SmartAgro Documentation

Kudzai Chris Kateera

Oct 29, 2020

Contents:

CHAPTER 1

SmartAgro

Smart, Iot-enabled greenhouse monitoring and control API

- Free software: GNU General Public License v3
- Documentation: https://smartagro.readthedocs.io.

1.1 Features

- TODO

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 SmartAgro, run these command in your terminal:

```
$ sudo apt-install libgpiod2
$ pip3 install smartagro
```

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

If you don't have pip3 installed, this Python installation guide can guide you through the process.

2.2 From sources

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

You can either clone the public repository:

```
$ git clone git://github.com/chris-kck/smartagro
```

Or download the tarball:

```
$ curl -OJL https://github.com/chris-kck/smartagro/tarball/master
```

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

```
$ python3 setup.py install
```

CHAPTER 3

Usage

To use SmartAgro in a project:

```
#Can be imported and implemented in different ways:
#import smatagro then use smartagro.smartagro.func1() or smartagro.utils.func2()
#from smartagro.smartagro import *
# from smartagro import *, smart.SmartAgro(), smart.func1(), utils.func2()
```

Prefered and easy way to use SmartAgro in a project:

```
from smartagro import *
import time

utils.find_broker() #search for a broker within your network
utils.discover_i2c() #discover devices connected to your Pi

# instatiate SmartAgro Object and conect to a broker. Optionally specify details
obj = smart.SmartAgro()

# Print different sensor' data values
print(f"Moisture 0 output: {obj.read_sensor(0)} %")
print(f"Light 2 output: {obj.read_sensor(2)} %")
print(f"dht temperature and humidity: {obj.read_dht()}")

# Activate an actuator directly with a pause then deactivate
obj.activate_actuator(15,1)
time.sleep(3)
obj.activate_actuator(15,0)

# Print out all 4 sensors' current values and publish to broker
print(obj.read_all())

#cleanup and exit the program
utils.cleanup()
exit(0)
```

CHAPTER 4

Package Modules

class smartagro.smart.**SmartAgro**

Implemented After searching for a broker Instantiates an object which has sensors added to it then configures a broker. Sensors are attached added with corresponding topics Sensor Data is published and Actuator can be activated

activate_actuator(*gpio_pin*, *state*)

A function to activate or deactivate an actuator.

Parameters

- **gpio_pin** (*int*) – GPIO pic of connected actuator.
- **state** (*bool*) – State whether it is on or Off

config_broker(*broker='test.mosquitto.org'*, *qos=0*, *port=1883*, *stream_schema='json'*)

Function to configure a new broker to be published to.

Parameters

- **broker** – The url or ip address of the broker.
- **qos** – quality of service determining how many times message is sent. 0,1,2
- **port** – broker port in use. default 1883, ssl 8883
- **stream_schema** – the data stream schema used. Default is json

Returns mqtt client object

get_dht()

A function to get readings from the single wire DHT11 device.

Returns Temperature and Humidity Readings

static on_connect(*client*, *userdata*, *flags*, *rc*)

The callback for when a connection is established with the server.

Parameters

- **client** – Mqtt Client

- **userdata** – Authentication data
- **flags** – Connection indicators
- **rc** – status code of connection

static on_message(*client*, *userdata*, *msg*)
The callback for when a PUBLISH message is received from the server.

Parameters

- **client** – MQtt client
- **userdata** – data used for authenticated connections
- **msg** – received message topic and payload in bytes

read_all()
A function to read all the values at once

Returns A list of current moisture, light, temperature, humidity values

read_sensor(*channel*)
Reads sensor, publishes topic to broker, adds to active sensors

Parameters **channel** – ADC channel to be read.

remove_device(*device*)
Function to remove device from published topics

Parameters **device** (*str*) – Device Topic

Utilities Module. With several functions that are repeatedly used

smartagro.utils.**cleanup**()
GPIO.cleanup() and exit(0) for a graceful exit.

smartagro.utils.**discover_i2c**()
Scans address space and ports to discover connected I2C or SPI devices uses os i2cdetect for the 1 I2C port and also scans two SPI ports

smartagro.utils.**find_broker**()
Scan for online MQTT brokers then scan within network by checking online hosts then scanning for open MQTT ports

Returns No return

smartagro.utils.**gpio_init**()
Function to initialize the GPIO pins, numbering system used and communication protocols. GPIO.BCM IS THE DEFAULT

smartagro.utils.**read_analogue**(*channel*, *spi_device=0*, *baud=1350000*)
Reads an analogue signal from the connected SPI ADC device and returns channel reading.

Parameters

- **channel** (*int*) – ADC channel where sensor is connected.
- **spi_device** (*int*) – Either 0 or 1 as there are only 2 spi ports
- **baud** (*int*) – the bit rate, measured in bit/s clock rate used for device

Returns Raw 1024 bit ADC output data.

smartagro.utils.**scan_network**()
Ger the IP address other than the loopback IP that the device has been allocated by DHCP Scan subnet /24 of IP address to check for LAN brokers' availability.

Returns list of online devices responding to ICMP echo request using ping.

smartagro.utils.**switch_actuator**(*gpio_pin*, *state*)
Function to switch actuator ON or OFF

Parameters

- **gpio_pin** (*int*) – The pin the fan relay (motor in demo) is connected to.
- **state** (*boolean*) – Boolean indicating whether fan is on or off.

CHAPTER 5

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:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/chris-kck/smartagro/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.

5.1.2 Fix Bugs

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

5.1.3 Implement Features

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

5.1.4 Write Documentation

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

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/chris-kck/smartagro/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 :)

5.2 Get Started!

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

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

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

5.3 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/chris-kck/smartagro/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_smartagro
```

5.5 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 6

Credits

6.1 Development Lead

- Kudzai Chris Kateera <kckateera@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.1.0 (2020-09-03)

- First pre-release on PyPI.

7.2 0.1.1 (2020-10-12)

- Second pre-release on PyPI. Added modules

7.3 0.2.0 (2020-10-26)

- First production release on PyPI. Fully Functional

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index