fbchat Documentation

Release 1.0.21

Taehoon Kim; Moreels Pieter-Jan; Mads Marquart

Sep 27, 2017

Contents

Release v1.0.21. (*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 1

Overview

Installation

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.

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
```

Introduction

fbchat uses your email and password to communicate with the Facebook server. That means that you should always store your password in a seperate 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

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 fasion, 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()
```

Threads

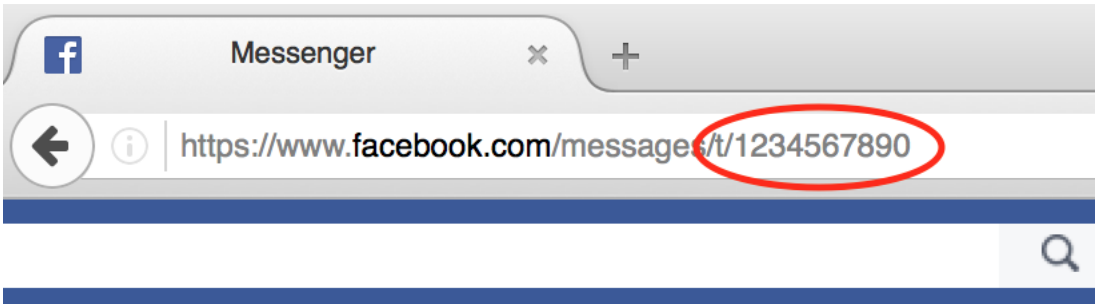
A thread can refer to two things: A Messenger group chat or a single Facebook user

models.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 differetiates 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.sendMessage('<message>', thread_id='<user id>', thread_type=ThreadType.USER)
client.sendMessage('<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>')
```

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

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.sendMessage('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*

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*

Sessions

fbchat provides functions to retrieve and set the session cookies. This will enable you to store the session cookies in a seperate 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

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, thread_id, thread_type, ts, metadata,
↪ msg, **kwargs):
        # Do something with the message 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, author_id, thread_id, thread_type, **kwargs):
        # Do something with the message here
        pass

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

The change was in the parameters that our *onMessage* method took: `message` 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

Examples

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

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.sendMessage('Hi me!', thread_id=client.uid, thread_type=ThreadType.USER)

client.logout()
```

Interacting with Threads

This will interract 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.sendMessage('<message>', thread_id=thread_id, thread_type=thread_type)

# Will send the default `like` emoji
client.sendEmoji(emoji=None, size=EmojiSize.LARGE, thread_id=thread_id, thread_
↪type=thread_type)

# Will send the emoji ``
client.sendEmoji(emoji='', size=EmojiSize.LARGE, thread_id=thread_id, thread_
↪type=thread_type)

# Will send the image located at `<image path>`
client.sendLocalImage('<image path>', message='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='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>`
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)
```

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))
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`
```

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, thread_id, thread_type, **kwargs):
        self.markAsDelivered(author_id, thread_id)
        self.markAsRead(author_id)

        log.info("Message from {} in {} ({}): {}".format(author_id, thread_id, thread_
↪type.name, message))

        # If you're not the author, echo
        if author_id != self.uid:
            self.sendMessage(message, thread_id=thread_id, thread_type=thread_type)

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

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, 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 == '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(type(self), self).onMessage(author_id=author_id, message=message,␣
↪thread_id=thread_id, thread_type=thread_type, **kwargs)

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

"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
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 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()
```

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 commandline (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)

class `tests.`**`TestFbchat`**(*methodName='runTest'*)

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

`test_changeNickname`()

`test_changeThreadColor`()

`test_changeThreadEmoji`()

test_changeThreadTitle()

test_defaultThread()

test_examples()

test_fetchAllUsers()

test_fetchInfo()

test_fetchThreadList()

test_fetchThreadMessages()

test_listen()

test_loginFunctions()

test_reactToMessage()

test_removeAddFromGroup()

test_searchFor()

test_sendEmoji()

test_sendImages()

test_sendMessage()

test_sessions()

test_setTypingStatus()

Full API

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

Client

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

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

Initializes and logs 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 (see *utils.USER_AGENTS*)
- **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 *INFO*

Raises FBchatException on failed login

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*

Returns *Message ID* of the executed action

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** (models.ThreadType) – See *Threads*

Raises FBchatException if request failed

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

Parameters

- **color** (models.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

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** (models.ThreadType) – See *Threads*

Raises FBchatException if request failed

doOneListen(*markAlive=True*)
Does one cycle of the listening loop. This method is useful if you want to control fbchat from an external event loop

Parameters **markAlive** (*bool*) – Whether this should ping the Facebook server before running

Returns Whether the loop should keep running

Return type bool

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

Returns *models.User* objects

Return type list

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 *models.Group* 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 *models.Page* 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 *models.Thread* objects, labeled by their ID

Return type dict

Raises FBchatException if request failed

fetchThreadList(*offset=0*, *limit=20*)
Get thread list of your facebook account

Parameters

- **offset** (*int*) – The offset, from where in the list to recieve threads from
- **limit** (*int*) – Max. number of threads to retrieve. Capped at 20

Returns *models.Thread* objects

Return type list

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 default to. See *Threads*
- **limit** (*int*) – Max. number of messages to retrieve
- **before** (*int*) – A timestamp, indicating from which point to retrieve messages

Returns *models.Message* objects

Return type list

Raises FBchatException if request failed

fetchUnread()

Todo

Documenting this

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 *models.User* objects, labeled by their ID

Return type dict

Raises FBchatException if request failed

friendConnect(*friend_id*)

Todo

Documenting this

---

getSession()
Retrieves session cookies

Returns A dictionay containing session cookies

Return type dict

graphql_request(*query*)
Shorthand for *graphql_requests(query)[0]*

Raises FBchatException if request failed

graphql_requests(**queries*)

Todo

Documenting this

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

listen(*markAlive=True*)
Initializes and runs the listening loop continually

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

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

login(*email*, *password*, *max_tries=5*)
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

Parameters **timeout** – See requests timeout

Returns True if the action was successful

Return type bool

markAsDelivered(*userID*, *threadID*)

Todo

Documenting this

markAsRead(*userID*)

Todo

Documenting this

markAsSeen()

Todo

Documenting this

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

onChatTimestamp(*buddylist={}*, *msg={}*)
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

onColorChange(*mid=None*, *author_id=None*, *new_color=None*, *thread_id=None*, *thread_type=ThreadType.USER*, *ts=None*, *metadata=None*, *msg={}*)
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** (models.ThreadColor) – The new color
- **thread_id** – Thread ID that the action was sent to. See *Threads*
- **thread_type** (models.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={}*)
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** (models.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

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

Todo

Documenting this

Parameters

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

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

onListening()
Called when the client is listening

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

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

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

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

onMarkedSeen(*threads=None*, *seen_ts=None*, *ts=None*, *metadata=None*, *msg={}*)
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

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

Parameters

- **mid** – The message ID
- **author_id** – The ID of the author
- **message** – The message
- **thread_id** – Thread ID that the message was sent to. See *Threads*
- **thread_type** (models.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

onMessageDelivered(*msg_ids=None*, *delivered_for=None*, *thread_id=None*, *thread_type=ThreadType.USER*, *ts=None*, *metadata=None*, *msg={}*)
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** (models.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

onMessageError(*exception=None*, *msg={}*)
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

onMessageSeen(*seen_by=None*, *thread_id=None*, *thread_type=ThreadType.USER*, *seen_ts=None*, *ts=None*, *metadata=None*, *msg={}*)
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** (models.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

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

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** (models.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

onPeopleAdded(*mid=None*, *added_ids=None*, *author_id=None*, *thread_id=None*, *ts=None*, *msg={}*)

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={}*)

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

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

Parameters

- **ts** – A timestamp of 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={}*)
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** (models.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

onUnknownMesssageType(*msg={}*)
Called when the client is listening, and some unknown data was recieved

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

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

Parameters

- **message_id** – *Message ID* to react to
- **reaction** (models.MessageReaction) – Reaction emoji to use

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

resetDefaultThread()
Resets default thread

searchForGroups(*name*, *limit=1*)
Find and get group thread by its name

Parameters

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

Returns *models.Group* objects, ordered by relevance

Return type list

Raises FBchatException if request failed

searchForPages(*name*, *limit=1*)
Find and get page by its name

Parameters **name** – Name of the page

Returns *models.Page* objects, ordered by relevance

Return type list

Raises FBchatException if request failed

searchForThreads(*name*, *limit=1*)
Find and get a thread by its name

Parameters

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

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

Return type list

Raises FBchatException if request failed

searchForUsers(*name*, *limit=1*)
Find and get user by his/her name

Parameters

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

Returns *models.User* objects, ordered by relevance

Return type list

Raises FBchatException if request failed

sendEmoji(*emoji=None*, *size=EmojiSize.SMALL*, *thread_id=None*, *thread_type=ThreadType.USER*)
Sends an emoji to a thread

Parameters

- **emoji** – The chosen emoji to send. If not specified, the default *like* emoji is sent
- **size** (models.EmojiSize) – If not specified, a small emoji is sent
- **thread_id** – User/Group ID to send to. See *Threads*
- **thread_type** (models.ThreadType) – See *Threads*

Returns *Message ID* of the sent emoji

Raises FBchatException if request failed

sendImage(*image_id*, *message=None*, *thread_id=None*, *thread_type=ThreadType.USER*)
Sends an already uploaded image to a thread. (Used by `Client.sendRemoteImage` and `Client.sendLocalImage`)

Parameters

- **image_id** – ID of an image that's already uploaded to Facebook
- **message** – Additional message
- **thread_id** – User/Group ID to send to. See *Threads*
- **thread_type** (`models.ThreadType`) – See *Threads*

Returns *Message ID* of the sent image

Raises FBchatException if request failed

sendLocalImage(*image_path*, *message=None*, *thread_id=None*, *thread_type=ThreadType.USER*)
Sends a local image to a thread

Parameters

- **image_path** – Path of an image to upload and send
- **message** – Additional message
- **thread_id** – User/Group ID to send to. See *Threads*
- **thread_type** (`models.ThreadType`) – See *Threads*

Returns *Message ID* of the sent image

Raises FBchatException if request failed

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

Parameters

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

Returns *Message ID* of the sent message

Raises FBchatException if request failed

sendRemoteImage(*image_url*, *message=None*, *thread_id=None*, *thread_type=ThreadType.USER*)
Sends an image from a URL to a thread

Parameters

- **image_url** – URL of an image to upload and send
- **message** – Additional message
- **thread_id** – User/Group ID to send to. See *Threads*
- **thread_type** (`models.ThreadType`) – See *Threads*

Returns *Message ID* of the sent image

Raises FBchatException if request failed

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** (models.ThreadType) – See *Threads*

setSession(*session_cookies*)
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

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

Parameters

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

Raises FBchatException if request failed

startListening()
Start listening from an external event loop

Raises FBchatException if request failed

stopListening()
Cleans up the variables from startListening

uid = None
The ID of the client. Can be used as *thread_id*. See *Threads* for more info.

Note: Modifying this results in undefined behaviour

Models

These models are used in various functions, both as inputs and return values. A good tip is to write `from fbchat.models import *` at the start of your source, so you can use these models freely

class fbchat.models.**EmojiSize**
Used to specify the size of a sent emoji

LARGE = '369239383222810'

MEDIUM = '369239343222814'

SMALL = '369239263222822'

class fbchat.models.**Enum**
Used internally by fbchat to support enumerations

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

exception fbchat.models.**FBchatFacebookError**(*message*, *fb_error_code=None*, *fb_error_message=None*, *request_status_code=None*)

fb_error_code
The error code that Facebook returned

alias of str

fb_error_message
The error message that Facebook returned (In the user's own language)

alias of str

request_status_code
The status code that was sent in the http response (eg. 404) (Usually only set if not successful, aka. not 200)

alias of int

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

class fbchat.models.**Group**(*uid*, *participants=set()*, *nicknames=[]*, *color=None*, *emoji=None*, ***kwargs*)
Represents a Facebook group. Inherits *Thread*

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

emoji
The groups's default emoji

alias of str

nicknames
Dict, containing user nicknames mapped to their IDs

alias of dict

participants
Unique list (set) of the group thread's participant user IDs

alias of set

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

length
The length of the mention

alias of int

offset
The character where the mention starts

alias of int

user_id
The user ID the mention is pointing at

alias of str

class fbchat.models.**Message**(*uid*, *author=None*, *timestamp=None*, *is_read=None*, *reactions=[]*, *text=None*, *mentions=[]*, *sticker=None*, *attachments=[]*, *extensible_attachment={}*)

Represents a Facebook message

attachments
A list of attachments

alias of list

author
ID of the sender

alias of int

extensible_attachment
An extensible attachment, e.g. share object

alias of dict

is_read
Whether the message is read

alias of bool

mentions
A list of *Mention* objects

alias of list

reactions
A list of message reactions

alias of list

sticker
An ID of a sent sticker

alias of str

text
The actual message

alias of str

timestamp
Timestamp of when the message was sent

alias of str

uid
The message ID

alias of str

class fbchat.models.**MessageReaction**

Used to specify a message reaction

ANGRY = ‘’

LOVE = ‘’

NO = ‘’

SAD = ‘’

SMILE = ‘’

WOW = ‘’

YES = ‘’

class fbchat.models.**Page**(*uid*, *url=None*, *city=None*, *likes=None*, *sub_title=None*, *category=None*, ***kwargs*)

Represents a Facebook page. Inherits *Thread*

category
The page’s category

alias of str

city
The name of the page’s location city

alias of str

likes
Amount of likes the page has

alias of int

sub_title
Some extra information about the page

alias of str

url
The page’s custom url

alias of str

class fbchat.models.**Thread**(*_type*, *uid*, *photo=None*, *name=None*, *last_message_timestamp=None*, *message_count=None*)

Represents a Facebook thread

last_message_timestamp
Timestamp of last message

alias of str

message_count
Number of messages in the thread

alias of int

name
The name of the thread

alias of str

photo
The thread’s picture

alias of str

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

uid
The unique identifier of the thread. Can be used a *thread_id*. See *Threads* for more info

alias of str

class fbchat.models.**ThreadColor**
Used to specify a thread colors

BILOBA_FLOWER = '#a695c7'

BRILLIANT_ROSE = '#ff5ca1'

CAMEO = '#d4a88c'

DEEP_SKY_BLUE = '#20cef5'

FERN = '#67b868'

FREE_SPEECH_GREEN = '#13cf13'

GOLDEN_POPPY = '#ffc300'

LIGHT_CORAL = '#e68585'

MEDIUM_SLATE_BLUE = '#7646ff'

MESSENGER_BLUE = ''

PICTON_BLUE = '#6699cc'

PUMPKIN = '#ff7e29'

RADICAL_RED = '#fa3c4c'

SHOCKING = '#d696bb'

VIKING = '#44bec7'

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

GROUP = 2

PAGE = 3

USER = 1

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

STOPPED = 0

TYPING = 1

class fbchat.models.**User**(*uid*, *url=None*, *first_name=None*, *last_name=None*, *is_friend=None*, *gender=None*, *affinity=None*, *nickname=None*, *own_nickname=None*, *color=None*, *emoji=None*, ***kwargs*)
Represents a Facebook user. Inherits *Thread*

affinity
From 0 to 1. How close the client is to the user

alias of float

color = None
A *ThreadColor*. The message color

emoji
The default emoji

alias of str

first_name
The users first name

alias of str

gender
The user's gender

alias of `str`

is_friend
Whether the user and the client are friends

alias of `bool`

last_name
The users last name

alias of `str`

nickname
The user's nickname

alias of `str`

own_nickname
The clients nickname, as seen by the user

alias of `str`

url
The profile url

alias of `str`

Utils

These functions and values are used internally by fbchat, and are subject to change. Do **NOT** rely on these to be backwards compatible!

class `fbchat.utils.`**ReqUrl**
A class containing all urls used by *fbchat*

`fbchat.utils.`**USER_AGENTS = ['Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_2) AppleWebKit/537.36 (KHTML, like Ge**
Default list of user agents

`fbchat.utils.`**random**`()` → x in the interval [0, 1).

Todo

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

Missing Functionality

- **Implement Client.searchForMessage**
  - This will use the graphql request API
- Implement chatting with pages properly
- Write better FAQ
- Explain usage of graphql

Documentation

Todo

Documenting this

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/fbchat/checkouts/master/fbchat/client.py:docstring of fbchat.Client.fetchUnread, line 1.)

Todo

Documenting this

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/fbchat/checkouts/master/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/master/fbchat/client.py:docstring of fbchat.Client.graphql_requests, line 1.)

Todo

Documenting this

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/fbchat/checkouts/master/fbchat/client.py:docstring of fbchat.Client.markAsDelivered, line 1.)

Todo

Documenting this

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/fbchat/checkouts/master/fbchat/client.py:docstring of fbchat.Client.markAsRead, line 1.)

Todo

Documenting this

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

FAQ

Version X broke my installation

We try to provide backwards compatability 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

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 ;)

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

t

Index

L

M

N

O

P

R

S

T

U

V

W

Y