
EES-EATON-2019 Documentation

Release 0.01

Abdullah Ahmed

Mar 10, 2020

CONTENTS

| | | |
|----------|-----------------------------|----------|
| 1 | v.0.01 documentation | 1 |
| 2 | Features | 3 |
| 3 | Models | 5 |
| 3.1 | Arduino | 5 |

V.0.01 DOCUMENTATION

A communication interface between an Arduino and a host machine over a serial connection.

FEATURES

- Establishes communication over serial interface
- Able to send and receive data encoded in ASCII format
- Arduino can handle up to 6 LDRs and can return their reading
- Each sensor is identified using a unique identifier to send and receive data
- Program contains database of sensors and can handle incorrect UIDs

Usage Import the file containing the models `python from models import Arduino, Sensor`

Create an instance of a serial connection to the Arduino

```
arduino = Arduino()
```

This creates an object with the following attributes:

| attr | value |
|----------|---------------------|
| port | /dev/ttyACM0 |
| baudrate | 9600 |
| bytesize | 8 |
| timeout | 2 |
| stopbits | serial.STOPBITS_ONE |

If any of the attributes need to be changed change the instable variable. Eg. to change the port:

```
arduino.port = "/dev/ttyACM0"
```

Create and add to the array storing all of the sensors

```
ldrs = []
```

To initialise a sensor:

`main.py`

```
ldr = Sensor(1, arduino)
```

1 is the unique id given to this sensor. This will be used to identify the sensor. This **must** be a unique id.

`arduino` is the `Arduino()` object created earlier. This tells the program where to communicate about the sensor.

Say for example the sensor is attached to pin A5.

`arduino_code.ino`

```
int sensors[2] = {A4,A5}; # add A5 to the array

...

switch (serial_string.toInt())
{
    case 1:
        Serial.print(analogRead(sensors[0]));
        break;
        // Add your own case statement
    case 2:
        Serial.print(analogRead(sensors[1]));
        break;

...

```

To get a reading from a sensor

```
print(ldr.getReading())
```

To read what is currently in the serial buffer simply do

```
arudino.readBuffer()
```

Close the connection:

```
arduino.closeConnection()
```


MODELS

3.1 Arduino

```
class Arduino:
    """
        Class to model a serial connection to an arduino.
    """

    def __init__(self):
        #Initialise Connection
        ...
```

| attr | value | description |
|---------------------------|--|--|
| self. serialConnection | serial.Serial(port = "/dev/ttyACM0", baudrate=9600, bytesize=8, timeout=2, stopbits=serial.STOPBITS_ONE) | Parameters to initialise the serial connection |

For more information on the attributes of the serial Module see [Docs](#)

3.1.1 Methods

self.readBuffer()

```
def readBuffer(self):
    """
        Reads all the content stored in the serial buffer and returns content as
        ↪ a string
    """
```

| attr | value | description |
|------|-------|-------------|
| None | . | . |

self.sendData() ``python def sendData(self, data): ``""" Sends given data over the serial connection encoded in byte form e