microparcel Documentation

Release 0.1.1

Vivien Henry

Sep 20, 2020

Contents:

CHAPTER 1

microparcel-python

Serialize and deserialize structured data.

microparcel impementation in Python.

- Free software: MIT license

Provides three differents entities:

A Message is the payload, with methods to access specifics bitfields on the data buffer

The Frame encapsulates the Message between a Start Of Frame and a CheckSum

And the Parser is used to Parse bytes into Message and encodes Messages in Frame

See Documentation: https://microparcel-python.readthedocs.io/en/latest/

1.1 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install microparcel, run this command in your terminal:

```
$ pip install microparcel
```

This is the preferred method to install microparcel, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for microparcel can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/lukh/microparcel
```

Or download the tarball:

```
$ curl  -OL https://github.com/lukh/microparcel/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

The Message is the Payload, transmitted via a serial line (UART, I2C), byte by byte.

The Message object holds a Data buffer (an array of bytes), the size is defined by a class template parameter. And provides methods to access a specific part of the message (a specific bitfield in the message): It uses offset, in bits, and a bitmask

```
import microparcel

# creates a message with a 8 bytes payload
msg = microparcel.Message(size=8)

# set the 5th, 6th, 7th bits of the payload (5,6,7 of the first byte) at the value "2"
msg.set(5, 3, 2)

# set the 13th, 14th, 15th bits of the payload (5,6,7 of the second byte) at the␣
↪value "3"
msg.set(13, 3, 3)

# set the 6th, 7th, 8th, 9th bits of the payload; eg:
# bits 6 and 7 of the first byte, bits 0 and 1 of the second
# at the value "1"
msg.set(6, 4, 1)


# for bisize higher that 8 (one byte), the offset must be aligned on a byte
# bitsize is limited to 16; and the rettype should be change to uint16_t
msg.set(24, 16, 0xFFAF)


# getter works in the same way:
msg.get(5, 3)
msg.get(24, 16)
```

3.1 Frame

A Frame encapsulate the Message between a StartOfFrame (SOF) and a CheckSum.

The SOF is an arbitrary value (in our case, 0xAA), and the CheckSum is the sum of all bytes, including the SOF, truncated to 8bits.

It allows a lighweight and fast data integrity validation.

3.2 Parser

The Parser takes bytes, and builds up a Message from the data stream.

```
import microparcel


serial = serial.Serial(serial_port, serial_baudrate)

# a Parser Class for Message with a Payload of 6.
TParser = microparcel.make_parser_cls(6)
parser = TParser()

# main loop
while not stop:
    ser_in = serial.read()
    if ser_in == "":
        continue

    raw_byte = ord(ser_in)

    msg = microparcel.Message(size=6)

    # parse byte
    status = parser.parse(raw_byte, msg)
    if status == parser.Status.Complete:
        pass # Handles the message here

    if status == parser.Status.Error:
        print("Error in parsing Serial Message: recv byte = {}, current msg = {}".
↪format(raw_byte, msg.data))

serial.close()
```

The Parser also encodes Message into Frames for sending data

```
import microparcel

serial = serial.Serial(serial_port, serial_baudrate)

# a Parser Class for Message with a Payload of 6.
TParser = microparcel.make_parser_cls(6)
parser = TParser()

def sendMsg(msg):
    if serial is None:
        raise FrontendError("Can't send message to the hardware, serial port not␣
↪opened")
```

(continues on next page)

(continued from previous page)

```
    frame = parser.encode(msg)
    buff = bytearray()
    for d in frame.data:
        buff.append(d)
    serial.write(buff)


# creates a message with a 8 bytes payload
msg = microparcel.Message(size=6)

# set the 5th, 6th, 7th bits of the payload (5,6,7 of the first byte) at the value "2"
msg.set(5, 3, 2)
# set the 13th, 14th, 15th bits of the payload (5,6,7 of the second byte) at the␣
↪value "3"
msg.set(13, 3, 3)
# ...

sendMsg(msg)
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/lukh/microparcel/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

microparcel could always use more documentation, whether as part of the official microparcel docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/lukh/microparcel/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *microparcel* for local development.

1. Fork the *microparcel* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/microparcel.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv microparcel
$ cd microparcel/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 microparcel tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/lukh/microparcel/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_microparcel
```

4.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 5

Credits

5.1 Development Lead

- Vivien Henry <vivien.henry@outlook.fr>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.0 (2019-09-24)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search