Groupy Documentation

Release 0.10.0

Robert Grant

Jun 05, 2018

Contents

The simple yet powerful API client for the GroupMe messaging service.

```
$ python3 -m pip install GroupyAPI
```

```
>>> from groupy import Client
>>> client = Client.from_token('api_token')

>>> groups = list(client.groups.list_all())

>>> for group in groups:
...     print(group.name)

>>> group = groups[0]
>>> for member in group.members:
...     print(member.nickname)

>>> for message in group.messages.list_all():
...     print(message.text)
...     for attachment in message.attachments:
...         print(attachment.type)
...     if 'love' in message.text.lower():
...         message.like()
```

CHAPTER 1

Features

Groupy supports the entire GroupMe API... plus one or two undocumented features.

- list current and former groups
- create, update, and destroy groups
- list group and chat messages
- access group leaderboards and galleries
- post new messages to groups and chats
- upload images for use as attachments
- download image attachments
- add, remove, and list group members
- like and unlike messages
- access your user account details
- update your SMS mode settings
- block and unblock other users
- post messages as a bot
- create, list, update, and destroy bots
- list chats with other users
- join new groups and rejoin former groups
- transfer group ownership

CHAPTER 2

Table of Contents

2.1 Installation

To get started, you'll need to get an account at Groupme.com. Next, you'll need to obtain your access token so you can make API requests:

1. Login to the developer portal.
2. Click the "Access Token" button on the top menu bar.
3. Your access token is displayed in bold text. Grab it.

Lastly, install Python >= 3.4. Now you're ready to install Groupy!

```
$ pip install GroupyAPI
```

2.1.1 Troubleshooting

Sometimes things go wrong. Here are some common things to check when encountering problems after installing.

It says no such package when I import groupy... Check whether you copied the `groupy` package into the correct python package directory. It must be a directory on your `sys.path`.

I get an unauthorized error when I try to do anything... Check that your API token is correct just before you use it to create a Groupy client.

2.2 Getting Started

First, make sure you have:

- **Groupy** installed
- your API key

See the *Installation* page for help if needed.

2.2.1 The Client

Creating a client

Assuming your API token is stored in `token`:

```
>>> from groupy.client import Client
>>> client = Client.from_token(token)
```

Clients are capable of listing groups, chats, and bots. It can also provide your user information.

Listing groups

Groups are listed in pages. You can specify which page and how many groups per page using the `page` and `per_page` parameters. `per_page` defaults to 10.

```
>>> client.groups.list()
<groupy.pagers.GroupList at 0x7fcd9f7174e0>
>>> client.groups.list(page=2, per_page=30)
<groupy.pagers.GroupList at 0x7fa02c23db70>
```

The *GroupList* returned can be iterated to obtain the groups in that page.

```
>>> for group in client.groups.list():
...     print(group.name)
```

Since paging can be a pain, the *GroupList* also possesses an *autopage()* method that can be used to obtain all groups by automatically handling paging:

```
>>> groups = client.groups.list()
>>> for group in groups.autopage():
...     print(group.name)
```

However, the easiest way to list all groups, is:

```
>>> for group in client.groups.list_all():
...     print(group.name)
```

Note: The ordering of groups is determined by most recent activity, so the group with the youngest message will be listed first. For this reason, autopaging is highly recommended when the goal is to list all groups.

Omitting fields

Sometimes, particularly when a group contains hundreds of members, the response is "too large" and contains an incomplete response. In that case, an *InvalidJsonError* is raised.

To avoid this, use the `omit` parameter to specify fields to omit.

```
>>> groups = client.groups.list(omit="memberships")
```

Note: Multiple fields must be formatted in a CSV (e.g. "memberships,description"). At the time of this writing, however, the API only supports omission of "memberships."

To refresh a group with fresh data from the server, thus replenishing any missing fields, use `refresh_from_server()`:

```
>>> group.refresh_from_server()
```

Listing chats

Listing chats is exactly list listing groups, except that you cannot choose to omit fields.

```
>>> for chat in client.chats.list_all():
...     print(chat.other_user['name'])
```

Listing bots

Bots are listed all in one go. That is, the list of bots you own is not paginated.

```
>>> for bot in client.bots.list():
...     print(bot.name)
```

Your own user information

At any time, you can easily access information about your GroupMe user account as a simple dictionary:

```
>>> fresh_user_data = client.user.get_me()
```

Since user information does not typically change during the lifetime of a single client instance, the user information is cached the first time it is fetched. You can access the cached user information as a read-only property:

```
>>> cached_user_data = client.user.me
```

2.2.2 Resources

In general, if a field is present in an API response, you can access it as an attribute of the resource. For example:

```
>>> group.name
'My cool group'
>>> member.id
'123456789'
```

Some fields are converted to more useful objects for you:

```
>>> message.created_at
datetime.datetime(2015, 2, 8, 2, 8, 40)
```

Groups

Creating new groups

```
>>> new_group = client.groups.create(name='My group')
```

Listing messages from a group

```
>>> message_page = group.messages.list()
>>> for message in group.messages.list_all():
...     print(message.text)
...
>>> message_page = group.messages.list_after(message_id=message)
```

Note: See "Listing messages" for details.

Accessing members of a group

```
>>> members = group.members
```

Viewing the leaderboard

```
>>> daily_best = group.leaderboard.list_day()
>>> weekly_best = group.leaderboard.list_week()
>>> my_best = group.leaderboard.list_for_me()
```

Viewing the gallery

```
>>> messages = group.gallery.list()
```

Destroying a group

```
>>> if group.destroy():
...     print('Bye bye!')
... else:
...     print('Something went wrong...')
```

Chats

A chat represents a conversation between you and another user.

Listing messages

```
>>> messages = chat.messages.list()
```

Note: See the section on messages below for details.

Members

Blocking/Unblocking a member

```
>>> block = member.block()
>>> member.unblock()
```

Removing members from groups

Note: Remember, members are specific to the group from which they are obtained.

```
>>> member.remove()
```

Messages

Creating a message (in a group)

```
>>> message = group_or_chat.post(text='hi')
```

Liking/Unliking a message

```
>>> message.like()
>>> message.unlike()
```

Listing messages

```
>>> messages = chat_or_group.messages.list()
>>> oldest_message_in_page = messages[-1]
>>> page_two = chat_or_group.messages.list_before(oldest_message_in_page.id)
>>> all_messages = list(chat_or_group.messages.list().autopage())
```

Attachments

Currently, **Groupy** supports the following types of attachments:

- *Location* - for locations

- *Image* - for images
- *Mentions* - for "@" mentions
- *Emoji* - for emoticons
- *Split* - for splitting bills *(deprecated)*

For all other types of attachments (such as those introduced in the future) there exists a generic *Attachment* class.

The following sections cover the various types of attachments and how to create them. Assume we have already imported the attachments module:

```
>>> from groupy import attachments
```

Locations

Location attachments are the simplest of all attachment types. Each includes a `name`, a latitude `lat`, and a longitude `lng`. Some location attachments also contain a `foursqure_venue_id`.

```
>>> location = attachments.Location(name='Camelot', lat=42, lng=11.2)
```

Images

Image attachments are unique in that they do not actually contain the image data. Instead, they specify the URL from which you can obtain the actual image. To create a new image from a local file object,

```
>>> with open('some-image', 'rb') as f:
>>>     image = client.images.from_file(f)
```

To fetch the actual image bytes of an image attachment, use the `client`:

```
>>> image_data = client.images.download(image)
```

Mentions

Mentions are an undocumented type of attachment. However, they are simple to understand. Mentions capture the details necessary to highlight "@" mentions of members in groups. They contain a list of `loci` and an equal-sized list of `user_ids`.

Assuming Bob's user ID is 1234, the mention of Bob in "Hi @Bob!" would be:

```
>>> mention = attachments.Mentions(loci=[(3, 4)],
...                                user_ids=['1234'])
```

Each element in `loci` has two integers, the first of which indicates the starting index of the mentioning text, while second indicates its length. The strings in `user_ids` correspond *by index* to the elements in `loci`. You can use the `loci` to extract the mentioning portion of the text, as well as obtain the mentioned member via `user_ids`.

An example with mutiple mentions probably illustrates this better. If Bill (user ID 2345) and Zoe Childs (user ID 6789) are mentioned in "@Bill hey I saw you with @Zoe Childs at the park!""

```
>>> mentions = attachments.Mentions(loci=[[0, 5], [25, 11]],
...                                 user_ids=['2345', '6789'])
```

Emojis

`Emojis` are also an undocumented type of attachment, yet frequently appear in messages. Emoji attachments have a `placeholder` and a `charmap`. The `placeholder` is a high-point or unicode character designed to mark the location of the emoji in the text of the message. The `charmap` serves as some sort of translation or lookup tool for obtaining the actual emoji.

Splits

Note: This type of attachment is depreciated. They were part of GroupMe's bill splitting feature that seems to no longer be implemented in their clients. **Groupy**, however, still supports them due to their presence in older messages.

2.3 Developer Docs

This section of the documentation is for other developers, and contains the complete information about each package, module, class, and method.

2.3.1 groupy.client

class groupy.client.**Client**(*session*)
The API client.

The client is the main point of interaction. It can directly list groups, chats, bots, and provide your user information. It can also download the image of a message attachment.

Parameters **session** (Session) – the request session

classmethod **from_token**(*token*)
Create a client directly from an API token.

Parameters **token** (*str*) – an API token

Returns a client

Return type *Client*

2.3.2 groupy.api

groupy.api.attachments

class groupy.api.attachments.**Attachment**(*type*, ***data*)
Base attachment class.

Every attachment has a type and additional data.

Parameters

- **type** (*str*) – attachment type
- **data** (*kwargs*) – additional attachment data

classmethod **from_bulk_data**(*attachments*)
Create multiple attachments from a list of attachment data.

Returns attachment sublcass objects

Return type list

classmethod from_data(*type*, ***data*)
Create an attachment from data.

Parameters

- **type** (*str*) – attachment type
- **data** (*kwargs*) – additional attachment data

Returns an attachment subclass object

Return type *~groupy.api.attachments.Attachment*

to_json()
Return the attachment as JSON serializable dict.

Returns serializable attachment data

Return type dict

class groupy.api.attachments.**Emoji**(*placeholder*, *charmap*)
An emoji attachment.

Parameters

- **placeholder** (*str*) – the "stand-in" character for the emoji
- **charmap** (list) – a list of 2-tuples specifying which emojis

class groupy.api.attachments.**Image**(*url*, *source_url=None*)
An image attachment.

Parameters

- **url** (*str*) – the absolute URL for the image
- **source_url** (*str*) – an optional, absolute URL for the image source

class groupy.api.attachments.**Images**(*session*, *path=None*)
A manager for handling image uploads/downloads.

base_url = 'https://image.groupme.com/'
the base url for the pictures API

download(*image*, *url_field='url'*, *suffix=None*)
Download the binary data of an image attachment.

Parameters

- **image** (*Image*) – an image attachment
- **url_field** (*str*) – the field of the image with the right URL
- **suffix** (*str*) – an optional URL suffix

Returns binary image data

Return type bytes

download_avatar(*image*, *url_field='url'*)
Downlaod the binary data of an image attachment at avatar size.

Parameters **url_field** (*str*) – the field of the image with the right URL

Returns binary image data

Return type bytes

download_large(*image*, *url_field='url'*)
Downlaod the binary data of an image attachment at large size.

Parameters **url_field** (*str*) – the field of the image with the right URL

Returns binary image data

Return type bytes

download_preview(*image*, *url_field='url'*)
Downlaod the binary data of an image attachment at preview size.

Parameters **url_field** (*str*) – the field of the image with the right URL

Returns binary image data

Return type bytes

from_file(*fp*)
Create a new image attachment from an image file.

Parameters **fp** (*file*) – a file object containing binary image data

Returns an image attachment

Return type *Image*

upload(*fp*)
Upload image data to the image service.

Call this, rather than *from_file()*, you don't want to create an attachment of the image.

Parameters **fp** (*file*) – a file object containing binary image data

Returns the URLs for the image uploaded

Return type dict

class groupy.api.attachments.**LinkedImage**(*url*, *source_url=None*)

class groupy.api.attachments.**Location**(*lat*, *lng*, *name*, *foursqure_venue_id=None*)
A location attachment.

Parameters

- **lat** (*float*) – latitude
- **lng** (*float*) – longitude
- **name** (*str*) – the name
- **foursqure_venue_id** (*str*) – an optional Foursquare venue ID

class groupy.api.attachments.**Mentions**(*loci*, *user_ids*)
A mentions attachment.

Parameters

- **loci** (list) – the start and end indices of one or more mentions
- **user_ids** (list) – the user_ids of one or more users mentioned

class groupy.api.attachments.**Split**(*token*)
A split attachment (deprecated in the API).

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

groupy.api.base

class groupy.api.base.**ManagedResource**(*manager*, ***data*)
Class to represent an API object.

class groupy.api.base.**Manager**(*session*, *path=None*)
Class for interacting with the endpoint for a resource.

Parameters

- **session** (Session) – the requests session
- **path** (*str*) – path relative to the base URL

base_url = 'https://api.groupme.com/v3/'
the base URL

groupy.api.blocks

class groupy.api.blocks.**Block**(*manager*, ***data*)
A block between you and another user.

exists()
Return True if the block still exists.

Returns True if the block exists

Return type bool

unblock()
Remove the block.

Returns True if successful

Return type bool

class groupy.api.blocks.**Blocks**(*session*, *user_id*)
Blocks manager.

Parameters

- **session** (Session) – the request session
- **user_id** (*str*) – your user ID

between(*other_user_id*)
Check if there is a block between you and the given user.

Returns True if the given user has been blocked

Return type bool

block(*other_user_id*)
Block the given user.

Parameters **other_user_id** (*str*) – the ID of the user to block

Returns the block created

Return type *Block*

list()
List the users you have blocked.

Returns a list of *Block*'s

Return type *list*

unblock(*other_user_id*)
Unblock the given user.

Parameters **other_user_id** (*str*) – the ID of the user to unblock

Returns `True` if successful

Return type bool

groupy.api.bots

class groupy.api.bots.**Bot**(*manager*, ***data*)
A bot.

destroy()
Destroy the bot.

Returns `True` if successful

Return type bool

post(*text*, *attachments=None*)
Post a message as the bot.

Parameters

- **text** (*str*) – the text of the message
- **attachments** (`list`) – a list of attachments

Returns `True` if successful

Return type bool

class groupy.api.bots.**Bots**(*session*)
A bot manager.

create(*name*, *group_id*, *avatar_url=None*, *callback_url=None*, *dm_notification=None*, ***kwargs*)
Create a new bot in a particular group.

Parameters

- **name** (*str*) – bot name
- **group_id** (*str*) – the group_id of a group
- **avatar_url** (*str*) – the URL of an image to use as an avatar
- **callback_url** (*str*) – a POST-back URL for each new message
- **dm_notification** (*bool*) – whether to POST-back for direct messages?

Returns the new bot

Return type *Bot*

destroy(*bot_id*)
Destroy a bot.

Parameters **bot_id** (*str*) – the ID of the bot to destroy

Returns `True` if successful

Return type bool

list()
Return a list of bots.

Returns all of your bots

Return type *list*

post(*bot_id*, *text*, *attachments=None*)
Post a new message as a bot to its room.

Parameters

- **bot_id** (*str*) – the ID of the bot
- **text** (*str*) – the text of the message
- **attachments** (*list*) – a list of attachments

Returns `True` if successful

Return type bool

groupy.api.chats

class groupy.api.chats.**Chat**(*manager*, ***data*)
A chat with another user.

post(*text=None*, *attachments=None*)
Post a message to the chat.

Parameters

- **text** (*str*) – the text of the message
- **attachments** (`list`) – a list of attachments

Returns `True` if successful

Return type bool

class groupy.api.chats.**Chats**(*session*)
A chat manager.

list(*page=1*, *per_page=10*)
List a page of chats.

Parameters

- **page** (*int*) – which page
- **per_page** (*int*) – how many chats per page

Returns chats with other users

Return type *ChatList*

list_all(*per_page=10*)
List all chats.

Parameters **per_page** (*int*) – how many chats per page

Returns chats with other users

Return type *ChatList*

groupy.api.groups

class groupy.api.groups.**ChangeOwnersResult**(*group_id*, *owner_id*, *status*)
The result of requesting a group owner change.

Parameters

- **group_id** (*str*) – group_id of the group
- **owner_id** (*str*) – the ID of the new owner
- **status** (*str*) – the status of the request

is_success
Return `True` if the request was successful.

status_texts = {'200': 'everything checked out', '400': 'the group is already owned
a map of statuses to meanings

success_code = '200'
the status that represents success

class groupy.api.groups.**Group**(*manager*, ***data*)
A group.

change_owners(*user_id*)
Change the owner of the group.

Note that the user must be a member of the group.

Parameters **user_id** (*str*) – the user_id of the new owner

Returns the result

Return type *ChangeOwnersResult*

create_bot(*name*, *avatar_url=None*, *callback_url=None*, *dm_notification=None*, ***kwargs*)
Create a new bot in a particular group.

Parameters

- **name** (*str*) – bot name
- **avatar_url** (*str*) – the URL of an image to use as an avatar
- **callback_url** (*str*) – a POST-back URL for each new message
- **dm_notification** (*bool*) – whether to POST-back for direct messages?

Returns the new bot

Return type *Bot*

destroy()
Destroy the group.

Note that you must be the owner.

Returns `True` if successful

Return type bool

get_membership()
Get your membership.

Note that your membership may not exist. For example, you do not have a membership in a former group. Also, the group returned by the API when rejoining a former group does not contain your membership. You must call *refresh_from_server()* to update the list of members.

Returns your membership in the group

Return type *Member*

Raises ***groupy.exceptions.MissingMembershipError*** – if your membership is not in the group data

leave()
Leave the group.

Returns `True` if successful

Return type bool

post(*text=None*, *attachments=None*)
Post a new message to the group.

Parameters

- **text** (*str*) – the text of the message
- **attachments** (`list`) – a list of attachments

Returns the message

Return type *Message*

refresh_from_server()
Refresh the group from the server in place.

rejoin()
Rejoin the group.

Note that this must be a former group.

Returns a current (not former) group

Return type *Group*

update(*name=None*, *description=None*, *image_url=None*, *office_mode=None*, *share=None*, ***kwargs*)
Update the details of the group.

Parameters

- **name** (*str*) – group name (140 characters maximum)
- **description** (*str*) – short description (255 characters maximum)
- **image_url** (*str*) – GroupMe image service URL
- **office_mode** (*bool*) – (undocumented)
- **share** (*bool*) – whether to generate a share URL

Returns an updated group

Return type *Group*

update_membership(*nickname=None*, ***kwargs*)
Update your own membership.

Note that this fails on former groups.

Parameters **nickname** (*str*) – new nickname

Returns updated membership

Return type `Member`

class `groupy.api.groups.`**Groups**(*session*)
A group manager.

change_owners(*group_id*, *owner_id*)
Change the owner of a group.

Note: you must be the owner to change owners

Parameters

- **group_id** (*str*) – the group_id of a group
- **owner_id** (*str*) – the ID of the new owner

Returns the result

Return type *ChangeOwnersResult*

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

Note that, although possible, there may be issues when not using an image URL from GroupMe's image service.

Parameters

- **name** (*str*) – group name (140 characters maximum)
- **description** (*str*) – short description (255 characters maximum)
- **image_url** (*str*) – GroupMe image service URL
- **share** (*bool*) – whether to generate a share URL

Returns a new group

Return type *Group*

destroy(*id*)
Destroy a group.

Parameters **id** (*str*) – a group ID

Returns `True` if successful

Return type bool

get(*id*)
Get a single group by ID.

Parameters **id** (*str*) – a group ID

Returns a group

Return type *Group*

join(*group_id*, *share_token*)
Join a group using a share token.

Parameters

- **group_id** (*str*) – the group_id of a group
- **share_token** (*str*) – the share token

Returns the group

Return type *Group*

list(*page=1*, *per_page=10*, *omit=None*)
List groups by page.

The API allows certain fields to be excluded from the results so that very large groups can be fetched without exceeding the maximum response size. At the time of this writing, only 'memberships' is supported.

Parameters

- **page** (*int*) – page number
- **per_page** (*int*) – number of groups per page
- **omit** (*int*) – a comma-separated list of fields to exclude

Returns a list of groups

Return type *GroupList*

list_all(*per_page=10*, *omit=None*)
List all groups.

Since the order of groups is determined by recent activity, this is the recommended way to obtain a list of all groups. See *list()* for details about `omit`.

Parameters

- **per_page** (*int*) – number of groups per page
- **omit** (*int*) – a comma-separated list of fields to exclude

Returns a list of groups

Return type *GroupList*

list_former()
List all former groups.

Returns a list of groups

Return type *list*

rejoin(*group_id*)
Rejoin a former group.

Parameters **group_id** (*str*) – the group_id of a group

Returns the group

Return type *Group*

update(*id*, *name=None*, *description=None*, *image_url=None*, *office_mode=None*, *share=None*, ***kwargs*)
Update the details of a group.

Note: There are significant bugs in this endpoint! 1. not providing `name` produces 400: "Topic can't be blank" 2. not providing `office_mode` produces 500: "sql: Scan error on column index 14: sql/driver: couldn't convert <nil> (<nil>) into type bool"

Note that these issues are "handled" automatically when calling update on a *Group* object.

Parameters

- **id** (*str*) – group ID
- **name** (*str*) – group name (140 characters maximum)
- **description** (*str*) – short description (255 characters maximum)
- **image_url** (*str*) – GroupMe image service URL
- **office_mode** (*bool*) – (undocumented)
- **share** (*bool*) – whether to generate a share URL

Returns an updated group

Return type *Group*

groupy.api.memberships

class groupy.api.memberships.**Member**(*manager*, *group_id*, ***data*)

A user's membership in a particular group.

Members have both an ID and a membership ID. The membership ID is unique to the combination of user and group.

It can be helpful to think of a "memmber" as a "membership." That is, a specific user in a specific group. Thus, two `Member` objects are equal only if their `id` fields are equal,. As a consequence, the two `Member` objects representing user A in two groups X and Y will _not_ be equal.

Parameters

- **manager** (*Manager*) – a manager for the group of the membership
- **group_id** (*str*) – the group_id of the membership
- **data** (*kwargs*) – additional membership data

add_to_group(*group_id*, *nickname=None*)

Add the member to another group.

If a nickname is not provided the member's current nickname is used.

Parameters

- **group_id** (*str*) – the group_id of a group
- **nickname** (*str*) – a new nickname

Returns a membership request

Return type *MembershipRequest*

block()

Block the user of the membership.

Returns the block created

Return type *Block*

is_blocked()

Check whether you have the user of the membership blocked.

Returns `True` if the user is blocked

Return type bool

post(*text=None*, *attachments=None*, *source_guid=None*)
Post a direct message to the user.

Parameters

- **text** (*str*) – the message content
- **attachments** – message attachments
- **source_guid** (*str*) – a client-side unique ID for the message

Returns the message sent

Return type *DirectMessage*

remove()
Remove the member from the group (destroy the membership).

Returns `True` if successfully removed

Return type bool

unblock()
Unblock the user of the membership.

Returns `True` if successfully unblocked

Return type bool

class groupy.api.memberships.**MembershipRequest**(*manager*, **requests*, ***data*)
A membership request.

Parameters

- **manager** (*Manager*) – a manager for the group of the membership
- **requests** (*args*) – the members requested to be added
- **data** (*kwargs*) – the membership request response data

class Results(*members*, *failures*)

failures
Alias for field number 1

members
Alias for field number 0

check_if_ready()
Check for and fetch the results if ready.

get()
Return the results now.

Returns the membership request results

Return type `Results`

Raises

- ***groupy.exceptions.ResultsNotReady*** – if the results are not ready
- ***groupy.exceptions.ResultsExpired*** – if the results have expired

get_failed_requests(*results*)
Return the requests that failed.

Parameters **results** (list) – the results of a membership request check

Returns the failed requests

Return type generator

get_new_members(*results*)
Return the newly added members.

Parameters **results** (list) – the results of a membership request check

Returns the successful requests, as Members

Return type generator

is_ready(*check=True*)
Return True if the results are ready.

If you pass check=False, no attempt is made to check again for results.

Parameters **check** (*bool*) – whether to query for the results

Returns True if the results are ready

Return type bool

poll(*timeout=30*, *interval=2*)
Return the results when they become ready.

Parameters

- **timeout** (*int*) – the maximum time to wait for the results
- **interval** (*float*) – the number of seconds between checks

Returns the membership request result

Return type Results

class groupy.api.memberships.**Memberships**(*session*, *group_id*)
A membership manager for a particular group.

Parameters

- **session** (Session) – the request session
- **group_id** (*str*) – the group_id of a group

add(*nickname*, *email=None*, *phone_number=None*, *user_id=None*)
Add a user to the group.

You must provide either the email, phone number, or user_id that uniquely identifies a user.

Parameters

- **nickname** (*str*) – new name for the user in the group
- **email** (*str*) – email address of the user
- **phone_number** (*str*) – phone number of the user
- **user_id** (*str*) – user_id of the user

Returns a membership request

Return type *MembershipRequest*

add_multiple(**users*)
Add multiple users to the group at once.

Each given user must be a dictionary containing a nickname and either an email, phone number, or user_id.

Parameters **users** (*args*) – the users to add

Returns a membership request

Return type *MembershipRequest*

check(*results_id*)
Check for results of a membership request.

Parameters **results_id** (*str*) – the ID of a membership request

Returns successfully created memberships

Return type list

Raises

- ***groupy.exceptions.ResultsNotReady*** – if the results are not ready
- ***groupy.exceptions.ResultsExpired*** – if the results have expired

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

Parameters **membership_id** (*str*) – the ID of a member in this group

Returns True if the member was successfully removed

Return type bool

update(*nickname=None*, ***kwargs*)
Update your own membership.

Note that this fails on former groups.

Parameters **nickname** (*str*) – new nickname

Returns updated membership

Return type *Member*

groupy.api.messages

class groupy.api.messages.**DirectMessage**(*manager*, ***data*)
A direct message between two users.

class groupy.api.messages.**DirectMessages**(*session*, *other_user_id*)
Manager for direct messages with a particular user.

Parameters

- **session** – request session
- **other_user_id** (*str*) – user_id of another user

create(*text=None*, *attachments=None*, *source_guid=None*)
Send a new direct message to the user.

Only provide the source_guid if you want to control it.

Parameters

- **text** (*str*) – the message content
- **attachments** – message attachments
- **source_guid** (*str*) – a client-side unique ID for the message

Returns the message sent

Return type *DirectMessage*

list(*before_id=None*, *since_id=None*, ***kwargs*)
Return a page of direct messages.

The messages come in reversed order (newest first). Note you can only provide _one_ of `before_id`, `since_id`.

Parameters

- **before_id** (*str*) – message ID for paging backwards
- **since_id** (*str*) – message ID for most recent messages since

Returns direct messages

Return type *MessageList*

list_all(*before_id=None*, *since_id=None*, ***kwargs*)
Return all direct messages.

The messages come in reversed order (newest first). Note you can only provide _one_ of `before_id`, `since_id`.

Parameters

- **before_id** (*str*) – message ID for paging backwards
- **since_id** (*str*) – message ID for most recent messages since

Returns direct messages

Return type *MessageList*

list_all_before(*message_id*, ***kwargs*)
Return all direct messages created before a message.

This can be used to page backwards through messages.

Parameters **message_id** (*str*) – the ID of a message

Returns direct messages

Return type *MessageList*

list_before(*message_id*, ***kwargs*)
Return a page of direct messages created before a message.

This can be used to page backwards through messages.

Parameters **message_id** (*str*) – the ID of a message

Returns direct messages

Return type *MessageList*

list_since(*message_id*, ***kwargs*)
Return a page of direct messages created since a message.

This is used to fetch the most recent messages after another. There may exist messages between the one given and the ones returned.

Parameters **message_id** (*str*) – the ID of a message

Returns direct messages

Return type *MessageList*

class groupy.api.messages.**Gallery**(*session*, *group_id*)
Manager for messages in the gallery.

This endpoint is undocumented!

Parameters

- **session** – request session
- **group_id** (*str*) – group_id for a particular group

class groupy.api.messages.**GenericMessage**(*manager*, *conversation_id*, ***data*)
A message.

Parameters

- **manager** – a message manager
- **conversation_id** (*str*) – the ID for the conversation
- **data** (*kwargs*) – the data of the message

like()
Like the message.

preview_length = 42
number of characters seen in the repr output

unlike()
Unlike the message.

class groupy.api.messages.**Leaderboard**(*session*, *group_id*)
Manager for messages on the leaderboard.

list(*period*)
List most liked messages for a given period.

Parameters **period** (*str*) – either “day”, “week”, or “month”

Returns the messages

Return type *list*

list_day()
List most liked messages for the last day.

Returns the messages

Return type *list*

list_for_me()
List your top liked messages.

Returns the messages

Return type *list*

list_mine()
List messages you liked.

Returns the messages

Return type *list*

list_month()
List most liked messages for the last month.

Returns the messages

Return type *list*

list_week()
List most liked messages for the last week.

Returns the messages

Return type *list*

class groupy.api.messages.**Likes**(*session*, *conversation_id*, *message_id*)
Manager for likes/unlikes of a particular message.

The message can be from either a group or a chat.

Parameters

- **session** – request session
- **conversation_id** (*str*) – unique ID for a the conversation from which the message originates
- **message_id** (*str*) – unique ID of the message to like/unlike

like()
Like the message.

unlike()
Unlike the message.

class groupy.api.messages.**Message**(*manager*, ***data*)
A group message.

class groupy.api.messages.**Messages**(*session*, *group_id*)
A message manager for a particular group.

Parameters

- **session** (Session) – the request session
- **group_id** (*str*) – the group_id of a group

create(*text=None*, *attachments=None*, *source_guid=None*)
Create a new message in the group.

Parameters

- **text** (*str*) – the text of the message
- **attachments** (*list*) – a list of attachments
- **source_guid** (*str*) – a unique identifier for the message

Returns the created message

Return type *Message*

list(*before_id=None*, *since_id=None*, *after_id=None*, *limit=20*)
Return a page of group messages.

The messages come in reversed order (newest first). Note you can only provide _one_ of before_id, since_id, or after_id.

Parameters

- **before_id** (*str*) – message ID for paging backwards
- **after_id** (*str*) – message ID for paging forwards
- **since_id** (*str*) – message ID for most recent messages since
- **limit** (*int*) – maximum number of messages per page

Returns group messages

Return type *MessageList*

list_after(*message_id*, *limit=None*)

Return a page of group messages created after a message.

This is used to page forwards through messages.

Parameters

- **message_id** (*str*) – the ID of a message
- **limit** (*int*) – maximum number of messages per page

Returns group messages

Return type *MessageList*

list_all(*limit=None*)

Return all group messages.

Parameters **limit** (*int*) – maximum number of messages per page

Returns group messages

Return type *MessageList*

list_all_after(*message_id*, *limit=None*)

Return all group messages created after a message.

Parameters

- **message_id** (*str*) – the ID of a message
- **limit** (*int*) – maximum number of messages per page

Returns group messages

Return type *MessageList*

list_all_before(*message_id*, *limit=None*)

Return all group messages created before a message.

Parameters

- **message_id** (*str*) – the ID of a message
- **limit** (*int*) – maximum number of messages per page

Returns group messages

Return type *MessageList*

list_before(*message_id*, *limit=None*)

Return a page of group messages created before a message.

This can be used to page backwards through messages.

Parameters

- **message_id** (*str*) – the ID of a message
- **limit** (*int*) – maximum number of messages per page

Returns group messages

Return type *MessageList*

list_since(*message_id*, *limit=None*)
Return a page of group messages created since a message.

This is used to fetch the most recent messages after another. There may exist messages between the one given and the ones returned. Use *list_after()* to retrieve newer messages without skipping any.

Parameters

- **message_id** (*str*) – the ID of a message
- **limit** (*int*) – maximum number of messages per page

Returns group messages

Return type *MessageList*

groupy.api.user

class groupy.api.user.**SmsMode**(*session*)

class groupy.api.user.**User**(*session*)

2.3.3 groupy.pagers

class groupy.pagers.**ChatList**(*manager*, *endpoint*, ***params*)

class groupy.pagers.**GalleryList**(*manager*, *endpoint*, ***params*)
Pager for gallery messages.

get_next_page_param(*item*)
Return the param from the given item.

Parameters **item** – the item that has the next page param

Returns next page param value

class groupy.pagers.**GroupList**(*manager*, *endpoint*, ***params*)
Pager for groups.

default_params = {'page': 1}
default to the first page

set_next_page_params()
Set the params in preparation for fetching the next page.

class groupy.pagers.**MessageList**(*manager*, *endpoint*, ***params*)
Pager for messages.

default_mode = 'before_id'
the default mode param

classmethod detect_mode(***params*)
Detect which listing mode of the given params.

Params kwargs params the params

Returns one of the available modes

Return type str

Raises **ValueError** – if multiple modes are detected

fetch_next()
Fetch the next page of results.

Returns the next page of results

Return type list

get_last_item_index()
Return the index of the last item in the page.

get_next_page_param(*item*)
Return the param from the given item.

Parameters **item** – the item that has the next page param

Returns next page param value

modes = {'after_id': -1, 'before_id': -1, 'since_id': 0}
all possible mode params and the index for their next page params

set_next_page_params()
Set the params so that the next page is fetched.

class groupy.pagers.**Pager**(*manager*, *endpoint*, ***params*)
Class for iterating over multiple pages of results.

This is a generic, base class. To create a specific type of pager, provide a definition for set_next_page_params in a subclass.

Parameters

- **manager** (*Manager*) – the manager from which to get results
- **endpoint** (*func*) – a callable from which results can be fetched
- **params** (*kwargs*) – initial params to pass to the manager

autopage()
Iterate through results from all pages.

Returns all results

Return type generator

default_params = {}
the base set of params

fetch()
Fetch the current page of results.

Returns the current page of results

Return type list

fetch_next()
Fetch the next page of results.

Returns the next page of results

Return type list

set_next_page_params()
Set the params in preparation for fetching the next page.

2.3.4 groupy.exceptions

exception groupy.exceptions.**ApiError**(*message=None*)
Base exception for all GroupMe API errors.

exception groupy.exceptions.**BadResponse**(*response*, *message=None*)
Exception raised when the status code of the response was 400 or more.

Parameters

- **response** (Response) – the response
- **message** (*str*) – a description of the exception

exception groupy.exceptions.**FindError**(*message*, *objects*, *tests*, *matches=None*)
Exception raised when the number of results is not 1.

exception groupy.exceptions.**GroupyError**(*message=None*)
Base exception for all Groupy exceptions.

Parameters **message** (*str*) – a description of the exception

message = None
a description of the exception

exception groupy.exceptions.**InvalidJsonError**(*response*, *message='The JSON was incomplete/invalid'*)
Exception raised for incomplete/invalid JSON in a response.

exception groupy.exceptions.**MissingMembershipError**(*group_id*, *user_id*, *message=None*)
Exception raied when your membership could not be found in a group.

exception groupy.exceptions.**MissingMetaError**(*response*, *message='The response contained no meta data'*)
Exception raised for a response that lacks meta data.

exception groupy.exceptions.**MissingResponseError**(*response*, *message='The response contained no response data'*)
Exception raised for a response that lacks response data.

exception groupy.exceptions.**MultipleMatchesError**(*objects*, *tests*, *matches*)
Exception raised when the number of results exceeds 1.

exception groupy.exceptions.**NoMatchesError**(*objects*, *tests*)
Exception raised when the number of results is 0.

exception groupy.exceptions.**NoResponse**(*request*, **args*, ***kwargs*)
Exception raised when the API server could not be reached.

Parameters

- **request** (PreparedRequest) – the original request that was made
- **message** (*str*) – a description of the exception

exception groupy.exceptions.**ResultsError**(*response*, *message*)
Exception raised for asynchronous results.

Parameters

- **response** (Response) – the response
- **message** (*str*) – a description of the exception

exception groupy.exceptions.**ResultsExpired**(*response*, *message='The results have expired'*)

Exception raised when the results have expired.

Parameters

- **response** (Response) – the response
- **message** (*str*) – a description of the exception

exception groupy.exceptions.**ResultsNotReady**(*response*, *message='The results are not ready yet'*)

Exception raised when results are not yet ready.

Parameters

- **response** (Response) – the response
- **message** (*str*) – a description of the exception

2.4 Contributing

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

You can contribute in many ways:

2.4.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/rhgrant10/Groupy/issues.

If you are reporting a bug, please include:

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

Fix Bugs

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

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

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

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/rhgrant10/Groupy/issues.

If you are proposing a feature:

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

2.4.2 Get Started!

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

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

```
$ git clone git@github.com:YOUR_NAME_HERE/Groupy.git
```

3. Install your local copy into a virtualenv. Since 3.3, Python ships with its own virutal environment creator: *venv*. Usage is simple:

```
$ cd Groupy/
$ pyvenv env
$ source env/bin/activate
$ pip install -r requirements.txt -r testing_requirements.txt -e .
```

4. Create a branch *from the dev branch* for local development:

```
$ git checkout -b new-hotness dev
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8, have great coverage, and pass all tests on all supported versions of python. Sounds tough, but *tox* makes this easy:

```
$ tox
```

Note that if you update `requirements.txt` or `testing_requirements.txt` you must tell tox to recreate its virtual environments using the `-r` flag:

```
$ tox -r
```

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

```
$ git add .
$ git commit
$ git push origin new-hotness
```

Please make sure to:

- *not* to commit sensitive data or extra files. You can use `git add -p` to add parts of files if necessary.
- follow proper git commit message standards. In particular, the first line should be under 60 characters long, and any detail should begin on the 3rd line (the second line must be blank).

7. Submit a pull request through the GitHub website.

2.4.3 Pull Request Guidelines

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

1. The pull request should include **tests**.
2. If the pull request adds functionality, the **docs** should be updated. Put your new functionality into a function with a **docstring**, and add the feature to the list in the documentation.
3. The pull request should work for Python 3.4, 3.5, and 3.6. Check https://travis-ci.org/rhgrant10/Groupy/pull_requests and make sure that the tests pass for all supported Python versions.

2.5 Change Log

2.5.1 v0.10.0 (June 4, 2018)

- fix image attachment creation
- support pickling/unpickling of `Resource` objects

2.5.2 v0.9.2 (December 23, 2017)

- fix travis CD

2.5.3 v0.9.0 (December 23, 2017)

- add `Block` equality by `user_id` and `blocked_user_id`
- add `Bot` equality by `bot_id`
- add `Message` equality by `id`
- add `Member` equality by `id`
- add `Group` equality by `group_id`

2.5.4 v0.8.1 (December 19, 2017)

- fix bug when converting stange/invalid timestamps to datetimes
- add missing `created_at` and `updated_at` in `Chat` resources

2.5.5 v0.8.0 (December 15, 2017)

This version represents a significant, backwards-incompatible change! The old API was brittle, poorly designed, and very difficult to test.

- No more global api key; create clients using a specific api key instead
- **Added missing group functionality:**
  - joining with a share token
  - rejoining a former group
  - changing group ownership
- **Added additional endpoints:**
  - leaderboard
  - blocks
- **Reverse engineered the undocumented gallery endpoint**
  - has all known supported listing methods
  - supports before, after, and since with UTC `datetime` s
- Added paging control for endpoints that support it
- **Message paging now maintains the order in which they come from the API:**
  - "before" messages go back in time
  - "after" messages go forward in time
- much more grandular exceptions and no layer violations
- removal of Pillow depenedency for images; simply upload/download image bytes and do with them as you please
- **result filtering has been moved and improved**
  - can now make and reuse a filter
  - can create arbitrary sets of tests and use them in a filter
- now uses the Apache license instead of GPLv3

2.5.6 v0.7.1 (March 29, 2017)

- Fixed an issue serializing attachments posted by the user (not a bot) (thanks to a-vilmin for reporting the issue)

2.5.7 v0.7.0 (November 19, 2016)

- Fixed an issue with attachments not being serializable. Now an attempt to call their `as_dict` method is made (thank you to awctomlinson for pointing it out)
- Fixed problem with `is_liked_by_me`, `is_from_me` and `metions_me` when used on `DirectMessages` (thank you to mmigrate)
- Added attachment support to `Bot`'s `post` method (thank you again to mmigrate)
- Fixed a mispelling in the `mentions_me` method name (thank you adelq)

2.5.8 v0.6.6 (April 23, 2016)

- Fixed a typo in the docs regarding the type of the `group` parameter of the `Bot` class (kudos to JCDJulian)
- Fixed the `Group.update` method signature to include the `group_id` (kudos to mmirate)
- Fixed `Member.identification` such that it uses `Member.guid` rather than `Member._guid` (kudos to mmirate)
- Fixed the uncaught exception chain that occurred when a 304 was returned in `Recipient.messages` (thanks to dvmorris and sbonds for pointing it out)
- Updated the list of contributors

2.5.9 v0.6.5 (January 17, 2016)

- Fixed typo the `Bot` class that caused the bots to have a "gorup_id" (kudos to JCDJulian)
- All modules except `object/listers.py` and `object/responses.py` now have full test coverage
- Updated AUTHORS.rst with all contributors to date (feel free to PR with an email address added to your username)
- Fixed leftover markdown formatting in the CHANGELOG.rst file

2.5.10 v0.6.4 (December 31, 2015)

- Fixed bugs with creating bots (kudos to qlyoung)
- Fixed bugs with posting messages as bots (kudos again to qlyoung)
- Fixed typo bugs in `Group` class (kudos to t3zla)
- Fixed missing Python 3 trove classifier
- Added documentation for contributions
- Updated documentation for setup and installation
- Added a couple more unit tests
- Reconfigured tox test results to not clobber results from other environments

2.5.11 v0.6.3 (December 23, 2015)

- Added support for `tox` (envs py34,py35)
- Added support for `bumpversion`
- Added `make` file for handy development
- Moved to `nosetests` and `coverage`
- Split requirements into regular and testing
- Updated some of the installation/troubleshooting docs
- Merged in open pull-requests for various oversights (kudos to ScufyfNrdHrdr, rAntonioH, and JacobAMason)

2.5.12 v0.6.2 (May 3, 2015)

- Fixed problem when posting messages as a bot
- Added `refresh` option for automatically updating group information after addition/removal of members
- Updated documentation

2.5.13 v0.6.1 (April 25, 2015)

- Fixed code in `responses.py` that was still using the old exception class name
- Changed the `Member.remove()` method to correctly use the `id` of the member rather than the `user_id`
- Slight beefing up of some documentation

2.5.14 v0.5.8 (December 9, 2014)

- Fixed problems with `requirements.txt` and `setup.py` that caused problems installing from `pip`
- Re-wrote many of the unittests (still in progress)
- Added Travis-CI and PyPI badges to the readme
- Bumped requirement for dropbox's `responses` to 0.3.0
- Now uses `setup` from `setuptools` rather than `distutils.core`

2.5.15 v0.5.3 (September 19, 2014)

- Fix packaging bug that caused inner packages to not be installed via `pip3`

2.5.16 v0.5.2 (September 14, 2014)

- Now installable via `pip3`:

```
$ pip3 install GroupyAPI
```

2.5.17 v0.5.1 (August 25, 2014)

Groups

- Added a class method for creating a new group
- Added an instance method for destroying a group

Members

- Fixed member identification on dictionaries

User

- Fixed the enable/disable SMS methods (now class methods as they should be)

Documentation

- Added some module docstrings

- Added API docs for all attachment classes
- Added docs for split attachments
- Moved FilterList docs into the Advanced Usage section
- Rewrote API docs for enabling SMS mode
- Fixed bad sphinx references
- Fixed typos
- Added miscellaneous sections to the README
- Updated feature list

2.5.18 v0.5.0 (August 20, 2014)

- Added support for downloaded the image of an image attachment
- Reorganized modules and project structure
- Updated documentation

2.5.19 v0.4.0 (August 18, 2014)

- Added ability to list all known members
- Re-wrote attachments classes

2.5.20 v0.3.1 (August 14, 2014)

- Fixed bug when adding members to a group
- Many additions to the documentation

2.5.21 v0.3.0 (August 12, 2014)

- Added post and messages methods to members
- Added after_id parameter for direct messages
- Fixed liking and unliking direct messages
- Fixed listing former groups
- Fixed group lists being limited to a max of 500 items
- Documentation now available on Read the Docs!

2.5.22 v0.2.0 (August 11, 2014)

- Added MessagePager class for returning lists of messages

2.5.23 v0.1.3 (August 10, 2014)

- Added attachment class
- Added basic documentation
- Fixed the automatic splitting of long texts
- Fixed invalid response error issue

2.5.24 v0.1.0 (August 9, 2014)

- Initial release

CHAPTER 3

About GroupMe

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.

Python Module Index

g

Index

N

P

R

S

T

U