Shawk Documentation

Release latest

Hawkins

Dec 08, 2018

Table of Contents

Shawk is a Python module designed to make sending and receiving SMS text messages easy for devices running Python. With internet of things compatibility in mind, this module has been designed to function fully on Raspberry Pi devices and be easy to use in that context of scripting.

If you're new to Shawk, I suggest you check out the *Getting Started* page.

CHAPTER 1

Module Index

- modindex

1.1 Getting Started

The idea of Shawk is to provide an easy-to-use interface for sending and receiving text messages. This getting started guide will give you a quick run-down on the highlights of how to use Shawk.

1.1.1 Installing Shawk

Installing Shawk is easy. Simply run *pip install shawk* to download and install the latest version.

To best understand how to use shawk, let's break it down into a few features:

- Creating a client
- Adding/removing contacts
- Sending messages
- Manually retrieving messages
- Automatically retrieving messages
- Defining behavior for new messages

1.1.2 Creating a client

The first thing you'll want to do after installing Shawk is create a client. This is the main interface you'll use in Shawk. You can define multiple clients, of course, but for the sake of this tutorial, we'll just use one.

If we say our Gmail login username is *"username@gmail.com"* and our password is *"password"*, then we can create our client like this:

```
import shawk

user = "username@gmail.com"
password = "password"
client = shawk.Client(user, password)
```

1.1.3 Adding/removing contacts

Adding contacts will be helpful for understanding who texted your client or specifying who you would like to text in an easy and readable manner. Email gateways are a bit wonky at best, so contacts require both a phone number and carrier. Additionally, you can specify a contact name to make things easier, but that is optional.

Say you want to add a contact Josh who uses Verizon and has a phone number of 1234567890.

You can add them as such:

```
client.add_contact(1234567890, 'Verizon', 'Josh') # Note: name/'Josh' is optional
```

You can similarly remove them in any of the following ways:

```
client.remove_contact(contact) # Where contact is some shawk.Contact object, which␣
↪you can get from messages
# or
client.remove_contact(number=1234567890)
# or
client.remove_contact(name='Josh')
```

1.1.4 Sending messages

You’re limited to texting only your contacts you’ve previously defined, or those whose explicit address you have obtained. This is, in-part, a restriction to prevent spam bots from abusing Shawk, but also a means of simplifying Number to Address mappings. What this means is, in order to send someone a text message, you can either add them as a contact or reply to a message they send you.

So, if you defined a contact, you can text them by specifying their name, number, or Contact:

```
client.send('Hey, Josh!', name='Josh')
# or
client.send('Hey, Josh!', number=1234567890)
# or
client.send('Hey, Josh!', contact)
```

1.1.5 Manually retrieving messages

There are two ways to get new messages in Shawk: Manually or Automatically.

We can get the new ones manually by first setting up our inbox and refreshing it:

```
client.setup_inbox(password) # We don't save your password, so send it again
client.refresh()
```

This will handle establishing the IMAP server connection for retrieving new messages to your Gmail account and manually retrieving the messages afterwards.

You can actually use a distinct Gmail account from the one you use to send messages by passing a user: string, but we won't focus on that for this simple tutorial. As always, read the rest of the docs if you'd like to know more about that.

1.1.6 Automatically retrieving messages

You can configure Shawk to automatically retrieve new messages for you pretty easily. This will cause the client to refresh its inbox periodically on time interval (found in client.refresh_interval).

To do this, setup your inbox as such:

```
client.setup_inbox(password)
client.enable_auto_refresh()

# or...
client.setup_inbox(password, auto=True)
```

Boom. Done. Your client now automatically pings the server and looks for new messages!

You can configure the time interval with *client.set_interval()*, which you can read more about in the docs.

1.1.7 Defining behavior for new messages

At this point, I hope you've asked yourself, "But what will Shawk do when it receives a message?" By default, Shawk will simply print the contents of the new messages when it finds them.

That's not particularly useful, so we'll let you dictate how Shawk *should* be used.

This default behavior can be overridden by defining one or more handler functions that receive a client, message, and a regex match object. These handler function are just callback functions with the *@client.text_handler(regex)* decorator, where regex is some regular expression in string form.

If no regex is provided, then the function is considered the default case handler, and will be used whenever no other handler's regex pattern match a received text.

Whew.

So, you can define your own behavior as follows:

```
c = shawk.Client('username@gmail.com', 'password')
c.setup_inbox('password')

@client.text_handler() # No arguments, so Shawk knows this is our default handler
def handler(client, msg):
    print("Hey, we're popular! %s texted us!" % msg.sender)
    print("Received: %s" % msg.text)

    client.send("I am replying to your text!", msg.sender)

# Or with a regex
@client.text_handler('^Print (.*)', 'i') # Starts with "print" (case insensitive␣
↪because of 'i' flag), matches text following.
def print_dot_z(client, msg, match, regex):
    print(match.group(1))
```

Naturally, you'll do something a bit more meaningful in your handler functions. But since they're just simple python functions, you've got free reign to interface with your scripts however you like.

Note that you can also create handlers that activate on Contacts instead of Regex. This is useful if you only want behavior to apply to a specific contact and consider text second, instead of the other way around.

You can do this like so:

```
josh_contact = client.add_contact(1234567890, 'Verizon', Josh)

@client.contact_handler(josh_contact)
def josh_handler(client, msg):
    print("This func only called if Josh texted us")
```

Also worth mentioning is that these are just normal functions, so you can define handlers that hook onto contacts within other handlers. I.e., a text_handler that when called will instantiate a contact_handler to continue the conversation with that specific contact after they reply. This is advanced behavior tho - so we'll cover that in another guide.

I hope you've found that Shawk is pretty easy to use, yet very powerful, since it allows your users to provide input and receive output via SMS, for free.

If you encounter any issues or have any questions, you can post them on GitHub://hawkins/shawk.

1.2 Configuring Gmail

1.2.1 SMTP

GMail should work with SMTP out of the box, but currently sending messages is limited to **roughly 100 messages per 24 hours**. After your limit is reached, the limit will be reset 24 hours later.

However, you can greatly increase these limits (**10,000 messages per 24 hours** by creating a Google Apps account. You can find out more about this here.

1.2.2 IMAP

1. Create a Google / Gmail account for your application (It doesn't have to be new, but I suggest you do not use your personal / work account).
2. Follow instructions to set up an App password here and be sure to select app "Other (custom name)" and enter anything to help you remember it. I.e., "shawk".

   Be careful though, as this password grants full access to your Google account! Do not share this with anyone and do not commit it!

   I recommend creating a *secure.ini* file which your scripts will read in your password from. Add this file to your *.gitignore* to prevent accidentally commiting it.
3. Follow instructions to enable IMAP access to Gmail here. You only need to complete Step 1, don't worry about Step 2. The Shawk library handles Step 2 for you.

1.3 Using Emoji

1.3.1 Syntax

Using Emoji in Shawk is pretty straightforward. Since Shawk utilizies the Emoji library on PyPi, the entire set of Emoji codes as defined by the unicode consortium is supported in addition to a bunch of aliases.

Unlike the Emoji library, we allow all aliases at all times. This means that you can say either ":thumbs_up_sign:" or ":thumbsup:" to insert the thumbs up emoji in your text messages.

1.3.2 Configuring

By default, both "demojizing" and "emojizing" are enabled. This means that when you receive a text message, it is "demojized", or emojis are translated to :: syntax, and sent messages are translated from :: syntax to unicode emojis.

To disable emojizing / demojizing, you can run the following code:

```
# Disable demojizing
client.disable_demojize()

# Disable emojizing
client.disable_emojize()
```

You can similarly re-enable either setting:

```
# Re-enable demojizing
client.enable_demojize()

# Re-enable emojizing
client.enable_emojize()
```

1.3.3 Example

Since emojizing is enabled by default, sending an emojized message is as simple as this:

```
# Create and configure client
client = shawk.Client(username, password)
contact = client.add_contact(number=1234567890, carrier='Verizon', name='Somebody')

# Send the message
client.send('Testing :thumbs_up_sign:', contact=contact)
```

1.4 Client

Client is the bread and butter of the Shawk framework. You'll use this to connect to a Gmail account in both SMTP and IMAP protocols.

Typically the workflow will resemble creating a Client, adding contacts, configuring an inbox for IMAP, and listening for messages.

1.5 Contact

Contact defines the structure of the contacts used in Shawk. These contain an address, a number, and an optional name.

The real magic isn't so much in the contact, but in how the Client uses the contact for simplifying receiving and sending messages.

1.6 Message

Message defines the structure of the messages used in Shawk. These contain a sender, a text body, and an international date format of when the message was received (only valid for received messages).

1.7 Gateways

Gateways defines the SMS Carrier and Gateway support for Shawk.

This contains a dictionary of Carrier -> Gateway mappings, as well as a regex for matching an email address to a supported domain.