Guide to OsensaPy Library

Release 0.1.7

This Python library is meant to allow the use of OSENSA transmitters with Python. It allows reading of all data (temperature, channel status, etc.) from each channel. It also includes the ability to read batches of temperature data from each channel in one command. The library works with Python 3, and interfaces with OSENSA transmitters running at a baud rate of 9600. This document outlines the available functions and their usage, and assumes that both Python 3 and Python 2 are available on the system.

To install the library, open a command line and run

```
$ sudo pip3 install osensapy
```

This will install the library and its dependencies in Python 3.

Invoke the Python command line with

```
$ python3
```

Import osensapy into the Python workspace with

```
> from osensapy import osensapy
```

This can also be done directly into your project.

Get a list of **available serial ports** that the transmitter might be attached to with

```
> osensapy.serial_get_portlist()
```

This will return a list of all ports, including the transmitter's port. It may take some trial and error to get the correct port if you do not already know.

```
>>> osensapy.serial_get_portlist()
/dev/ttyUSB0
/dev/ttyS0
/dev/ttyAMA0
/dev/serial0
/dev/serial1
['/dev/ttyUSB0', '/dev/ttyS0', '/dev/ttyAMA0', '/dev/serial0', '/dev/serial1']
```

Create a “transmitter” object with

```
> transmitter = osensapy.Transmitter(“/dev/ttyUSB0”,  247)
```

Here, “/dev/ttyUSB0” is an example of what the serial port looks like in a Unix system. In Windows, you may see port names like “COM3” instead. “247” is the Device ID here.

```
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from osensapy import osensapy
>>> osensapy.serial_get_portlist()
/dev/ttyUSB0
/dev/ttyS0
/dev/ttyAMA0
/dev/serial0
/dev/serial1
['/dev/ttyUSB0', '/dev/ttyS0', '/dev/ttyAMA0', '/dev/serial0', '/dev/serial1']
>>> transmitter = osensapy.Transmitter("/dev/ttyUSB0", 247)
```

Read the transmitter’s **serial number** with

```
> transmitter.read_serial_number()
```

```
>>> transmitter.read_serial_number()
17080016
```

Read the transmitter’s **device ID** with

```
> transmitter.read_device_id()
```

```
>>> transmitter.read_device_id()
247
```

Read a channel’s **temperature reading** (in °C) with

```
> transmitter.read_channel_temp(‘C’)
```

Here ‘C’ is the channel, written in quotations as a character. The default channel is ‘A’

```
>>> transmitter.read_channel_temp('C')
21.883087158203125
```

Read a truncated version of **temperature** reading (in °C) with

```
> transmitter.fast_single(‘C’)
```

‘C’ is the channel, written in quotations as a character. The default channel is ‘A’.

```
>>> transmitter.fast_single('C')
('C', 21.9, 0)
```

Read a channel's **LED current** with

```
> transmitter.read_channel_led_curr('C')
```

Here 'C' is the channel, written in quotations as a character. The default channel is 'A'. The LED current signifies how brightly the LED is flashed, as a number out of 4000.

```
>>> transmitter.read_channel_led_curr('C')
31
```

Read a channel's **status** with

```
> transmitter.read_channel_status('C')
```

Here 'C' is the channel, written in quotations as a character. The default channel is 'A'. A status of '1' signifies that the channel is reading correctly.

```
>>> transmitter.read_channel_status('C')
1
```

Read the transmitter's **device temperature** with

```
> transmitter.read_pcb_temp()
```

This returns the temperature of the board in °C.

```
>>> transmitter.read_pcb_temp()
28.80000000000001
```

Read the channel's **decay time** with

```
> transmitter.read_channel_decay('C')
```

'C' is the channel, written in quotations as a character. The default channel is 'A'. This returns the decay time measured by the channel in µs.

```
>>> transmitter.read_channel_decay('C')
1484.6944580078125
```

Read the channel's **light level** with

```
> transmitter.read_channel_light_level('C')
```

```
>>> transmitter.read_channel_light_level('C')
3006
```

'C' is the channel, written in quotations as a character. The default channel is 'A'.

Read **all the data** from a channel with

```
> transmitter.read_channel_data('C')
```

'C' is the channel, written in quotations as a character. The default channel is 'A'. This will return an array of 5 measurements. The order of data in the array is

`[temperature, decay time, light level, LED current, status].`

```
>>> transmitter.read_channel_data('C')
[21.9530029296875, 1485.2435302734375, 3013, 31, 1]
```

Read a **batch of temperatures** from multiple channels with

```
> transmitter.fast_batch(3)
```

"3" is the number of channels of data to read, and always starts with channel 'A'. This function will return data in an array for each channel. Each array will be ordered as

`[channel, temperature, status].`

Note that a status of '0' with truncated reads signifies that the channel is functioning correctly. In this case, Channel B has a status of 7; it had no probe.

```
>>> transmitter.fast_batch(3)
[('A', 23.2, 0), ('B', 327.6, 7), ('C', 21.9, 0)]
```

Finally, **close** the transmitter object before exiting with

```
> transmitter.close()
```