mailbot Documentation

Release 0.4dev

Mathieu Agopian

Mar 21, 2017

Contents

MailBot is a little python library that let's you execute previously registered callbacks on reception of emails.

This allows you to do fancy things like doing API calls, running scripts, sending notifications, ...

CHAPTER 1

Features

MailBot does its best to:

- be fully tested
- apply the pep8 recommendations
- be lightweight, concise and readable

MailBot connects to a mail server using the IMAP protocol, thanks to the excellent IMAPClient from Menno Smits.

CHAPTER 2

Other resources

Fork it on: http://github.com/magopian/mailbot/

Documentation: http://mailbot.rtfd.org/

CHAPTER 3

Installing

From PyPI:

```
pip install mailbot
```

From github:

```
pip install -e http://github.com/magopian/mailbot/
```

CHAPTER 4

Usage

You first need to instantiate MailBot, giving it informations to connect to your IMAP server. MailBot uses IMAPClient, and as such takes the same parameters.

You also need to provide the username and password. Here's an simple example:

```
from mailbot import MailBot, register

from mycallbacks import MyCallback


mailbot = MailBot('imap.myserver.com', 'username', 'password')

# register your callback
register(MyCallback)

# check the unprocessed messages and trigger the callback
mailbot.process_messages()
```

You may want to place the `process_messages` in a loop, a celery task, or in a cron job, tu regularly check new messages and process them.

CHAPTER 5

Registering callbacks

callbacks.py:

```
from mailbot import register, Callback


class MyCallback(Callback):

    def trigger(self):
        print("Mail received: {0}".format(self.subject))

register(MyCallback)
```

By default, callbacks will be executed on each and every mail received, unless you specify it differently, either using the 'rules' attribute on the callback class, or by registering with those rules:

CHAPTER 6

Providing the rules as a parameter

Here's a callback that will only be triggered if the subject matches the pattern 'Hello ' followed by a word, anywhere in the subject (it uses `re.findall`):

```
from mailbot import register, Callback


class MyCallback(Callback):
    rules = {'subject': [r'Hello (\w)']}

    def trigger(self):
        print("Mail received for {0}".format(self.matches['subject'][0]))

register(MyCallback)
```

This callback will be triggered on a mail received with the subject "Hello Bryan", but won't if the subject is "Bye Bryan".

CHAPTER 7

Providing the rules when registering

The similar functionality can be achieved using a set of rules when registering:

```
from mailbot import register, Callback


class MyCallback(Callback):

    def trigger(self):
        print("Mail received for %s!" self.matches['subject'][0])

register(MyCallback, rules={'subject': [r'Hello (\w)']})
```

CHAPTER 8

How does it work?

When an email is received on the mail server the MailBot is connected to (using the IMAP protocol), it'll check all the registered callbacks and their rules.

If each provided rule (either as a class parameter or using the register) matches the mail's subject, from, to, cc and body, the callback will be triggered.

Mails are flagged according to their state, in the `process_messages` method:

- unread (unseen): mail to be processed by MailBot
- read (seen):
  - starred (flagged): MailBot is checking callbacks, and triggering them if needed, the mail is being processed
  - not starred (unflagged): MailBot is done with this mail, and won't process it anymore

Specifying a timeout

To avoid a mail from staying in the "processing" state for too long (for example because a previous `process_message` started processing it, but then failed), you may specify a `timeout` parameter (in seconds) when instantiating MailBot:

```
from mailbot import MailBot


mailbot = MailBot('imap.myserver.com', 'username', 'password', timeout=180)
```

This doesn't mean that the mail will be reset after 3 minutes, but that when `process_messages` is called, it'll first reset mails that are in the processing state and older than 3 minutes.

CHAPTER 9

Specifying rules

Rules are regular expressions that will be tested against the various email data:

- `subject`: tested against the subject
- `from`: tested against the mail sender
- `to`: tested against each of the recipients in the "to" field
- `cc`: tested against each of the recipients in the "cc" field
- `body`: tested against the (text/plain) body of the mail

If no rule are provided, for example for the "from" field, then no rule will be applied, and emails from any sender will potentially trigger the callback.

For each piece of data (subject, from, to, cc, body), the callback class, once instantiated with the mail, and the `check_rules` method called, will have the attribute `self.matches[item]` set with all the captures from the given patterns, if any, or the full match.

Here are example subjects for the subject rules: `[r'Hello (\w+), (.*)',r'[Hh]i (\w+)]`

For each of the following examples, `self.matches['subject']` will be a list of all the captures for all the regular expressions.

If a regular expression doesn't match, then it'll return an empty list.

- 'Hello Bryan, how are you?': [('Bryan', 'how are you?')]
- 'Hi Bryan, how are you?': ['Bryan']
- 'aloha, hi Bryan!': ['Bryan']
- 'aloha Bryan': rules not respected, callback not triggered, []

Here are example subjects for the subject rules (no captures): `[r'Hello \w+', r'[Hh]i \w+]`

- 'Hello Bryan, how are you?': ['Hello Bryan']
- 'Hi Bryan, how are you?': ['Hi Bryan']
- 'aloha, hi Bryan!': ['hi Bryan']

- ‘aloha Bryan’: rules not respected, callback not triggered, []

CHAPTER 10

Rules checking

A callback will be triggered if the following applies:

- for each item/rule, **any** of the provided regular expressions matches
- **all** the rules (for all the provided items) are respected

Notice the "any" and the "all" there:

- for each rule, there may be several regular expressions. If any of those match, then the rule is respected.
- if one rule doesn't match, the callback won't be triggered. Non existent rules don't count, so you could have a single rule on the subject, and none on the other items (from, to, cc, body).

As an example, let's take an email with the subject "Hello Bryan", from "John@doe.com":

```
from mailbot import register, Callback


class MyCallback(Callback):
    rules = {'subject': [r'Hello (\w)', 'Hi!'], 'from': ['@doe.com']}

    def trigger(self):
        print("Mail received for {0}".format(self.matches['subject'][0]))

register(MyCallback)
```

All the rules are respected, and the callback will be triggered

- subject: even though 'Hi!' isn't found anywhere in the subject, the other regular expression matches
- from: the regular expression matches
- to, cc, body: no rules provided, so they aren't taken into account

The last bullet point also means that if register a callback with no rules at all, it'll be triggered on each and every email, making it a "catchall callback".