Groupy Documentation

Release 0.5.0

Robert Grant

August 20, 2014

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
- Add members to and remove members from 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

1.2 Table of Contents

1.2.1 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!

Using `pip`

Note: Installation via `pip` coming soon!

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?

1.2.2 Basic Usage

This page gives an overview of all but the most advanced features of **Groupy**.

First, you'll want to make sure that

- **Groupy** is *installed*

- **Groupy** can *find your API key*

See the *Installation* page for instructions. Now that that's out of the way, let's get started!

Listing Things

The most basic operation is listing things. `Group`s, `Member`s, and `Bot`s can be listed directly.

```
>>> import groupy
>>> groups = groupy.Group.list()
>>> members = groupy.Member.list()
>>> bots = groupy.Bot.list()
```

The object lists are returned as a `FilterList`. These behave just like the built-in `list` does with some convenient functionality: `first` and `last`.

```
>>> groups.first == groups[0]
True
>>> groups.last == groups[-1]
True
```

The most useful feature of a `FilterList`, however, is its `filter()` method. It parses whatever keyword arguments are passed to it and filters the list such that only the items meeting all criteria are included. The keywords correspond to object properties, but also indicate how to test the relation to the value of the keyword argument. Thus a keyword-value pair such as `name='Bob'` would keep only those items with a `name` property equal to `"Bob"`, whereas a pair like `age__lt=20` keeps only those items with an `age` property *less than* `20`.

Some simple examples:

```
>>> from groupy import Group
>>> groups = Group.list()
>>> for g in groups:
...     print(g.name)
...
My Family
DevTeam #6
Friday Night Trivia
>>> for g in groups.filter(name__contains='am'):
...     print(g.name)
My Family
DevTeam #6
>>>
>>> members = groups.first.members()
>>> for m in members:
...     print(m.nickname)
...
Dan the Man
Manuel
Fred
Dan
>>> for m in members.filter(nickname='Dan'):
...     print(m.nickname)
...
Dan
>>> for m in members.filter(nickname__contains='Dan'):
...     print(m.nickname)
...
Dan the Man
Dan
```

```
>>> for m in members.filter(nickname__ge='F'):
...     print(m.nickname)
...
Manuel
Fred
```

Groups

From a Group, you can list its Members and Messages.

```
>>> from groupy import Group
>>> groups = Group.list()
>>> group = groups.first
>>> messages = group.messages()
>>> members = group.memers()
```

A group returns all of its members in a single list. So determining the number of members in a group should be a familiar task.

```
>>> len(members)
5
```

Messages, however, are a different matter. Since there may be thousands of messages in a group, messages are returned in pages. The default (and maximum) number of messages per page is 100. To determine the total number of messages in a group, simply access the `message_count` attribute. Additional pages of messages can be obtained using `older()` and `newer()`.

```
>>> len(messages)
100
>>> group.message_count
3014
>>> older = messages.older()
>>> newer = messages.newer()
```

There are also methods for collecting a newer or older page of messages into one list: `iolder()` and `inewer()`. An example of using the former to retrieve all messages in a group:

```
>>> from groupy import Group
>>> group = Group.list().first
>>> messages = group.messages()
>>> while messages.iolder():
...       pass
>>> len(messages) == group.message_count
True
```

Often you'll want to post a new message to a group. New messages can be posted to a group using its `post()` method.

```
>>> from group import Group
>>> group = Group.list().first
>>> group.post('Hello to you')
>>> print(group.messages().newest.text)
'Hello to you'
```

Note: Posting a message does not affect `message_count`. However, retrieving any page of messages *does* update it.

Groups have many attributes, some of which can be changed.

```
>>> group.name
'My Family'
>>> group.image_url
'http://i.groupme.com/a01b23c45d56e78f90a01b12c3456789'
>>> group.description
'Group of my family members - so we can keep up with each other.'
>>> group.update(name="My Group of Family Members")
>>> group.name
'My Group of Family Members'
>>> group.update(name="[old] Family Group", description="The old family group")
>>> group.name
'[old] Family Group'
>>> group.description
'The old family group'
```

Some Groups also have a `share_url` that others can visit to join the group.

```
>>> group.share_url
'https://groupme.com/join_group/1234567890/SHARE_TOKEN'
```

Beware that not every group is created with a share link, in which case the value of `share_url` would be `None`. However, this can be changed in the same way as other group information.

```
>>> print(group.share_url)
None
>>> group.update(share=True)
>>> group.share_url
'https://groupme.com/join_group/1234567890/SHARE_TOKEN'
```

Note: The `SHARE_TOKEN` is specific to each group's share link.

The remainder of a Groups aattributes cannot be changed. Some more important ones are shown below.

```
>>> group.group_id
'1234567890'
>>> group.creator_user_id
'0123456789'
>>> print(group.created_at)
2013-12-25 9:53:33
>>> print(group.updated_at)
2013-12-26 4:21:08
```

Messages

Unlike Groups, Members, and Bots, Messages *cannot* be listed directly. Instead, Messages are listed either from Group or Member instances.

To list the messages from a group, use a group's `messages()` method.

```
>>> from groupy import Group
>>> group = Group.list().first
>>> messages = group.messages()
```

To list the messages from a member, use a member's `messages()` method.

```
>>> from groupy import Member
>>> member = Member.list().first
>>> messages = member.messages()
```

Messages have several properties. Let's look at a few of them. Messages have a timestamp indicating when the message was created.

```
>>> message = messages.newest
>>> message.created_at
2014-4-29 12:19:05
```

As with other API objects, timestamp data is returned as `datetime.datetime` instances.

Messages also contain information about the member who posted it.

```
>>> message.user_id
'0123456789'
>>> message.name
'Kevin'
>>> message.avatar_url
'http://i.groupme.com/a01b23c45d56e78f90a01b12c3456789'
```

Of course, messages have text and attachments. A message may or may not have text or attachments, but every message must have one or the other.

```
>>> message.text
'Hello'
>>> message.attachments
[Image(url='http://i.groupme.com/a01b23c45d56e78f90a01b12c3456789')]
```

Note: Although the majority of messages will have just one attachment, there is no limit on the number of attachments. In fact, despite most clients being incapable of displaying them, the API doesn't even limit the number of each kind of attachment. For example, a single message might have two images, three locations, and one emoji.

There are multiple types of messages. System messages are messages that are not sent by a member, but generated by member actions. Many things generate system messages, including member changes, group updates (name, avatar, etc.), member changes (nickname, avatar, etc.), and changing the topic.

Additionally there are group messages and direct messages. Group messages are messages in a group, whereas direct messages are messages between two members.

Each message has a few properties that can be used to differentiate the types.

```
>>> message.group_id
'1234567890'
>>> message.recipient_id
None
>>> message.system
False
```

In the above example, we can see that `message.system` is `False`, which indicates that the message was sent by a member, not the system. We can also see that although the message has a `message.group_id`, it does *not* have a `message.recipient_id`, which means it is a group message. Had it been a system message, `message.system` would have been `True`. Had it been a direct message, `message.group_id` would have been `None` and `message.recipient_id` would contain a valid user ID.

Lastly, each message contains a list of user IDs to indicate which members have "liked" it.

```
>>> message.favorited_by
['2345678901', '3456789012']
```

Because often more information about the member is desired, a list of actual `Member` instances can be retrieved using the `likes()` method.

```
>>> message.likes()
[Rob, Jennifer, Vlad]
```

Messages can also be liked and unliked.

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

Note: Currently, the message instance itself does **not** update its own attributes. You must re-fetch the message.

Members

`Member` instances represent other GroupMe users. Finding members can be accomplished in one of three ways.

Firstly, members may be listed from a group. This lists just the members of a particular group.

```
>>> from groupy import Group
>>> group = Group.list().first
>>> members = group.members()
```

Secondly, members may be listed from a message. This lists just the members who have "liked" a particular message.

```
>>> messages = group.messages()
>>> message = message.newest
>>> members = message.likes()
```

Lastly, *all* the members you've seen thus far can be listed directly.

```
>>> from groupy import Member
>>> members = Member.list()
```

Note: Although many attributes of a member are specific to a particular group, members listed in this fashion are taken from a single group with one exception: the nickname of each member listed from `list()` is the most frequent of the names that the member uses among the groups of which you are both members.

Each member has a user ID, a nickname, and a URL indicating their avatar image that are specific to the group from which the member was listed.

```
>>> member = members.first
>>> member.user_id
'0123456789'
>>> member.nickname
'Bill'
>>> member.avatar_url
'http://i.groupme.com/a01b23c45d56e78f90a01b12c3456789'
```

Members have one more property of interest: `muted`. This indicates whether the member has that group muted.

```
>>> member1, member2 = members[:2]
>>> member1.muted
False
>>> member2.muted
True
```

Messaging a member and retrieving the messages between you and the member is done in the same way as when messaging a group.

```
>>> member.post("Hello")
>>> member.messages().newest.text
'Hello'
```

Groups and Members

Members can be added and removed from groups. Adding one or multiple members to a group is quite intuitive. The following examples assume that no one from `group1` is a member of `group2` (although the API doesn't care if you add a member who is already a member).

```
>>> from groupy import Group
>>> group1, group2 = Group.list()[:2]
>>> member = group1.members().first
>>> group2.add(member)
```

Multiple members can be added simultaneously as well. Suppose you wanted to add everyone from `group1` to `group2`.

```
>>> group2.add(*group1.members())
```

Removing members, however, must be done one at a time:

```
>>> for m in group2.members():
...   group2.remove(m)
...
```

GroupMe and You

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_info = User.get()
```

It contains your GroupMe profile/account information and settings:

```
>>> print(your_info.user_id)
12345678
>>> print(your_info.name)
Billy Bob <-- the MAN!
>>> print(your_info.image_url)
http://i.groupme.com/a01b23c45d56e78f90a01b12c3456789
>>> print(your_info.sms)
False
>>> print(your_info.phone_number)
+1 5055555555
>>> print(your_info.email)
bb@example.com
```

It also contains some meta information:

```
>>> print(your_info.created_at)
2011-3-14 14:11:12
>>> print(your_info.updated_at)
2013-4-20 6:58:26
```

`created_at` and `updated_at` are returned as `datetime` objects.

Bots

Bots can be a useful tool because each has a callback URL to which every message in the group is POSTed. This allows your bot the chance to do... well, something (whatever that may be) in response to every message!

Note: Keep in mind that bots can only post messages to groups, so if anything else is going to get done, it'll be done by you, not your bot. That means adding and removing users, liking messages, direct messaging a member, and creating or modifying group will be done under your name.

Bot creation is simple. You'll need to give the bot a name and associate it with a specific group.

```
>>> from groupy import Bot, Group
>>> group = Group.list().first
>>> bot = Bot.create('R2D2', group)
```

`bot` is now the newly created bot and is ready to be used. If you want, you can also specify a callback URL *(recommened)*, as well as an image URL to be used for the bot's avatar.

Just about the only thing a bot can do is post a message to a group. **Groupy** makes it easy:

```
>>> from group import Bot
>>> bot = Bot.list().first
>>> bot.post("I'm a bot!")
```

Note that the bot always posts its messages to the group in which it belongs. You can create multiple bots. Listing all of your bots is straightforward.

```
>>> from groupy import Bot
>>> bots = Bot.list()
```

Now `bots` contains a list of all of your bots.

1.2.3 Advanced Usage

This part of the documentation contains explanations and examples of more complex aspects of **Groupy**.

Attachments

`Messages` can contain various types of `Attachments`. Currently, **Groupy** supports the following types of attachments:

- `Image` - for images
- `Location` - for locations
- `Split` - [1]

[1] This type of attachment will be depreciated soon.

- `Emoji` - for emoticons
- `Mentions` - for "@" mentions

Each of these classes has a `create()` method that accepts arguments specific to it's class.

Types

Locations Location attachments are the simplest of all attachment types. Each includes a name, a latitude, and a longitude.

```
>>> loc = groupy.Location.create('My house', lat=34, lng=-84.3)
```

Some location attachments also contain a `foursqure_venue_id`.

Images Image attachments are unique in that they do not actually contain the image data.

Emojis Emojis are relatively undocumented but frequently appear in messages.

Mentions Mentions are a new type of attachment and is yet undocumented.

Splits This type of attachment is not only largely undocumented, it is depreciated.

1.2.4 Developer Docs

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

The `config` Module

The `config` module contains all the configuration options.

`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

The `endpoint` Module

class `groupy.api.endpoint.`**`Bots`**
Endpoint for the bots API.

Bots can be listed, created, updated, and destroyed. Bots can also post messages to groups.

classmethod **`create`**(*name*, *group_id*, *avatar_url=None*, *callback_url=None*)
Create a new bot.

Parameters

- **name** (*str*) – the name of the bot
- **group_id** (*str*) – the ID of the group to which the bot will belong
- **avatar_url** (*str*) – the GroupMe image URL for the bot's avatar
- **callback_url** (*str*) – the callback URL for the bot

Returns the new bot

Return type `dict`

classmethod destroy(*bot_id*)
Destroy a bot.

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

classmethod index()
List bots.

Returns a list of bots

Return type `list`

classmethod post(*bot_id*, *text*, *picture_url=None*)
Post a message to a group as a bot.

Parameters

- **bot_id** (*str*) – the ID of the bot
- **text** (*str*) – the message text
- **picture_url** (*str*) – the GroupMe image URL for a picture

Returns the created message

Return type `dict`

class groupy.api.endpoint.**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.endpoint.`**`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.endpoint.`**`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.endpoint.`**`Images`**
Endpoint for the image service API.

GroupMe images are created through an upload service that returns a URL at which it can be accessed.

classmethod **`create`**(*image*)
Submit a new image.

Parameters **image** (`file`) – object with a file-like interface and containing an image

Returns the URL at which the image can be accessed

Return type `dict`

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

This method basically returns the inner "payload."

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

Returns API response data

Return type json

class `groupy.api.endpoint.`**`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.endpoint.`**`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.endpoint.`**`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.endpoint.`**`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.

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

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

Returns the user's information

Return type `dict`

The `errors` Module

The `error` module contains all of the exceptions thrown by the GroupMe API.

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

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

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

The `status` Module

The `status` module contains API response status code constants and a method that returns the textual description of such a constant.

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

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

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

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

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

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

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

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

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

`groupy.api.status.`**`INTERNAL_SERVER_ERROR`** = 500
: Something unexpected occurred

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

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

`groupy.api.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`

The `responses` Module

class `groupy.object.responses.`**`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.object.responses.`**`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

`update``(`*name=None*, *description=None*, *image_url=None*, *share=None*`)`

Change group information.

Parameters

- **name** (*str*) – the new name of the group
- **description** (*str*) – the new description of the group
- **image_url** (*str*) – the URL for the new group image
- **share** (`bool`) – whether to generate a share URL

class `groupy.object.responses.`**`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

classmethod list()
List all known members regardless of group membership.

Returns a list of all known members

Return type `FilterList`

class `groupy.object.responses.`**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.object.responses.`**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.

classmethod `create`(*name*, *group*, *avatar_url=None*, *callback_url=None*)

Create a new bot.

Parameters

- **name** (*str*) – the name of the bot
- **group** (`Bot`) – the group to which the bot will belong
- **avatar_url** (*str*) – the URL for a GroupMe image to be used as the bot's avatar
- **callback_url** (*str*) – the URL to which each group message will be POSTed

Returns the new bot

Return type `Bot`

`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.object.responses.`**`User`**(***kwargs*)

A GroupMe user.

This is you, as determined by your API key.

classmethod `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`

classmethod `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`

nickname
Your user name.

The `attachments` Module

The `listers` Module

class `groupy.object.listers.`**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`

first
The first element in the list.

last
The last element in the list.

class `groupy.object.listers.`**`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

CHAPTER 2

Indices and tables

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