MOE Documentation

Release 0.0.1

Camilo Garcia La Rotta

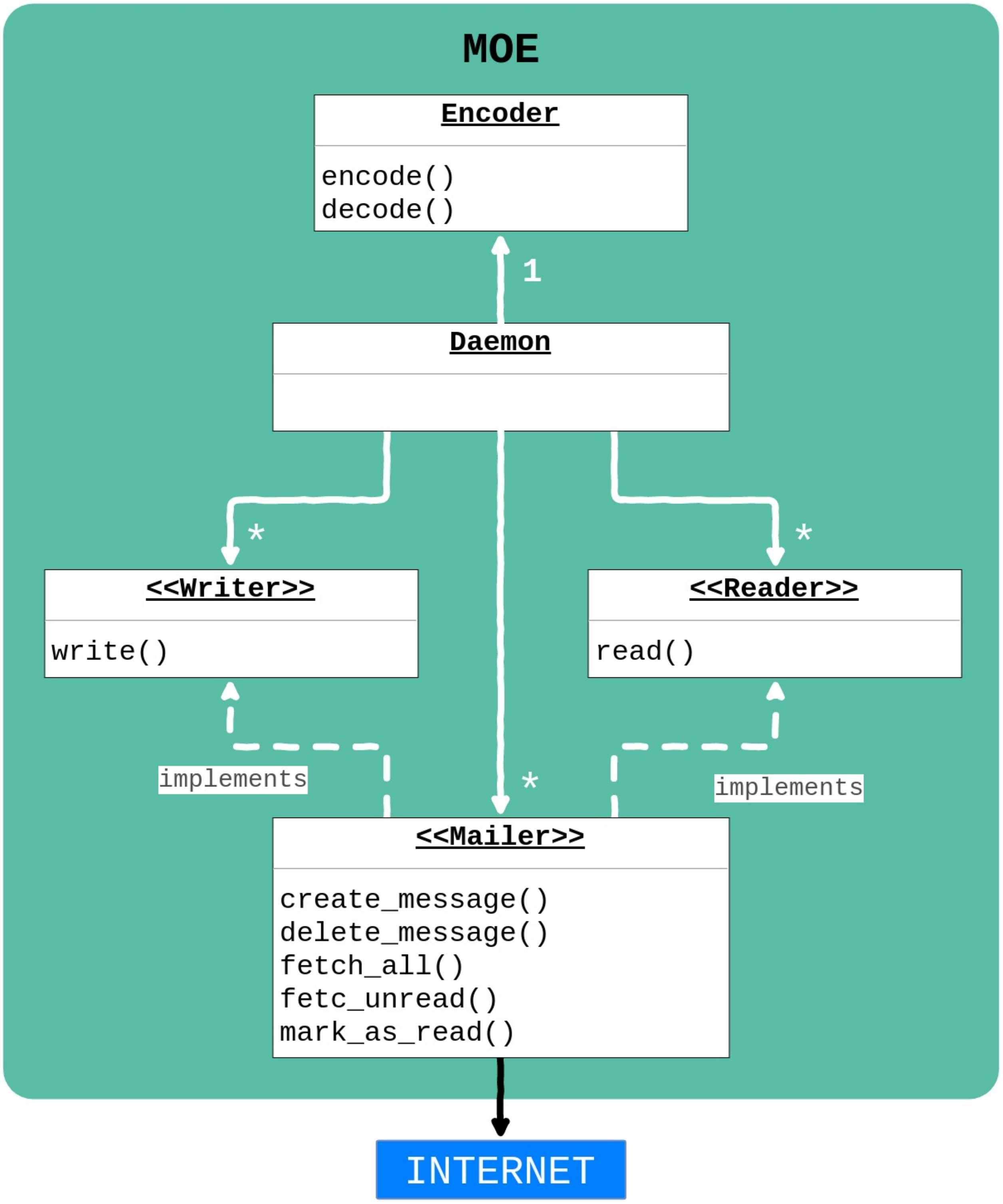
Apr 23, 2019

Architecture:

CHAPTER 1

See Also

1.1 Architecture



One of `MOE`s goals is to be open for extension by whomever wants to tinker with it. Hence, there are only 5 software components in `MOE`: a **Reader** to input messages to `MOE`, a **Writer** to output messages locally, a **Mailer** to output messages onlinei, an **Encoder** to translate to and from any cypher code, and a **Daemon** to control them all.

1.1.1 Software

Reader Interface

Any class implementing a `read()` method can be used as a **Reader**. It's any means by which a string can be inputed to `MOE`: e.g. camera, button, CLI, email, text message. An example of an already implemented **Writer** is: **Gmailer**.

Writer Interface

Any class implementing a `write()` method can be used as a **Writer**. It's any means by which a string can be outputed from `MOE`: e.g. sound, printer, email, text message. Examples of already implemented **Writer**s are: **Echoer** and **Gmailer**.

Mailer Interface

A **Mailer** is any class that mplements **Reader** and **Writer**, as well as the following methods:

```
create_message()      delete_message()
fetch_all()           fetch_unread()
mark_as_read()
```

It is any means by which a string can be outputed through the internet from `MOE`: e.g. API's. An example of an already implemented **Mailer** is: **Gmailer**.

MOE's email dictionnary

The data unit for a **Mailer**, refered to as `MOE`'s email dictionnary in the documentation, is the following:

```
{
  id: int,          # id by which the email is identified by the Mailer
  content: str,     # body of the email
  labels: [str],    # labels of the email (by default the 'MOE' label is used as an␣
↪index)
  unread: bool      # weter the email has already been read by MOE
}
```

Encoder Class

The **Encoder** class receives a CSV with the dictionnary of the cypher code you want to use (e.g. MORSE.csv). Its `encode()` and `decode()` methods are called by **Daemon**.

Daemon

The entrypoint for `MOE`. It is the single point of control for all of the components.

Dependencies

`MOE` is coded in Python3.6. It uses f-strings and type inting to facilitate documentation. For the full list of dependencies, please check requirements.txt

1.1.2 Hardware

The following components are just a reference. You may choose the brand/model of your choice.

- Raspberry Pi (any model)
- Breadboard
- Jumper Cables
- Buttons
- LEDs
- Buzzer
- Optional
  - Wifi Dongle or Ethernet cable
  - Camera
  - Thermal Printer

1.2 Components

1.2.1 moe.encoder module

The daemon module contains all the logic required for the encoding/decoding of any cypher code.

class moe.encoder.**Encoder**(*file*, *default_space=' '*, *default_separator='¶'*, *default_value=''*)

Bases: `object`

Encodes and decodes text from a given dictionnary passed through CSV.

Parameters

- **file** (*str*) – The csv file containing the encoding: LETTER, CODE. It can have an entry for default space called SPACE. i.e. the characte to separate encoded words It can have an entry for default separator called SEPARATOR. i.e. the character to separate encoded letters It can have an entry for default non supported values called DEFAULT.
- **default_space** (*str, optional*) – Defaults to ' '. If no default space is present in the CSV file, it can be passed here.
- **default_separator** (*str, optional*) – Defaults to '¶'. If no default space is present in the CSV file, it can be passed here.
- **default_value** (*str, optional*) – Defaults to ''. If no default value is present in the CSV file, it can be passed here.

decode(*coded_string*)

Decode a given coded string.

Parameters **coded_string** (*str*) – The string to be decoded.

Returns The string containing the equivalent text.

Return type str

encode(*text*)

Encode a given text with the configured dictionnary.

Parameters **`text`** (*`str`*) – The text to be encoded.

Returns The encoded string.

Return type str

1.2.2 moe.daemon module

The daemon module contains all the logic required for the daemon controlling MOE to work

1.2.3 moe.mailer package

moe.mailer.gmailer module

The gmailer module contains a Mailer implementation based on Gmail.

class moe.mailer.gmailer.**Gmailer**(*user*, *destination*, *secret='client_secret.json'*, *credentials='credentials.json'*)

Bases: `object`

Implementation of Mailer that leverages Gmail.

Capable of configuring existent gmail accounts for MOE to use. It does so through Gmail's Label and a Filter capabilities.

Parameters

- **`user`** (*`str`*) – Email address of the user.
- **`destination`** (*`str`*) – Email address of the other MOE user.
- **`secret`** (*`str, optional`*) – Defaults to 'client_secret.json'. File containing the OAuth 2.0 client ID of the MOE application.
- **`credentials`** (*`str, optional`*) – Defaults to 'credentials.json'. File containing the OAuth 2.0 Google user authentification.

Raises

- `ValueError` – Invalid user email.
- `ValueError` – Invalid destination email.

`create_message`(*content*, *subject='MOE message'*)

Creates a message object for an email.

It's receiver its the configured receiver of Mailer. It's sender is the configured user of Mailer.

Parameters

- **`content`** (*`str`*) – The subject of the email message.
- **`subject`** (*`str, optional`*) – Defaults to DEFAULT_SUBJECT. The text of the email message.

Returns An object containing a base64url encoded email object.

Return type object

`delete_message`(*message_id*)

Deletes a message from the inbox.

Parameters **`message_id`** (*`str`*) – The id of the message to delete.

Return type `None`

`fetch_all`()
Fetch all the emails in MOE's inbox.

Returns A list with the MOE email dicts in chronological order.

Return type List[Dict]

`fetch_unread`()
Fetch all the emails in MOE's inbox that are unread.

Returns A list of MOE emails.

Return type List[Dict]

`mark_as_read`(*msg*)
Marks the email message as read from the MOE inbox in Gmail

If the message has already been read the function does not do anything.

Parameters **`msg`** (*`Dict`*) – The MOE email to mark as read.

Returns The updated MOE email.

Return type Dict

`read`()
Reads the latest unread message from the MOE inbox in Gmail.

If there is an unread email, the email is marked as seen, but not deleted. If there is no unread email, it returns an empty object.

Returns MOE's email object.

Return type Dict

`write`(*content*)
Sends an email with the content to the configured receiver.

This method ensures Gmailer is an implementation of the Writer interface.

Parameters **`content`** (*`str`*) – The content to send.

Returns The id of the sent message.

Return type str

1.2.4 moe.reader package

Module contents

1.2.5 moe.writer package

moe.writer.echoer module

The echoer module contains an implementation of Writer which prints strings to stdout.

class `moe.writer.echoer.`**`Echoer`**(*stream=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
Bases: `object`

Writer that prints strings to a given stream.

Parameters **`stream`** (*`object, optional`*) – Defaults to sys.stdout. Stream to which to write.

`set_stream`(*stream*)
Define the stream to which to write the strings.

Parameters **`stream`** (*`object`*) – Stream to which to write to.

Returns The Echoer instance

Return type object

`write`(*string*)
Write the string to a given stream

Parameters **`string`** (*`str`*) – string to write.

Return type `None`

1.3 Contribution guidelines

I am completely open to suggestions, criticism, and collaborators. Feel free to open issues and label them accordingly.

Before sending a Pull Request, please make sure that you're assigned the task on a GitHub issue.

- If a relevant issue already exists, discuss on the issue and get it assigned to yourself on GitHub.
- If no relevant issue exists, open a new issue and get it assigned to yourself on GitHub.

1.3.1 Git Workflow

This project follows a simple Git Workflow [1][2]:

- Branch off the latest master and name the branch `<descriptive_name>-<issue_number>`

```
git clone https://github.com/CamiloGarciaLaRotta/MOE.git
cd MOE
# activate your python venv
pip install -r requirements.txt

git checkout -b add-hello-world-support-42
```

- Keep your branch up to date by rebasing on top of master .. code-block:: bash

  git fetch origin git rebase origin/master

- When you are done, push your branch and open a Pull Request in GitHub .. code-block:: bash

  git push -u origin add-hello-world-support-42

- Once the Pull Request is approved, perform an explicit merge .. code-block:: bash

  git checkout master git pull origin master git merge –no-ff add-hello-world-support-42

1.3.2 Pull Request Checklist

- In order to keep a homogenious naming convention, you have to name your implementations of **Reader**, **Writer** and **Mailer** with a name ending in `-er` (:
- You may find useful my configuration file for VSCode and editorconfig.

- It is highly recommended that you use the root Makefile as **git pre-commit hook**, as it will do everything all for you (linting, testing, documentation)

```
   #.git/hooks/pre-commit
   #!/bin/sh

   make validate

then ``chmod +x .git/hooks/pre-commit``
```

- Your code must pass the linting (Pylint, Flake8), as well as remove trailing whitespace of all files.
- If your contribution adds/modifies `MOE`s functionality, please add appropriate tests.
- Please add appropriate documentation to your code in Google Style.
- If your contribution requires extra dependencies, please mention it in the Pull Request and update project's depencencies: `pip freeze > requirements.txt`

1.4 Vulnerabilities

If you find any vulnerabilities, **please** open an Issue and add the label `Vulnerability`. It will be handled as top priority issue.

CHAPTER 2

Indices

- genindex
- modindex

Python Module Index

Index