Groupy Documentation

Release 0.3.1

Robert Grant

April 29, 2015

Contents

The simple yet powerful wrapper for the GroupMe API

CHAPTER 1

Introduction

GroupMe is a messaging app that allows you to create groups and have others join them with you. In addition to group messaging, fellow group members can be messaged directly. GroupMe is available for most platforms, lets you share links, images, and locations, and messages can be favorited (or "liked"). You can read more about GroupMe, but the best part about it is that they provide an API!

The GroupMe API is documented, but there are some notable omissions. Many of the properties of groups and messages are not documented, and some features are only hinted at by the documentation. Regardless, all of the information about your groups, their members, their messages, you, and your bots can be obtained through the GroupMe API. You can read the API documentation for more (or less) detailed information.

But Groupy lets you forget about the GroupMe API and focus on what you need to get done!

1.1 Features

- List and filter your current and former groups
- Create, destroy, and update your groups
- List and filter messages from groups and members
- Post new messages to groups
- Send direct messages to members
- List and filter the members of a group
- Like and unlike messages (even direct messages!)
- List and filter your bots
- Use your bots to post messages
- Create, update, and destroy bots
- Get your user information
- Enable and disable SMS mode

CHAPTER 2

Table of Contents

2.1 Installation

You'll need to get a GroupMe account to get started. Got it? Okay, now you'll need to obtain your access token so you can make API requests:

1. Login to the developer portal.
2. Click the "Bots" button on the top menu bar.
3. Click the "Click here to reveal" button and copy your access token.
4. Paste it into a new file called `.groupy.key` and save it in your user's home directory.

Now you're ready to install Groupy!

2.1.1 Using `pip`

Note: Installation via `pip` coming soon!

2.1.2 From Source

1. Download Groupy from GitHub.
2. Copy the package directory (`Groupy/groupy`) into your package directory for `Python3`.

If you have `git`, it's as easy as:

```
$ git clone https://github.com/rhgrant10/Groupy.git
$ cd Groupy
$ cp -r groupy /usr/lib/python3/dist-packages     # see note below
```

If you don't have `git` installed (and don't wish to install it), that's okay too! You can get the project as a zip file using `wget`:

```
$ wget https://github.com/rhgrant10/Groupy/archive/master.zip
$ unzip master.zip
$ cd Groupy-master
$ cp -r groupy /usr/lib/python3/dist-packages        # see note below
```

If neither `git` nor `wget` are on your system (for example, you might have Windows installed rather than a flavor of Linux), that's still okay! Simply click this link to download it using your browser as a zip file.

Note: See this StackOverflow question for help determining the right location.

There, all done! Feels good, right?

2.1.3 Verifying Installation

You can verify that the installation worked by simply importing `groupy` from a `Python3` shell:

If you get no errors, you're good to go!

2.2 Quickstart

2.2.1 Who Am I?

One of the most basic pieces of information you'll want to obtain is your own! Groupy makes this very simple:

```
>>> from groupy import User
>>> your = User.get()
>>> print(your.nickname)
Fred
>>> print(your.user_id)
1234567890
>>> print(your.email)
fred.no@email.com
>>>
```

2.2.2 The Order of Things

Not all objects are reported equally! For example, although you can list all of your groups, there is *no direct way* to list all of the members. This is because of the way in which the API is structured. Don't worry, it's not complicated, but some find it surprising, so we'll just get this part out of the way!

Groups are the starting point for everything.

Members are listed from groups. There is no global members list! The API doesn't know about members that are not in at least one of the groups you're in currently or were in previously. In other words, if you've never "seen" the member in a group, you'd have to know their user id upfront to direct message them or add them to a group.

Group messages are listed from groups, and direct (personal) messages are listed from (you guessed it) members. Messages come in pages, with the most recent page being the default page. Groupy has an easy way to page through the messages or to collect an entire message history.

2.2.3 Fetching Stuff

Mainly, you'll want to get a list of groups, messages, or members. With Groupy, this is easy:

```
from groupy import Group

# List current groups
groups = Group.list()
```

```
# List the members and messages of a single group
group = groups[0]
members = group.members()
messages = group.messages()
```

Now `members` contains all the members of `group`, and `messages` contains the most recent page of messages in `group`. Messages can also be listed from a member:

```
member = members[0]
direct_messages = member.messages()
```

Now `direct_messages` contains the most recent page of messages between you and `member`. Easy right? But what if you wanted to get older messages? Well, that's easy too:

```
older_messages = messages.older()
```

`older_messages` now contains the page of messages that preceeds `messsages`. In other words, the newest message in `older_messages` immediately preceeds the oldest message in `messages`. This can be checked quite easily with Groupy:

```
# Make a "timeline" using the oldest and newest messages from each page.
timeline = [older_messages.oldest, older_messages.newest,
            messages.oldest, messages.newest]

# Define a function to pluck timestamp information from a message.
def get_timestamp(msg):
    return msg.created_at

# Map the function to the timeline.
timestamps = list(map(get_timestamp, timeline))

# This should not raise an AssertionError!
assert sorted(timeline) == timeline
```

But what if we wanted to get all of the messages since the beginning of a group? Well, that's also easy:

```
# Get the most recent page of messages.
messages = group.messages()

# Extend the page with additional pages of messages.
while messages.iolder():
    pass
```

It may take a while, depending on your connection speed and number of messages in the group, but now `messages` contains all of the messages from the group. `iolder` is the "in-place" version of `older`, and works by fetching the page of messages that preceeds the oldest message it has and adding them to the list in-place.

Lastly, `older` and `iolder` have counter-parts `newer` and `inewer` for fetching newer messages. That means checking for new messages is as easy as:

```
messages = group.messages()
count = len(messages)

messages.inewer()
if len(messages) > count:
    # New messages have arrived... do stuff
```

2.2.4 Messaging

Let's face it: sometimes we just want to send a message. Messages can be sent to both groups and members! To message a group:

```
group.post("Hello world")
```

Super easy, right? What about messaging a member? Also easy:

```
member.post("Hello... person")
```

There is another fact of life we must face: sometimes you like messages. We all do it; how hard can it be? Not hard:

```
message.like()
```

What if we made a mistake and decide we don't like the message after all? Not a problem:

```
message.unlike()
```

Note that both `like` and `unlike` return `True` if the action was successful:

```
if message.like():
    # success
else:
    # Uh-oh...
```

What about finding out who has already liked a message? Likes are reported as a list of members:

```
favorited_by = message.likes()
```

Now `favorited_by` is a list of the members who liked the message.

2.2.5 Groups and Members

Members can be added and removed from groups. Adding one or multiple members to a group is quite intuitive:

```
# Add one member
group.add(member)

# Add several members
group.add(*members)
```

Removing members is done one at a time:

```
# Remove one member
group.remove(member)

# Remove several members
for m in members:
  group.remove(m)
```

2.3 Advanced Usage

More docs coming soon =)

2.4 API Docs

2.4.1 groupy.config

groupy.config.**API_URL = 'https://api.groupme.com/v3'**
The URL for the GroupMe API

groupy.config.**IMAGE_API_URL = 'https://image.groupme.com'**
The URL for the GroupMe Image Service API

groupy.config.**KEY_LOCATION = '~/.groupy.key'**
Full path to the file in which your access token can be found

2.4.2 groupy.api

class groupy.api.**Endpoint**
An API endpoint capable of building a url and extracting data from the response.

This class serves as the base class for all of the API endpoints.

classmethod build_url(*path=None*, **args*)
Build and return a url extended with *path* and filled in with *args*.

Parameters

- **path** (*str*) – a suffix for the final URL. If *args* are present, this should be a python format string pertaining to the given *args*.
- **args** (list) – a list of arguments for the format string *path*.

Returns a complete URL

Return type str

static clamp(*value*, *lower*, *upper*)
Utility method for clamping a *value* between a *lower* and an *upper* value.

Parameters

- **value** – the value to clamp
- **lower** – the "smallest" possible value
- **upper** – the "largest" possible value

Returns *value* such that `lower <= value <= upper`

classmethod response(*r*)
Extract the data from the API response *r*.

This method essentially strips the actual response of the envelope.

Parameters **r** (*requests.Response*) – the HTTP response from an API call

Returns API response data

Return type json

class groupy.api.**Groups**
Endpoint for the groups API.

Groups can be listed, loaded, created, updated, and destroyed.

classmethod **create**(*name*, *description=None*, *image_url=None*, *share=True*)
Create a new group.

Parameters

- **name** (*str*) – the name of the new group
- **description** (*str*) – the description of the new group
- **image_url** (*str*) – the group avatar image as a GroupMe image URL
- **share** (`bool`) – `True` if a link to join should be generated, `False` otherwise

Returns the new group

Return type `dict`

classmethod **destroy**(*group_id*)
Destroy (or leave) a group.

Note: If you are not the owner of a group, you can not destroy it.

Parameters **group_id** (*str*) – the ID of the group to destroy/leave

classmethod **index**(*page=1*, *per_page=500*, *former=False*)
Return a list of groups.

Parameters

- **page** (*int*) – the page of groups to return
- **per_page** (*int*) – the number of groups in the page
- **former** (`bool`) – `True` if former groups should be listed instead of current groups, `False` otherwise

Returns a list of groups

Return type `list`

classmethod **show**(*group_id*)
Return a specific group by its *group_id*.

Parameters **group_id** (*str*) – the ID of the group to show.

Returns the group with the given *group_id*

Return type `dict`

classmethod **update**(*group_id*, *name=None*, *description=None*, *share=None*, *image_url=None*)
Update the information for a group.

Parameters

- **group_id** (*str*) – the ID of the group to update
- **name** (*str*) – the new name of the group
- **description** (*str*) – the new description of the group
- **share** (`bool`) – True if a share link should be generated, False otherwise
- **image_url** (*str*) – the GroupMe image URL for the new group avatar.

Returns the modified group

Return type `dict`

class groupy.api.**Members**
Endpoint for the members API.

Members can be added and removed from a group, and the results of adding members can be obtained.

classmethod add(*group_id*, **members*)
Add one or more members to a group.

Parameters

- **group_id** (*str*) – the ID of the group to which the members should be added
- **members** (list) – the members to add.

Returns the results_id for this request

Return type dict

classmethod remove(*group_id*, *member_id*)
Remove a member from a group.

Parameters

- **group_id** (*str*) – the ID of the group from which the member should be removed
- **member_id** (*str*) – the ID of the member to remove

classmethod results(*group_id*, *result_id*)
Check the result of adding one or more members.

Parameters

- **group_id** (*str*) – the ID of the group to which the add call was made
- **result_id** (*str*) – the GUID returned by the add call

Returns a list of successfully added members

Return type list

class groupy.api.**Messages**
Endpoint for the messages API.

Messages can be listed and created.

classmethod create(*group_id*, *text*, **attachments*)
Create a new message in a group.

All messages must have either text or one attachment. Note that while the API provides for an unlimited number of attachments, most clients can only handle one of each attachment type (location, image, split, or emoji).

Parameters

- **group_id** (*str*) – the ID of the group in which to create the message
- **text** (*str*) – the text of the message
- **attachments** (list) – a list of attachments to include

Returns the created message

Return type dict

classmethod index(*group_id*, *before_id=None*, *since_id=None*, *after_id=None*, *limit=100*)
List the messages from a group.

Listing messages gives the most recent 100 by default. Additional messages can be obtained by specifying a reference message, thereby facilitating paging through messages.

Use `before_id` and `after_id` to "page" through messages. `since_id` is odd in that it returns the *most recent* messages since the reference message, which means there may be messages missing between the reference message and the oldest message in the returned list of messages.

Note: Only one of `before_id`, `after_id`, or `since_id` can be specified in a single call.

Parameters

- **group_id** (*str*) – the ID of the group from which to list messages
- **before_id** (*str*) – a reference message ID; specify this to list messages just prior to it
- **since_id** (*str*) – a reference message ID; specify this to list the *most recent* messages after it (**not** the messages right after the reference message)
- **after_id** (*str*) – a reference message ID; specifying this will return the messages just after the reference message
- **limit** (*int*) – a limit on the number of messages returned (between 1 and 100 inclusive)

Returns a `dict` containing `count` and `messages`

Return type `dict`

class `groupy.api.`**`DirectMessages`**
Endpoint for the direct message API.

classmethod **`create`**(*recipient_id*, *text*, **attachments*)
Create a direct message to a recipient user.

Parameters

- **recipient_id** (*str*) – the ID of the recipient
- **text** (*str*) – the message text
- **attachments** (`list`) – a list of attachments to include

Returns the created direct message

Return type `dict`

classmethod **`index`**(*other_user_id*, *before_id=None*, *since_id=None*, *after_id=None*)
List the direct messages with another user.

Parameters

- **other_user_id** (*str*) – the ID of the other party
- **before_id** (*str*) – a reference message ID; specify this to list messages prior to it

Returns a list of direct messages

Return type `list`

class `groupy.api.`**`Likes`**
Endpoint for the likes API.

Likes can be created or destroyed.

Note: The `conversation_id` is poorly documented. For messages in a group, it corresponds to the

`group_id` (or `id` since they seem to always be identical). For direct messages, it corresponds to the `user_id` of both conversation participants sorted lexicographically and concatenated with a plus sign ("+").

---

classmethod **`create`**(*conversation_id*, *message_id*)
Like a message.

Parameters

- **conversation_id** (*str*) – the ID of the group or recipient
- **message_id** (*str*) – the ID of the message

classmethod **`destroy`**(*conversation_id*, *message_id*)
Unlike a message.

Parameters

- **conversation_id** (*str*) – the ID of the group or recipient
- **message_id** (*str*) – the ID of the message

class `groupy.api.`**`Users`**
Endpoint for the users API.

classmethod **`me`**()
Get the user's information.

Returns the user's information

Return type `dict`

class `groupy.api.`**`Sms`**
Endpoint for the SMS API.

SMS mode can be enabled or disabled.

classmethod **`create`**(*duration=4*, *registration_id=None*)
Enable SMS mode.

Parameters

- **duration** (*int*) – duration of SMS mode in hours (max of 48)
- **registration_id** (*str*) – the push registration_id or token to suppress (if omitted, SMS and push notifications will both be enabled)

classmethod **`delete`**()
Disable SMS mode.

2.4.3 `groupy.objects`

class `groupy.objects.`**`ApiResponse`**(***kwargs*)
Base class for all API responses.

Note: All keyword arguments become properties.

---

class `groupy.objects.`**`Recipient`**(*endpoint*, *mkey*, *idkey*, ***kwargs*)
Base class for Group and Member.

Recipients can post and recieve messages.

Parameters

- **endpoint** (`Endpoint`) – the API endpoint for messages
- **mkey** (*str*) – the `dict` key under which the endpoint returns messages
- **idkey** (*str*) – the `dict` key whose value represents the key for posting and retrieving messages

messages (*before=None*, *since=None*, *after=None*, *limit=None*)
Return a page of messages from the recipient.

Parameters

- **before** (*str*) – a reference message ID
- **since** (*str*) – a reference message ID
- **after** (*str*) – a reference message ID
- **limit** (*int*) – maximum number of messages to include in the page

Returns a page of messages

Return type `MessagePager`

post (*text*, **attachments*)
Post a message to the recipient.

Although the API limits messages to 450 characters, this method will split the text component into as many as necessary and include the attachments in the final message. Note that a list of messages sent is always returned, even if it contains only one element.

Parameters

- **text** (*str*) – the message text
- **attachments** (`list`) – the attachments to include

Returns a list of raw API responses (sorry!)

Return type `list`

class `groupy.objects.`**Group** (***kwargs*)
A GroupMe group.

add (**members*)
Add a member to a group.

Each member can be either an instance of `Member` or a `dict` containing `nickname` and one of `email`, `phone_number`, or `user_id`.

Parameters **members** (`list`) – members to add to the group

Returns the results ID of the add call

Return type str

classmethod list (*former=False*)
List all of your current or former groups.

Parameters **former** (`bool`) – `True` if former groups should be listed, `False` (default) lists current groups

Returns a list of groups

Return type `FilterList`

members ()
Return a list of the members in the group.

Returns the members of the group

Return type FilterList

refresh()
Refresh the group information from the API.

remove(*member*)
Remove a member from the group.

Parameters **member** (Member) – the member to remove

Returns True if successful, False otherwise

Return type bool

class groupy.objects.**Member**(***kwargs*)
A GroupMe member.

identification()
Return the identification of the member.

A member is identified by their nickname and user_id properties. If the member does not yet have a GUID, a new one is created and assigned to them (and is returned alongside the nickname and user_id properties).

Returns the nickname, user_id, and guid of the member

Return type dict

classmethod identify(*member*)
Return or create an identification for a member.

Member identification is required for adding them to groups. If member is a dict, it must contain the following keys:

- nickname
- user_id or email or phone_number

If an identification cannot be created then raise an AttributeError.

Parameters **member** – either a Member or a dict with the required keys

Returns the identification of member

Return type dict

Raises **AttributeError** if an identication cannot be made

class groupy.objects.**Message**(*recipient*, ***kwargs*)
A GroupMe message.

Parameters **recipient** (Recipient) – the reciever of the message

like()
Like the message.

Returns True if successful, False otherwise

Return type bool

likes()
Return a FilterList of the members that like the message.

Returns a list of the members who "liked" this message

Return type FilterList

unlike()
Unlike the message.

Returns `True` if successful, `False` otherwise

Return type bool

class groupy.objects.**Bot**(***kwargs*)
A GroupMe bot.

Each bot belongs to a single group. Messages posted by the bot are always posted to the group to which the bot belongs.

destroy()
Destroy the bot.

Returns `True` if successful, `False` otherwise

Return type bool

classmethod list()
Return a list of your bots.

Returns a list of your bots

Return type `FilterList`

post(*text*, *picture_url=None*)
Post a message to the group of the bot.

Parameters

- **text** (*str*) – the message text
- **picture_url** (*str*) – the GroupMe image URL for an image

Returns `True` if successful, `False` otherwise

Return type bool

class groupy.objects.**User**(***kwargs*)
A GroupMe user.

This is you, as determined by your API key.

disable_sms()
Disable SMS mode.

Disabling SMS mode causes push notifications to resume and SMS text messages to be discontinued.

Returns `True` if successful, `False` otherwise

Return type `bool`

enable_sms(*duration=4*, *registration_token=None*)
Enable SMS mode.

Enabling SMS mode causes GroupMe to send a text message for each message sent to the group.

Parameters

- **duration** (*int*) – the number of hours for which to send text messages
- **registration_token** (*str*) – the push notification token for for which messages should be suppressed; if omitted, the user will recieve both push notifications as well as text messages

Returns `True` if successful, `False` otherwise

Return type bool

classmethod get()
Return your user information.

Returns your user information

Return type dict

class groupy.objects.**Attachment**(*type_*, ***kwargs*)
A GroupMe attachment.

Attachments are polymorphic objects representing either an image, location, split, or emoji. Use one of the factory methods to create an attachment.

classmethod emoji(*placeholder*, *charmap*)
Create an emoji attachment.

Parameters

- **placeholder** (*str*) – a high code-point character
- **charmap** (list) – a two-dimensional charmap

Returns an emoji attachment

Return type Attachment

classmethod image(*url*)
Create an image attachment.

Parameters **url** (*str*) – the GroupMe image URL for an image

Returns image attachment

Return type Attachment

classmethod location(*name*, *lat*, *lng*)
Create a location attachment.

Parameters

- **name** (*str*) – the name of the location
- **lat** (*float*) – the latitude component
- **lng** (*float*) – the longitude component

Returns a location attachment

Return type Attachment

classmethod new_image(*image*)
Create an image attachment for a local image.

Note that this posts the image to the image service API and uses the returned URL to create an image attachment.

Parameters **image** (file) – a file-like object containing an image

Returns image attachment

Return type Attachment

classmethod split(*token*)
Create a split attachment.

Note: The split attachment is depreciated according to GroupMe.

Parameters **token** (*str*) – the split token

Returns a split attachment

Return type Attachment

class groupy.objects.**FilterList**
A filterable list.

Acts just like a regular list, except it can be filtered using a special keyword syntax. Also, the first and last items are special properties.

filter(***kwargs*)
Filter the list and return a new instance.

Arguments are keyword arguments only, and can be appended with operator method names to indicate relationships other than equals. For example, to filter the list down to only items whose name property contains "ie":

```
new_list = filter_list.filter(name__contains='ie')
```

As another example, this filters the list down to only those with a created property that is less than 1234567890:

```
new_list = filter_list.filter(created__lt=1234567890)
```

Acceptable operators are:

- __lt: less than
- __gt: greater than
- __contains: contains
- __eq: equal to
- __ne: not equal to
- __le: less than or equal to
- __ge: greater than or equal to

Use of any operator listed here results in a InvalidOperatorError.

Returns a new list with potentially less items than the original

Return type FilterList

class groupy.objects.**MessagePager**(*group*, *messages*, *backward=False*)
A filterable, extendable page of messages.

Parameters

- **group** (Group) – the group from which to page through messages
- **messages** (list) – the initial page of messages
- **backward** (bool) – True if the oldest message is at index 0, False otherwise

inewer()
Add in-place the next (newer) page of messages.

Returns True if successful, False otherwise

Return type bool

iolder()
Add in-place the previous (older) page of messages.

Returns `True` if successful, `False` otherwise

Return type `bool`

newer()
Return the next (newer) page of messages.

Returns a newer page of messages

Return type `MessagePager`

newest
Return the newest message in the list.

Returns the newest message in the list

Return type `Message`

older()
Return the previous (older) page of messages.

Returns an older page of messages

Return type `MessagePager`

oldest
Return the oldest message in the list.

Returns the oldest message in the list

Return type `Message`

prepend(*messages*)
Prepend a list of messages to the list.

Parameters **messages** (`list`) – the messages to prepend

2.4.4 groupy.errors

exception groupy.errors.**GroupMeError**
A general GroupMe error.

exception groupy.errors.**InvalidOperatorError**
Error thrown when an unsupported `FilterList` filter is used.

exception groupy.errors.**InvalidResponseError**
Error representing an unparsable response from the API.

2.4.5 groupy.status

groupy.status.**BAD_GATEWAY** = 502
GroupMe is down or being upgraded

groupy.status.**BAD_REQUEST** = 400
Invalid format or invalid data is specified in the request

groupy.status.**CREATED** = 201
Resource was created successfully

groupy.status.**ENHANCE_YOUR_CLAIM = 420**
You are being rate limited

groupy.status.**FORBIDDEN = 403**
The request was understood, but it has been refused

groupy.status.**INTERNAL_SERVER_ERROR = 500**
Something unexpected occurred

groupy.status.**NOT_FOUND = 404**
The URI requested is invalid or the requested resource does not exist

groupy.status.**NOT_MODIFIED = 304**
There was no new data to return

groupy.status.**NO_CONTENT = 204**
Resource was deleted successfully

groupy.status.**OK = 200**
Success

groupy.status.**SERVICE_UNAVAILABLE = 503**
The GroupMe servers are up but overloaded with requests

groupy.status.**UNAUTHORIZED = 401**
Authentication credentials were missing or incorrect

groupy.status.**description**(*code*)
Return the text description for a code.

Parameters **code** (*int*) – the HTTP status code

Returns the text description for the status code

Return type str

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*

a

c

e

g

o

M

N

O

P

R

S

U