# Microparcel Tools Documentation

***Release 0.0.1***

**Vivien Henry**

**Sep 25, 2019**

# Contents:

CHAPTER 1

# Microparcel Tools

Code generation tool for microparcel

- Free software: MIT license
- Documentation: https://microparcel-tools.readthedocs.io.

## 1.1 Features

- Generates a Communication protocol from a schema.
- Generates C++ and python code
- Based on microparcel, suitable for embedded systems

## 1.2 Credits

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

CHAPTER 2

# Installation

## 2.1 Stable release

To install Microparcel Tools, run this command in your terminal:

```
$ pip install microparcel_tools
```

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

Install in a virtualenv or as root (not recommended) for access to direct command line tool.

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 Tools can be downloaded from the Github repo.

You can either clone the public repository:

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

Or download the tarball:

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

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

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

# CHAPTER 3

# Usage

To use microparcel_tools from command line:

```
microparcel_tools path/to/schema.json --py path/to/generate/py/dir -cxx path/to/
↪generate/c++/dir
```

or

```
python -m microparcel_tools path/to/schema.json --py path/to/generate/py/dir -cxx␣
↪path/to/generate/c++/dir
```

The schemas is the source file, written in JSON. See the examples in examples. It will generates: * A Message File: defines all the necessary fields * A Router File per endpoint; they provides static "make" methods and pure virtual "process" methods.

basically, it describes nodes and fields.

A node can have children node, it allows to organize messages by subject. And a node without children (called a Leaf) can have Fields.

The endpoints are the termination of the serial line (eg: Master or Slave). A message is send by one or more endpoint (defined by sender) This defines which Endpoint has a "make" method and a "process" method

The microparcel_tool generates, in python or C++, source code to generate microparcel messages, or to process them, in order to send them via a serial line.

This piece of Schema will generate

```
"name":"Farm",
"version":{"major":0, "minor":1},
"endpoints":["Master", "Slave"],
"nodes":{
    "name":"MsgType",
    "children":[
        {
            "name":"Measure",
            "children":[
```

(continues on next page)

(continued from previous page)

```
                {
                    "name":"Windspeed",
                    "senders":["Slave"],
                    "fields":[
                        {"name":"ID", "short_name":"ID", "bitsize":4},
                        {"name":"Unit", "short_name":"Un", "enum_name":"SpeedUnit"},
                        {"name":"Value", "short_name":"Va", "bitsize":12},
                    ]
                },
```

A Master Router

```
class FarmMasterRouter {
    virtual void processWindspeed(uint8_t in_windspeedid, FarmMsg::SpeedUnit in_
↪windspeedunit, uint16_t in_windspeedvalue) = 0;
};
```

A Slave Router

```
class FarmSlaveRouter {
    static FarmMsg makeWindspeed(uint8_t in_windspeedid, FarmMsg::SpeedUnit in_
↪windspeedunit, uint16_t in_windspeedvalue){
        FarmMsg msg = FarmMsg();

        msg.setAddress(in_address);
        msg.setProtocolVersion(in_protocolversion);

        msg.setMsgType(FarmMsg::MsgType_Measure);
        msg.setMeasure(FarmMsg::Measure_Windspeed);

        msg.setWindspeedID(in_windspeedid);
        msg.setWindspeedUnit(in_windspeedunit);
        msg.setWindspeedValue(in_windspeedvalue);

        return msg;
    }
};
```

## 3.1 Sending Message

Creating and sending a message is easy (from Slave side):

```
#include <microparcel/microparcel.h>
#include "FarmMsg.h"
#include "FarmSlaveRouter.h"

class FarmSlaveRouterImplementation: public FarmSlaveRouter {
    // need to implement all pure virtual process methods for others messages.
    // virtual void process...{
    //}
};

// prototype to send data
void send(uint8_t *data, uint8_t datasize);
```

(continues on next page)

(continued from previous page)

```
int main(){
    // a Parser for FarmMessage.
    using TParser = microparcel::Parser<FarmMsg::kSize>;

    FarmMsg msg = FarmSlaveRouterImplementation::makeWindspeed(5, FarmMsg::SpeedUnit_
↪Knot, 100);


    // builds the frame, with SOF and checksum
    TParser::Frame_T frame = TParser.encode(msg);

    // send over physical layer of choice
    send((uint8_t*)&inFrame, TFrame::FrameSize);

}
```

## 3.2 Receiving message

The master side and the slave implements the virtual process methods; where their parameters are the relevant one (windspeed and ID of the measure)

A Router has a "process" methods:

```
void process(FarmMsg &in_msg){
    // big automatic generated switch-case
```

Calling "process" with a VALID message received from microparcel 's Parser.parse will call the right virtual processes method.

```
#include <microparcel/microparcel.h>
#include "FarmMsg.h"
#include "FarmMasterRouter.h"

class FarmMasterRouterImplementation: public FarmMasterRouter {
    virtual void processWindspeed(uint8_t in_windspeedid, FarmMsg::SpeedUnit in_
↪windspeedunit, uint16_t in_windspeedvalue){
        // DO SOMETHING
        notifyViaWifi(in_windspeedid, in_windspeedvalue);
    }
};

// a way to get data from a Serial Line (UART ?)
uint8_t getByteFromDataLine();
bool isDataLineEmpty();

int main(){
    // a Parser for FarmMessage.
    using TParser = microparcel::Parser<FarmMsg::kSize>;

    FarmMasterRouterImplementation fmri;
    FarmMsg msg;

    TParser parser;
```

(continues on next page)

(continued from previous page)

```
    TParser::Status status;

    // main loop of embedded application
    while(true){
        // continue till the fifo is empty
        while(!isDataLineEmpty()){
            uint8_t byte = getByteFromDataLine();
            status = parser.parse(byte, &msg);
            switch(status){
                // not complete and error could be treated differently...
                // error means mainly that the checksum is not valid; transmission
↪failed.
                case TParser::eNotComplete:
                case TParser::eError:
                    break;

                case TParser::eComplete:
                    // msg is complete, handle it
                    fmri.process(msg);
                    break;
            }
        }

    }
}
```

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_tools/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 Tools could always use more documentation, whether as part of the official Microparcel Tools 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_tools/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_tools* for local development.

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

```
$ git clone git@github.com:your_name_here/microparcel_tools.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_tools
$ cd microparcel_tools/
$ 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_tools 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_tools/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_tools
```

## 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 0.0.1 (2019-02-28)

- First release on PyPI.

CHAPTER 7

# Indices and tables

- genindex
- modindex
- search