LNIE API Documentation

Release 0.7.0

Kim Tae Hoon

Apr 26, 2017

Contents

CHAPTER 1

Introduction

line is a python library that allow you to send and receive a LINE message. With *line* you can use LINE on any systems like Ubuntu or make your owb LINE bot which will automatically reply for your message!

Enjoy *line* and *May the LINE be with you...*

Warning: Some codes are removed because of the request of LINE corporation. (2014.08.08)

CHAPTER 2

Key Features

- login to LINE server
- get a list of *contact*, *group* or chat *room*
- send and receive a *message* or *sticker*
- invite, join or leave a *group* or *room*
- *longPoll* method which will allow you to make a LINE bot easily

CHAPTER 3

Todo

- Sending a Image
- More usable methods and objects

CHAPTER 4

Installation

First, you need to install **Apache Thrift**. Install instructions are here. (This might take some time...)

Next:

```
$ pip install line
```

Or, you can use:

```
$ easy_install line
```

Or, you can also install manually:

```
$ git clone git://github.com/carpedm20/line.git
$ cd LINE
$ python setup.py install
```

CHAPTER 5

API Documentation

LINE Episode I: A New Hope

If you are a core pythonic programmer, you can jump into writing the code right away! But if you are not familiar with Python, you should read this tutorial first before proceeding to the more details of *line*. Now, this manual will git you a quick introduction on how you can send a message and do other things with *line*

Part 1: Login and Pin authentication

Let's start with login to LINE and pass through a pin authentication.

```
>>> from line import LineClient
>>> client = LineClient("carpedm20@gmail.com", "xxxxxxxxxx")
Enter PinCode '7390' to your mobile phone in 2 minutes
>>> client = LineClient("carpedm20", "xxxxxxxxxx")
Enter PinCode '9779' to your mobile phone in 2 minutes
```

Warning: You will failed to login because of the request of LINE corporation. (I have to remove some codes) However, you can use this library by login with *authToken*. The instruction about *authToken* login is explained in bellow paragraphs.

As you can see, you can login by making a *LineClient* instance and pass your email and password as parameters. If you have a NAVER account and link it to LINE account, you can login with your NAVER account!

Then, you will see a *PinCode* which you have to put in to your mobile phone to authenticate your *LineClinet* instance as a desktop Line client. This number will be expired in 2 minutes, so don't be lazy!

If you enter your *Pincode* to your mobile phone, now you can see your *authToken* which will notify your LINE session.

```
>>> authToken = client.authToken
>>> print authToken
DJg5VZTBdkjMCQOeodf4.9guiWkX1koTnwiGNVkacva.49blBzv5W9ex/
↪2M06QQofByLxigMCAnnGfmTOAgY3wo=
```

With this *authToken*, you don't have to enter *Pincode* when you create a new *LineClient* instance!

```
>>> client = LineClient(authToken=authToken)
```

You can save your authToken in cache like *redis* or something else!

Note: If the client will be expired after a specific time (I couldn't find a exact time yet), so you have to get a new *authToken* after it is expired.

Part 2: Profile and Contacts

Now, let's see your profile to check whether *PinCode* authentication was successful or not.

```
>> profile = client.profile
>> print profile
<LineContact >
```

You might want to send any *message* to your friend that you have succeeded to login to LINE! But you have to choose which one to send a *message*.

```
>>> print client.contacts  # your friends
[<LineContact > <LineContact >]
```

Then, choose anyone to send a hello world message, and send it away!

```
>>> friend = client.contacts[0]
>>> friend.sendMessage("hello world!")
True
```

If you want to send an *image*, you can use *sendImage()* with specific path for image

```
>>> friend.sendImage("./image.jpg") # use your path for image to send
True
```

Or you can use an URL for image to send any *image* to your friends with *sendImageWithURL()*!

```
>>> friend.sendImageWithURL("https://avatars3.githubusercontent.com/u/3346407?v=3&
↪s=460")
True
```

If you want to send a *sticker* (which is one of the most fun features of LINE!)

```
>>> friend.sendSticker() # send a default sticker
True
>>> friend.sendSticker(stickerId="13",stickerPackageId="1",stickerVersion="100")
True
```

If you see *True* message, then it means message is successfully sended to your friend. If you want to receive 10 recent messages:

```
>>> messages = friend.getRecentMessages(count=10)
>>> print messages
[LineMessage (contentType=NONE, sender=None, receiver=<LineContact >, msg="hello␣
↪World!")]
```

I just make a one conversation with so I only get one message with *getRecentMessages* method.

Part 3: Rooms and Groups

There are two type of chat rooms in LINE, one is just a *room* with multiple users, and the other is *group* which have more features then room. For examle, *group* has its own name but *room* don't have any room for itself.

Now let's see a list of *group* and *room* you are participated in.

```
>>> print client.groups
[<LineGroup  #4>, <LineGroup  #1 (invited)>]
>>> print client.rooms
<LineRoom [<LineContact >]>, <LineRoom [<LineContact >, <LineContact >]>]
```

In the case of *client.groups* you can see a word *(invited)* and this represent that you are invited to a group but you didn't accep the invitation yet. '#{number}' means the number of members in the specific group. If you want to accept it:

```
>>> group = client.groups[1]
>>> group.acceptGroupInvitation()
True
```

Other methods are same as the case of *contact* like if you want to get a list of recent messages, use *getRecentMessages* method:

```
>>> messages = client.contacts[0].getRecentMessages(count=10)
>>> messages = client.groups[0].getRecentMessages(count=15)
```

If you have too much groups and want to find a specific group with its *name*:

```
>>> group = client.getGroupByName('GROUP_NAME')
>>> contact = client.getContactByName('CONTACT_NAME')
```

There are other methods in *contact*, *rooms* and *group* instances so I'll recommend you to take a look at the models section.

Part 4: Make your own bot

So, most of you may want to use *line* to make your LINE bot. I also started this project to make a bot, so let's talk about how to make our own bot. Below code is a basic structure of a LINE bot:

```
1  from line import LineClient, LineGroup, LineContact
2
3  try:
4      client = LineClient("ID", "PASSWORD")
5      #client = LineClient(authToken="AUTHTOKEN")
6  except:
7      print "Login Failed"
8
9  while True:
10     op_list = []
```

```
11
12    for op in client.longPoll():
13        op_list.append(op)
14
15    for op in op_list:
16        sender   = op[0]
17        receiver = op[1]
18        message  = op[2]
19
20        msg = message.text
21        receiver.sendMessage("[%s] %s" % (sender.name, msg))
```

One of the most important line is #12, and you might notice there is a new method named *longPoll*. This method pull a list of operations which should be handled by our LINE bot. There are various type of operations, but our interest might be *RECEIVE_MESSAGE* operation. This operation contain a new message sent by other *contact*, *room* or *group*. So we can get a received *message* and its *sender* by

```
sender   = op[0]
receiver = op[1]
message  = op[2]
```

LineClient

Introduction

This is the most important class to use LINE with python. You have to make an instance of *LineClient* first and have to give your *id* and *password* as a parameters to login to LINE server. Then you should enter *PinCode* to pass through *PinCode* authentication

```
>>> from line import LineClient
>>> client = LineClient("carpedm20@gmail.com", "xxxxxxxxxx")
Enter PinCode '7390' to your mobile phone in 2 minutes
>>> client = LineClient("carpedm20", "xxxxxxxxxx")
Enter PinCode '9779' to your mobile phone in 2 minutes
```

With *authToken* of your line instance, you don't have to enter *Pincode* everytime when you create a new *LineClient* instance.

```
>>> client = LineClient(authToken=authToken) # login with authToken
```

LineClient

Models

Introduction

This page intorduce you a list of core models which is used in LINE API. The name of each models tell you what it is intuitively. In most cases, you don't have to create this instances, but if you want to change *line*, I hope this documents wil help you to find what you want.

LineMessage

LineBase

LineContct

LineRoom

LineGroup

LineAPI

Introduction

This is a python wrapper of official LINE thirft API. There are other functions which is not implemented to *line* like *kickoutFromGroup* things, so you can add other API here and use it as your way.

LineAPI

Known issue

1. Garbage data with python Thrift

If you use methods like *curve.ttypes.Location* which get or send double type data through *Thrift*, you might get some garbage values.

Thre reason of this error is that *Thrift 0.9.1* installed via *pip* has an issue with serialization&deserialization of double type using CompactProtocol as described in here.

Below is a patch which is suggedsted by Wittawat Tantisiriroj (wtantisiriroj@gmail.com)

– Patch –

```
diff --git a/lib/py/src/protocol/TCompactProtocol.py b/lib/py/src/protocol/
↪TCompactProtocol.py
index cdec607..c34edb8 100644
--- a/lib/py/src/protocol/TCompactProtocol.py
+++ b/lib/py/src/protocol/TCompactProtocol.py
@@ -250,7 +250,7 @@ def writeI64(self, i64):

  @writer
  def writeDouble(self, dub):
-    self.trans.write(pack('!d', dub))
+    self.trans.write(pack('<d', dub))

  def __writeString(self, s):
    self.__writeSize(len(s))
@@ -383,7 +383,7 @@ def readBool(self):
  @reader
  def readDouble(self):
    buff = self.trans.readAll(8)
```

```
-     val, = unpack('!d', buff)
+     val, = unpack('<d', buff)
      return val

  def __readString(self):
```

CHAPTER 6

Indices and tables

- search

CHAPTER 7

Echo bot example

```
from line import LineClient, LineGroup, LineContact

try:
   client = LineClient("ID", "PASSWORD")
   #client = LineClient(authToken="AUTHTOKEN")
except:
   print "Login Failed"

while True:
   op_list = []

   for op in client.longPoll():
      op_list.append(op)

   for op in op_list:
      sender   = op[0]
      receiver = op[1]
      message  = op[2]

      msg = message.text
      receiver.sendMessage("[%s] %s" % (sender.name, msg))
```

CHAPTER 8

License

Source codes are distributed under BSD license.

CHAPTER 9

Author

Taehoon Kim / @carpedm20