Groupy Documentation

Release 0.5.6

Robert Grant

April 29, 2015

Contents

The simple yet powerful wrapper for the GroupMe API.

CHAPTER 1

Table of Contents

1.1 Introduction

1.1.1 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.

1.1.2 About Groupy

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

It is composed of two main parts:

- API wrapper (`groupy.api`)
- Object abstraction (`groupy.object`)

Current Features

Groups

- Create, update, and destroy your own groups
- List and filter your current and former groups
- Add and remove members from your current groups
- List and filter group members
- List and filter group messages

Members

- List and filter all known members
- List and filter direct messages
- Post direct messages to members

Messages

- Collect all messages from a group or member
- Like and unlike messages (even direct messages!)
- List and filter members who liked a message
- Inspect and create attachments

Bots

- List and filter your bots
- Use your bots to post messages
- Create, update, and destroy bots

Users

- Get your user information
- Enable and disable SMS mode

Planned Development

(in no particular order)

- Unit tests
- Installation via pip
- More direct way to add and remove yourself from groups
- Remove multiple members in one method call
- Porcelain for checking results of adding members
- Automatic updating of object attributes without the need to re-fetch objects
- Member objects that are aware of membership in all groups
- Additional ways to access objects
- More convenience methods instead of accessing API attributes directly
- Documentation about the API wrapper package
- Python 2.7 support

1.2 Installation

1.2.1 Prerequisites

To get started, you'll need to *get an account* at Groupme.com.

Got it? Great!

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.
4. Copy your access token.

You must also *create a key file*.

1. Paste your access token into a new file.
2. Save it as `.groupy.key` in your user's home directory.

Lastly, ensure you're running Python >= 3! Now you're ready to install Groupy!

1.2.2 Instructions

Below are instructions for various ways of performing installation.

Using `pip`

```
$ pip3 install GroupyAPI
```

From Source

Basic Steps

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

Note: Your package directory may be elsewhere. For help determining the correct location, see this StackOverflow question.

With `git`

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 above
```

Without `git`

If you don't have `git` installed, ask yourself why?

If you're satisfied with your answer to that question and you're still reading this section, fine. You don't *need* `git`. You can download it as a ZIP file.

- master branch
- dev branch

Installation is a simple matter of unzipping the file and copying over the `groupy` directory to your `Python3` package directory.

```
$ 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 above
```

1.2.3 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 whether your key file (`.groupy.key` by default) contains your API token, and that `groupy/config.py` contains a definition for `KEY_LOCATION` that correctly specifies the location and name of your key file.

1.3 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!

1.3.1 Listing Things

The most basic operation is listing things. `Groups`, `Members`, and `Bots` 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 additions.

You can read more about the types of lists used by **Groupy** in the *Advanced Usage* section, but for the remainder of this page, the following truth should suffice.

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

1.3.2 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.members()
```

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')
>>> 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/123456789'
>>> 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` attributes cannot be changed. Some of the more important attributes 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
```

1.3.3 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 as a `datetime.datetime` instance, as well as information about the member who posted it. Of course, messages can have text and attachments.

```
>>> message = messages.newest
>>> print(message.created_at)
2014-4-29 12:19:05
>>> message.user_id
'0123456789'
>>> message.name
'Kevin'
>>> message.avatar_url
'http://i.groupme.com/123456789'
>>> message.text
'Hello'
>>> message.attachments
[Image(url='http://i.groupme.com/123456789')]
```

Note: Not every message will have text and not every message will have attachments but every message must have one or the other.

Note: Although the majority of messages will have just one attachment, there is no limit on the number of attachments. In fact, despite that most clients are incapable of displaying more than one of each type of attachment, the API doesn't limit the types of attachments in any way. For example, a single message might have two images, three locations, and one emoji, but it's not likely that any client would show them all or handle the message without error.

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 membership changes (entering/leaving, adding/removing), group updates (name, avatar, etc.), and member updates (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 among 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.

1.3.4 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/123456789'
```

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'
```

1.3.5 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)
...
```

1.3.6 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/123456789
>>> 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.

1.3.7 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 *(recommended)*, 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.

1.4 Advanced Usage

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

1.4.1 Filter Lists

`FilterLists` are exactly like the built-in `list` but with some convenient additions.

`first` and `last`

`first` and `last` are merely convenience properties. `first` corresponds to the item at index `0`, while `last` corresponds to the item at index `-1`.

```
>>> from groupy.object.listers import FilterList
>>> fl = FilterList(range(1, 11))
>>> fl
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> fl.first
1
>>> fl.last
10
```

One important difference, however, is when there are no elements in the list.

```
>>> fl = FilterList()
>>> fl
[]
>>> print(fl.first)
None
>>> fl[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> print(fl.last)
None
>>> fl[-1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Note that no exception is raised and `None` is returned instead.

filter()

The `filter()` method parses its keyword arguments as filtering criteria. Only the items meeting all criteria are returned.

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`.

This is probably better explained with 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
```

1.4.2 Attachments

Attachments are a common part of `Messages` and there are several different types. Currently, **Groupy** supports the following types of attachments:

- `Location` - for locations
- `Image` - for images
- `Mentions` - for "@" mentions
- `Emoji` - for emoticons
- `Split` - for splitting bills[1]

For all other types of attachments (such as those introduced in the future) there exists a `GenericAttachment`.

Types

The section covers the various types of attachments and how to create them.

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`.

```
>>> from groupy import attachments
>>> loc = attachments.Location('My house', lat=34, lng=-84)
>>> loc
Location('My house', lat=34, lng=-84)
>>> loc.name
'My house'
>>> loc.lat, loc.lng
(34, -84)
```

[1] Split attachments are depreciated.

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, use the `file()` method.

```
>>> from groupy import attachments
>>> image_attachment = attachments.Image.file(open(filename, 'rb'))
>>> image_attachment
Image(url='http://i.groupme.com/123456789')
>>> image_attachment.url
'http://i.groupme.com/123456789'
```

We can see that the image has been uploaded in exchange for a URL via the GroupMe image service.

To fetch the actual image from an image attachment, simply use its `download()` method. The image is returned as a `Pillow Image`, so saving it to a file is simple.

```
>>> image_file = image_attachment.download()
>>> image_file.save(filename)
```

Mentions

`Mentions` are a new type of attachment and have yet to be documented. 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`. Let's find a good example to demonstrate mentions.

```
>>> from groupy import Group
>>> message = None
>>> mention = None
>>> for g in Group.list():
...   for m in g.messages():
...     for a in m.attachments:
...       if a.type == 'mentions' and len(a.user_ids) > 1:
...         message = m
...         mention = a
...         break
>>> message.text
'@Bill hey I saw you with @Zoe Childs at the park!'
>>> mention.user_ids
['1234567', '5671234']
>>> mention.loci
[[0, 5], [25, 11]]
```

As you can see, 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`.

```
>>> for uid, (start, length) in zip(mention.user_ids, mention.loci):
...   end = start + length
...   uid, message.text[start:end]
...   member = message.group.members().filter(user_id=uid).first
...   member.uid, member.nickname
('1234567', '@Bill')
('1234567', 'Bill')
('5671234', '@Zoe Childs')
('5671234', 'Zoe Childs')
```

To create a mention, simply pass in a `list` of user IDs and an equally-sized `list` of loci.

```
>>> from groupy.attachments import Mentions
>>> Mentions(['1234567', '2345671'], [[0, 4], [5, 3]])
Mentions(['1234567', '2345671'])
```

Emojis

Emojis are relatively undocumented but frequently appear in messages. More documentation will come as more is learned.

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.

`Split` attachments have a single attribute: `token`.

Sending Attachments

To send an attachment along with a message, simply append it to the `post()` method as another argument.

```
>>> from groupy import Group
>>> from groupy.attachment import Location
>>> loc = Location.create('My house', lat=33, lng=-84)
>>> group = Group.list().first
>>> group.post("Hey meet me here", loc)
```

If there are several attachments you'd like to send in a single message, simply keep appending them!

```
>>> from groupy.attachment import Image
>>> img = Image.file('front-door.png')
>>> group.post("I said meet me here!", loc, img)
```

Alternatively, you can collect multiple attachments into an `iterable`.

```
>>> attachments = [img, loc]
>>> group.post("Are you listening?", *attachments)
```

1.5 Developer Docs

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

1.5.1 The `api` Package

The `api.endpoint` Module

The `api.errors` Module

The `api.status` Module

1.5.2 The `object` Package

The `object.responses` Module

The `object.attachments` Module

The `object.listers` Module

1.5.3 The `config` Module

1.6 Change Log

1.6.1 v0.5.3 (September 19, 2014)

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

1.6.2 v0.5.2 (September 14, 2014)

- Now installable via *pip3*:

```
$ pip3 install GroupyAPI
```

1.6.3 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

1.6.4 v0.5.0 (August 20, 2014)

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

1.6.5 v0.4.0 (August 18, 2014)

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

1.6.6 v0.3.1 (August 14, 2014)

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

1.6.7 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!

1.6.8 v0.2.0 (August 11, 2014)

- Added MessagePager class for returning lists of messages

1.6.9 v0.1.3 (August 10, 2014)

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

1.6.10 v0.1.0 (August 9, 2014)

- Initial release

CHAPTER 2

Indices and tables

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