestStatus**(*Enum*)

INVITED = 1

GOING = 2

DECLINED = 3

1.6 Todo

This page will be periodically updated to show missing features and documentation

1.6.1 Missing Functionality

- **Implement Client.searchForMessage**
  - This will use the graphql request API

- Implement chatting with pages properly
- Write better FAQ
- Explain usage of graphql

1.6.2 Documentation

Todo: Documenting this

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/fbchat/checkouts/v1.7.1/fbchat/_client.py:docstring of fbchat.Client.markAsSeen, line 1.)

Todo: Documenting this

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/fbchat/checkouts/v1.7.1/fbchat/_client.py:docstring of fbchat.Client.friendConnect, line 1.)

Todo: Documenting this

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/fbchat/checkouts/v1.7.1/fbchat/_client.py:docstring of fbchat.Client.onInbox, line 1.)

Todo: Make this work with private calls

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/fbchat/checkouts/v1.7.1/fbchat/_client.py:docstring of fbchat.Client.onCallStarted, line 1.)

Todo: Make this work with private calls

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/fbchat/checkouts/v1.7.1/fbchat/_client.py:docstring of fbchat.Client.onCallEnded, line 1.)

1.7 FAQ

1.7.1 Version X broke my installation

We try to provide backwards compatibility where possible, but since we're not part of Facebook, most of the things may be broken at any point in time

Downgrade to an earlier version of fbchat, run this command

```
$ pip install fbchat==<X>
```

Where you replace <X> with the version you want to use

1.7.2 Will you be supporting creating posts/events/pages and so on?

We won't be focusing on anything else than chat-related things. This API is called `fbCHAT`, after all ;)

1.7.3 Submitting Issues

If you're having trouble with some of the snippets, or you think some of the functionality is broken, please feel free to submit an issue on Github. You should first login with `logging_level` set to `logging.DEBUG`:

```
from fbchat import Client
import logging
client = Client('<email>', '<password>', logging_level=logging.DEBUG)
```

Then you can submit the relevant parts of this log, and detailed steps on how to reproduce

> **Warning:** Always remove your credentials from any debug information you may provide us. Preferably, use a test account, in case you miss anything

PYTHON MODULE INDEX

f

INDEX

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

Y