slackspread

Release 0.0.3

Sep 19, 2019

Modules

CHAPTER 1

Introduction

This package contains two classes that are basic wrappers around slack and google spread APIs meant to be used to build more complex programs (including slack bots) upon them. Installing is simply made using pip:

```
pip3 install --upgrade slackspread
```

Only installation will be covered here, head over to the specific documentation about the slack bot here and the google spreadsheet wrapper here.

CHAPTER 2

Connect to slack API with `SlackBot()`

If using the slack API with python is new to you, head over here to get a nice introduction. When everything is in place and you've got a slack bot token from Slack, store it as an environment variable (`MYBOT_TOKEN` in the below exemple) and initiate the slack web client on python by giving the variable's name as argument.

```
from slackbot import SlackBot
mybot = SlackBot(token = "MYBOT_TOKEN")
```

CHAPTER 3

Connect to google spread API with `Gspread()`

`Gspread` revolves on a json credentials file to authenticate on google spreadsheets API. The `init` method of `Gspread` needs both a credentials json file and a set of environment variables to replace sensitive values on the json credentials file.

Details on have to obtain that file can be found here. This file would generally have the following form

```
{
   "type": "service_account",
   "project_id": "#project id",
   "private_key_id": "#private key id",
   "private_key": "#private key",
   "client_email": "#client email",
   "client_id": "#client id",
   "auth_uri": "https://accounts.google.com/o/oauth2/auth",
   "token_uri": "https://oauth2.googleapis.com/token",
   "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
   "client_x509_cert_url": "client certification url"
}
```

Some of the fields in the `credentials.json` must be kept private (in the above example, the ones preceded with `#`), so we strongly advise to replace those fields with empty or non-explicit values in the json file, especially if you're to push it to git repository, and use environment variables to store those fields instead. The `Gspread` object *will* look for environment the following variables at initialisation:

```
PROJECT_ID
PRIVATE_KEY_ID
PRIVATE_KEY
CLIENT_ID
CLIENT_EMAIL
CLIENT_X509_CERT_URL
```

More precisely, `Gspread()` takes three arguments :

name name of spreadsheet to connect to

environ_prefix prefix for above listed environment variables

credentials path to json file containing credentials with *false* sensitive fields

The prefix is combined to the above-listed variables names to build the environment variables names that the class will look for. The `init` method will replace the corresponding fields in the `credentials.json` dictionary with the values read from those variables.

Say your project's name is something like `my daily budget`. You would first store the following environment variables

```
BUDGET_PROJECT_ID
BUDGET_PRIVATE_KEY_ID
BUDGET_PRIVATE_KEY
BUDGET_CLIENT_ID
BUDGET_CLIENT_EMAIL
BUDGET_CLIENT_X509_CERT_URL
```

And store somewhere a `credentials.json`, let's say at `~/.gscredits/budget-credentials.json` (on which you would have replaced the sensitive fields with non-explicit or wrong values). All you need to do is call `Gspread` with the following syntax :

```
from easyspread import Gspread
budget_spread = Gspread(
    name = 'my daily budget',
    environ_prefix = 'BUDGET',
    credentials = "~/.gscredits/budget-credentials.json"
)
```

Using environment variables makes it possible to have your code working while being safe on a network server, since the json file is stored without any sensitive data in it and the sensitive values are protected as environment variables.

Note that you can use the same credits for different spreadsheets. Each set of credits corresponds to a single project on google cloud, but can connect to *any* spreadsheet, provided it was authorized to in the spreadsheet's parameters.

3.1 Slackbot

class slackbot.**SlackBot**(*token*)

Bases: object

Slack web client to build a python slack bot.

`Slackbot` is a simple wrapper around basic API calls to slack API. Instantiation needs an environment variable storing the connexion token. When initiating a `Slackbot` instance, the bot id is read from slack API and stored as attribute.

Parameters **token** (*str*) – name for environment variable storing slack application token

Variables

- **client** – the slack web client instance with which API calls are made
- ***mention_regex*** – a regex to identify a direct mention in a text
- **id** – the bot's id

static closing_time(*closing_hour: str*, *rtm_client=None*) → None

Close an RTM client if the current hour is later than the given closing hour.

Parameters

- **closing_hour** (*str*) – hour after which RTM client must be stopped. Must be in `HH:mm` string format
- **rtm_client** (`slack.RTMClient()`) – The RTM client to stop (see above to see usage)

mention_regex = '<@(|[WU].+?)>(.*)'

static parse_mention(*message_text*, *regex*) → tuple
From a message, find and extract a mention and the related text.

Result is a length-two tuple of the form `(user_id, text)`. If no mention is found in the message, the result is `(None, text)`.

Parameters

- **message_text** (*str*) – the body of a message
- **regex** (*regex*) – the regex identifying a mention. Using `SlackBot.mention_regex` will usually suffice

send_message(*channel*, *message*) → None
Immediate wrap around `slack.WebClient.chat_postMessage()`

users_list() → list
Get the list of non-deleted users.

A wrapper around `slack.WebClient.users_list`. It returns a flat list of dictionaries, with keys `name`, `id` and `email`. `id` is used to tag people in messages with the syntax `<@%s> % id`.

3.2 Easyspread

class easyspread.**Gspread**(*name: str*, *environ_prefix: str*, *credentials: str*)
Bases: `object`

Google spreadsheet handler.

Parameters

- **name** (*str*) – name of spreadsheet to connect to
- **environ_prefix** (*str*) – prefix of environment variables containing valid credits
- **credentials** (*dict*) – path to json containing credentials

Variables

- **credentials** – valid credentials to manipulate a spreadsheet through the API
- **name** – attribute to store argument `name` (spreadsheet name)
- **sheets** – stores the worksheets opened with `self.open_worksheet()` as a dict, with the worksheet's title as key

Environment variables need to have the form `PROJECT_<varname>`

```
PROJECT_ID
PRIVATE_KEY_ID
PRIVATE_KEY
CLIENT_ID
CLIENT_EMAIL
CLIENT_X509_CERT_URL
```

append_row(*worksheet: str*, *item: list*)
Append a row to a worksheet.

The method returns `self` so that calls to `append_row` can be chained.

Parameters

- **worksheet** (*str*) – worksheet to append row to
- **item** (*list*) – values to append

delete_records(*worksheet: str*, *rows: int = 1*)
Delete records by resizing a worksheet.

static get_credentials(*prefix: str*, *file: str*)
Read private keys from environ and update json credentials

open_worksheet(*title: str*)
Open a worksheet and store it in `self.sheets` dictionary

read_records(*worksheet: str*) → list
Return records from a worksheet.

The method skips the header row.

class easyspread.**NetworkWait**(*delay: int = 10*, *hostname: str = 'www.google.com'*)
Bases: `object`

Wait (infinitely) for an internet connexion.

Parameters

- **delay** (*int*) – seconds between each ping trial
- **hostname** (*str*) – default `www.google.com` : the hostname to ping

Variables **connected** – indicates that a connexion was found

static ping(*hostname: str*) → bool
Ping the hostname to see if an internet connexion is available.

stubborn_ping(*delay: int*, *hostname: str*) → None
Ping every `delay` seconds until a connexion is found.

3.3 Release notes

3.3.1 Version 0.0.3

Improve the README, write docstrings and usage documentation.

Improve test coverage and mock every API call during tests so that tests can be made without network.

3.3.2 Version 0.0.2

Delete an intempestive print not supposed to be here during execution.

3.3.3 Version 0.0.1

Initial release !

3.4 Contributing

All contributions to the package are welcome ! It is likely that the repository settings on Gitlab are not perfectly tuned for anyone to contribute, so feel free to contact the package's maintainer if you encounter any problem.

Since `slackspread` is a simple and little package, there are no real rules when contributing, rather some simple guidelines to keep it clean and sound.

3.4.1 Coding convention

Code

For python's code, we generally follow PEP8 convention, for many of our contributors use PyCharm as IDE, which automatically checks for that convention. As can be seen below, we'd be happy that you add type indications to a function's signature.

Docstrings

We mostly follow the rules given by `sphinx` documentation for docstrings, about which you can read here.

- present tense and imperative mode is preferred for the first sentence, which should end with a period
- always use triple quotes for docstrings (`"""Your line."""`) it makes it

easier for next contributors to add new lines to the docstring. - for multi-line docstrings, please do not leave a starting blank line - specify argument types and return with `:param x:`, `:type x:`,

`:return: something` and `:rtype: str` (for instance)

- try as much as possible to provide at least one usable example
- do not start an argument specification with a cap
- if you write more than one description paragraph, please put all but the first *after* the function's arguments specification, but *before examples*

A typical docstring would thus be :

```
def a_wonderful_function(first_arg: int, second_arg: str) -> None:
    """Print first the second, secondly the first."""
    print("A string %s with a number %d" % (second_arg, first_arg))
    return None

def another_function(one_arg: str, prefix: str = 'a prefix : ') -> str:
    """Add a prefix to a string.

    This function is not really useful.

    :param one_arg: the string to be prefixed
    :type one_arg: str
    :param prefix: default ``'a prefix'`` : the prefix to use
    :type prefix: str
    :return: the input string prefixed with the given prefix
    :rtype: str

    Here goes the rest of the description, details, etc.
```

(continues on next page)

(continued from previous page)

```
    :Example:

    >>> another_function('rosso', 'pesto')
    """
    return None
```

3.4.2 Pull requests

A default template is provided for merge requests. It is not compulsory to use it, it is only meant to ease reviewing your changes and keeping things simple and straightforward.

3.4.3 Commit conventions

The more your commits will follow these conventions, the easier it will be to understand what you did and navigate through the repo's history.

- use present tense and imperative mode for the first line of your commit : instead of "Added new function" or "Adds new function", write "Add new function"
- keep the first line less than 80 characters and without period at the end
- feel free to explain with as many lines as needed after the first line, but always leave the second one empty
- using emojis to prefix commits is highly appreciated, not only for fun but also because it adds another layer of easily understood contextualisation

About emojis, feel free to use whichever you want to, here are the ones that are the most often used in the package :

- or When your changes improve performance
- When you modify comments, docstrings or non-python files
- When you modify the documentation files (in `docs/source`)
- When you change or fix some object mechanism
- When you refactor something (more than a little change)
- When you resolve a bug starting from `develop`
- When you resolve a bug starting from `master`
- When you delete or remove something (in the code or a whole file)
- When you improve format and code's appearence (indentation, etc.)
- or When you move files upward or downward in the folder's tree
- and When you move files from a folder to another in the same level of the folder's tree
- (or any clock) When the changes are temporary or need to be refined
- When the changes are a work in progress and call for subsequent changes
- When you add tests or make broken tests work
- or When some work or improvement has been done (in short, when none of the above apply)
- When a new functionality has been completed
- and Reserved for releases of merge commits

- when you did something you're very proud of
- genindex

Python Module Index

Index