Smart Farm Hub Documentation

Release 0.1.7

Joseph Malemela

Nov 11, 2020

Contents:

CHAPTER 1

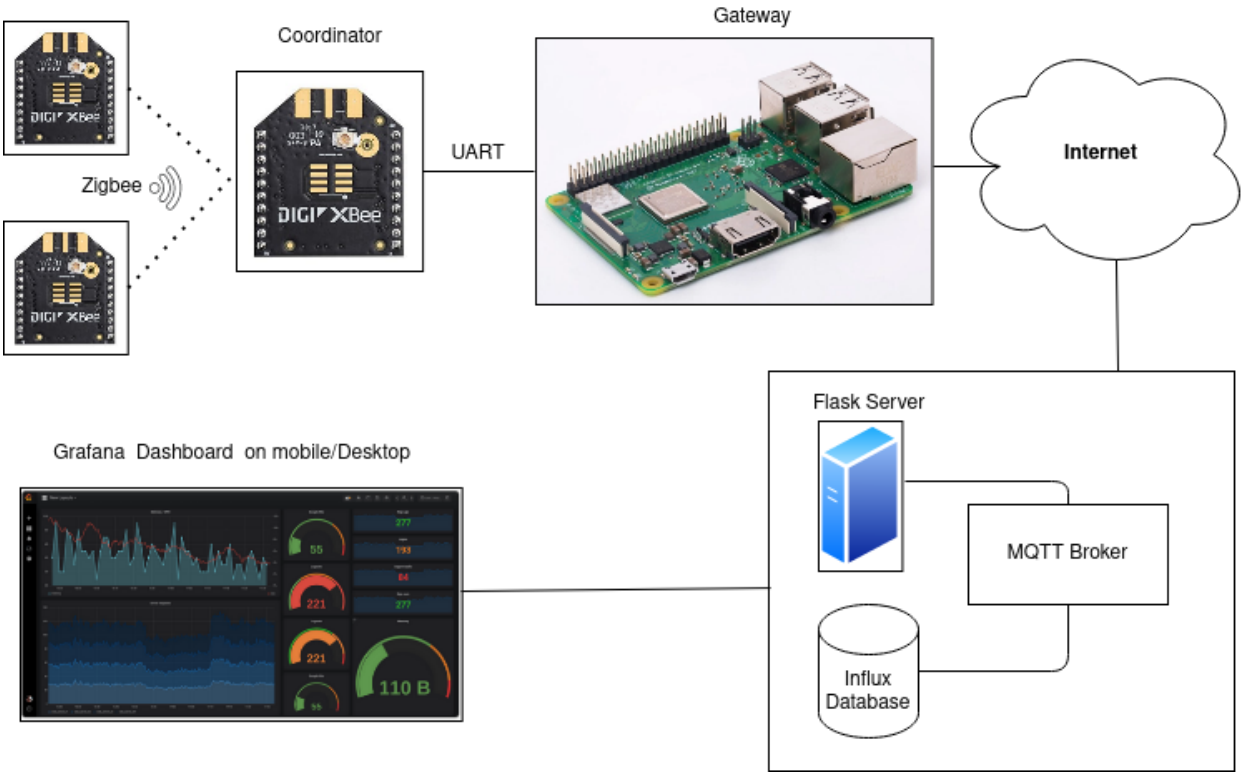
Smart Farm Hub

Gateway API for a Zigbee Wireless Network and Cloud Based Docker image for a smart farm hub written in Python

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

1.1 Features

- Add a range of Zigbee Devices or Sensors or Actuators to a RPI machine interface
- Add sensors or actuators to Xbee3 modules
- Publish sensor information to MQTT broker on the cloud
- Connect to the Web application to monitor and control the wireless sensor network remotely
- Automatically Detect new devices on the network and sensors
- Send commands from grafana to wireless sensor network



1.2 Credits

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

The server Docker container instructions were adopted from https://github.com/iothon/docker-compose-mqtt-influxdb-grafana

CHAPTER 2

Installation

2.1 Stable release

To install Smart Farm Hub, run this command in your terminal:

```
$ pip install LowCostSmartFarmHub
```

This is the preferred method to install Smart Farm Hub, 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 Smart Farm Hub can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/itumeleng96/LowCostSmartFarmHub
```

Or download the tarball:

```
$ curl -OJL https://github.com/itumeleng96/LowCostSmartFarmHub/tarball/master
```

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

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

CHAPTER 3

Usage

To use Smart Farm Hub in a project:

```
import LowCostSmartFarmHub
```

To use the Gateway,Sensor,Actuator or Node Device class

```
from LowCostSmartFarmHub import Sensor
from LowCostSmartFarmHub import Gateway
from LowCostSmartFarmHub import Actuator
from LowCostSmartFarmHub import NodeDevice
```

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/itumeleng96/LowCostSmartFarmHub/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

Smart Farm Hub could always use more documentation, whether as part of the official Smart Farm Hub 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/itumeleng96/LowCostSmartFarmHub/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 *LowCostSmartFarmHub* for local development.

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

```
$ git clone git@github.com:your_name_here/LowCostSmartFarmHub.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 LowCostSmartFarmHub
$ cd LowCostSmartFarmHub/
$ 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 LowCostSmartFarmHub tests
$ python setup.py test or pytest
$ 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 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/itumeleng96/LowCostSmartFarmHub/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_LowCostSmartFarmHub
```

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:

```
$ bump2version 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

- Joseph Malemela <itukzz96@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.0 (2020-09-29)

- First release on PyPI.

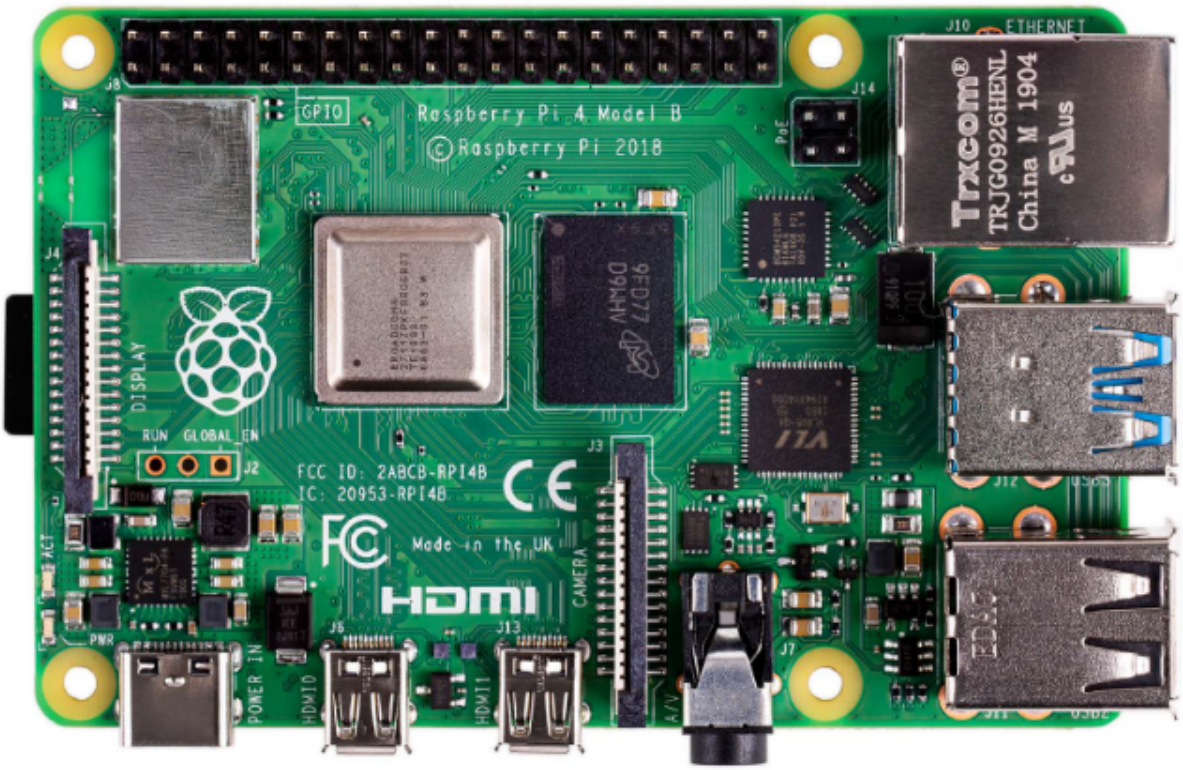
CHAPTER 7

Low Cost Smart Farm Hub terminology

This section covers basic LowCostSmartFarmHub concepts and terminology.The various modules and classes used in the API will make references to these terms.

7.1 Gateway device

A gateway is a device that acts as the middle-man between the internet and the wireless-sensor network made up of radio modules.The gateway has to be connected to the internet through WIFI or ethernet to transmit local data to the internet.



7.2 Actuator

Any device that is capable of recieving commands and converting them to an electrical signal for lighting or any other mechanical movement.This device is usually connected to the Node Devices(RF modules) or to the gateway directly.

7.3 Sensor

Any device capable of measuring and gathering environmental data in a farm and communicating with the gateway or node-device. This device can be I2C,analog or digital sensor.

7.4 Node-device

Any device that can act as node in the smart-farm wireless network. This can be any Zigbee RF module or a zigbee device in the network.

7.5 ZigBee RF modules

A radio frequency (RF) module is a small electronic circuit used to transmit and receive radio signals on different frequencies.The RF modules used in this API are from Digi-XBee and they are the XBee3 Through Hole RF Modules.

7.6 Radio module operating modes from Digi-XBee

The operating mode of an XBee radio module establishes the way a user, or any microcontroller attached to the XBee, communicates with the module through the Universal Asynchronous Receiver/Transmitter (UART) or serial interface.

Depending on the firmware and its configuration, the radio modules can work in three different operating modes:

- Application Transparent (AT) operating mode
- API operating mode
- API escaped operating mode

In some cases, the operating mode of a radio module is established by the firmware version and the firmware's AP setting. The module's firmware version determines whether the operating mode is AT or API. The firmware's AP setting determines if the API mode is escaped (**AP** = 2) or not (**AP** = 1). In other cases, the operating mode is only determined by the AP setting, which allows you to configure the mode to be AT (**AP** = 0), API (**AP** = 1) or API escaped (**AP** = 2).

CHAPTER 8

Configure the Raspberry PI Device

The LowCostSmartFarmHub Python Library provides the ability to communicate with XBee devices connected to a low-power gateway device that publishes data to a MQTT Broker.

> **Warning:** Communication features described in this topic and sub-topics are only applicable for machines like RPI3B+ with UART interfaces connecting to local XBee devices.

8.1 Create A gateway and add devices to it

The RPI gateway can connect to a local Xbee Device on the UART interface specified by the user.

** Instantiate the Gateway and connect to local XBee device on UART **

```
[...]

# Instantiate a Gateway device object

gateway = Gateway("RPI 3B+","Farm location 1")


# connect to Local XBee device on UART interface

gateway.connect_uart_stream("COM1",9600,True)


[...]
```

The previous methods may fail for the following reasons:

- There is no XBee device on the serial UART interface a ‘‘Connection Exception ‘‘.
- Other errors caught as `XBeeException`:

- The operating mode of the device is not `API` or `ESCAPED_API_MODE`, throwing an `InvalidOperatingModeException`.

8.2 Get and Set Gateway Paramaters

The Gateway Class has various methods that allow the user to set and get certain attributes of the gateway.

Class Method	Description
read_device_info()	returns information about gateway
add_actuator(actuator)	Adds actuator to the gateway
add_sensor(Sensor)	Adds sensor to the gateway

CHAPTER 9

Work with the Raspberry PI Gateway

The Gateway Class provides methods to connect to local XBee devices and discover remote XBee devices and add them to the gateway using the digi-xbee API https://github.com/digidotcom/xbee-python

Warning: Ensure that the XBee Coordinator device is connected to the Gateway before executing the discover zigbee devices method

9.1 Discover remote Zigbee devices on the same network

Using the coordinator in API mode, the remote devices can be found using this method

Instantiate Gateway and Discover Local Devices

```
[...]

# Instantiate a Gateway device object
gateway = Gateway()

# connect to Local XBee device on UART interface
gateway.connect_uart_stream("COM1",9600,True)

devices=gateway.discover_zigbee_devices()

#devices =[remote xbee1,remote xbee2, e.t.c]

[...]
```

9.2 Detect all sensors and actuators connected directly to Gateway

This method detects all the devices connected directly to gateway and adds them to the gateway

Instantiate Gateway and detect Devices on gateway

```
[...]

# Instantiate a Gateway device object
gateway = Gateway()

#Detect Devices on Gateway
gateway.detect_devices(add_devices=True)

[...]
```

CHAPTER 10

Using the MQTT Client on the gateway to publish and subscribe to Broker

The gateway publishes data and recieves commands from the MQTT broker using the methods described below.

Warning: Ensure that the Cloud Applications are runnning or that the provided broker address is a valid MQTT broker address ready to recieve data from gateway

10.1 Connect to MQTT broker

With the gateway connected to the internet, the gateway can communicate with the server using the methods below.

Connect Gateway to MQTT broker

```
[...]

# Instantiate a Gateway device object
gateway = Gateway()

gateway.mqtt_connect(client_id="xvsvs",broker='www.mosquitto-broker.com',port=1883)

#returns a MQTT client

[...]
```

10.2 Publish all devices information to MQTT broker

The gateway can publish all the information from the devices on the sensor network

Publish All Sensor Data to MQTT broker

```
[...]

# Instantiate a Gateway device object
gateway = Gateway()

client=gateway.mqtt_connect(client_id="xvsvs",broker='www.mosquitto-broker.com',
↪port=1883)

client.publish()

[...]
```

CHAPTER 11

Examples

CHAPTER 12

Indices and tables

- genindex
- modindex
- search