pysds011

Release 0.0.4

Michele Pagot

Feb 16, 2021

DOCUMENT MAP:

CHAPTER

ONE

OVERVIEW

Simple python driver for SDS011 PM sensor from Nova.

- Free software: MIT license

CHAPTER

TWO

INSTALLATION

```
pip install pysds011
```

You can also install the in-development version with:

```
pip install https://github.com/michelepagot/pysds011/archive/develop.zip
```

CHAPTER

THREE

USAGE

3.1 Use the package in a python script

To use pysds011 in a project, start by importing this package:

```
import pysds011
```

Provide an UART channel, user is in charge to also `open/close` it:

```
import serial

ser = serial.Serial('/dev/ttyUSB0', 9600)
ser.open()
```

Provide a logger:

```
import logging

log = logging.getLogger(__name__)
```

Now create a driver instance, injecting serial and logging:

```
sd = driver.SDS011(ser, log)
```

Usually the first step is to wake up the sensor:

```
sd.cmd_set_sleep(0)
```

Then start to interact with it:

```
sd.cmd_set_mode(sd.MODE_QUERY)
fw_ver = sd.cmd_firmware_ver()
dust_data = sd.cmd_query_data()
```

3.2 Use the command line tool

This package is provided with a command line tool to be able to immidiately start playing with your sensor Command line is named `pysds011`

First stop should be the embedded help. Here just an outdated version of it:

```
pysds011 --help
Usage: pysds011 [OPTIONS] COMMAND [ARGS]...

  pysds011 cli app entry point

Options:
  --port TEXT          UART port to communicate with dust sensor.
  --id TEXT            ID of sensor to use. If not provided, the driver will
                       internally use FFFF that targets all.

  -v, --verbosity LVL  Either CRITICAL, ERROR, WARNING, INFO or DEBUG
  --help               Show this message and exit.

Commands:
  dust        Get dust value
  fw-version  Get SDS011 FW version
  help        Get specific help of a command
  id          Get and set the sensor address
  mode        Get and Set acquisition MODE [0,1] 1: QUERY modeSensor...
  sleep       Get and Set sleep MODE 1:sleep 0:wakeup Just 'sleep' without
              a...
```

And each command has its own `help`:

```
pysds011.exe help dust

Usage: pysds011 help [OPTIONS]

Get dust value

Options:
  --warmup INTEGER  Time in sec to warm up the sensor
  --format TEXT     result format (PRETTY|JSON|PM2.5|PM10)
  --help            Show this message and exit.
```

Nova SDS011 sensor is connected to your machine through UART, so to read the actual `dust` value, you need to provide a **port** value:

```
pysds011.exe --port COM4 dust
PM 2.5: 25.9 g/m^3  PM 10: 62.4 g/m^3 CRC=OK
```

> **Warning:** `dust` command changes both `mode` and `sleep`. In particular it leave the sensor sleeping

Dust value can be presented in **multiple format**:

- PRETTY (default)
- JSON:

```
pysds011.exe --port COM4 dust --format JSON
{'pm25': 15.6, 'pm10': 21.8, 'pretty': 'PM 2.5: 15.6 g/m^3  PM 10: 21.8 g/m^3'}
```

- Single PM:

```
pysds011.exe --port COM4 dust --format PM2.5
26.0

pysds011.exe --port COM4 dust --format PM10
99.0
```

Read the dust sensor FW version:

```
pysds011.exe --port COM4 fw-version

FW version Y: 18, M: 11, D: 16, ID: 0xe748
```

Set the sensor in `sleep` more:

```
pysds011.exe --port COM4 sleep 1
```

Take care that in sleep mode the only accepted command is the one to **wakeup**:

```
pysds011.exe --port COM4 sleep 0
```

`mode` command is about the sensor acquisition mode * 0report active mode * 1report query mode

Both the `sleep` and `mode` commands, asserted without and value, read the actual sensor configuration:

```
pysds011.exe --port COM4 mode
1
```

SDS011 sensors has an addressing functionality but most you do not need to care about it at all. Command `id` is to manage sensor address. Use `id` without any parameter to get the address of the connected sensor:

```
pysds011 --port COM9 id
0x48 0xe7
```

Now you can use it to address a particular command to a particular sensor:

```
pysds011.exe --id 48e7 --port COM4 fw-version

FW version Y: 18, M: 11, D: 16, ID: 48e7
```

To change your sensor address you must specify both current and new `id`:

```
pysds011 --id 48e7 --port COM9 id 1111
```

Now your sensor has a new address:

```
pysds011 --port COM9 id
0x11 0x11
```

Something unexpected is going on? Would you like to figure out what is wrong from your own? Give a try to the DEBUG logging:

```
pysds011 --port COM9 -v DEBUG dust
debug: Process subcommands
debug: BEGIN
debug: is:b'\xff\xff'
debug: driver mode:1
debug: data:[1, 1] dest:b'\xff\xff'
debug: Resized data:[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
debug: > aab406010100000000000000000000ffff06ab
debug: <first byte:b'\xaa':<class 'bytes'>:1
debug: < c50601010048e636ab
debug: mode:1
debug: data:[1, 1] dest:b'\xff\xff'
debug: Resized data:[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
debug: > aab402010100000000000000000000ffff02ab
debug: <first byte:b'\xaa':<class 'bytes'>:1
debug: < c50201010048e632ab
debug: data:[] dest:b'\xff\xff'
debug: Resized data:[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
debug: > aab404000000000000000000000000ffff02ab
debug: <first byte:b'\xaa':<class 'bytes'>:1
debug: < c03c00a50048e60fab
debug: b'\xaa\xc0<\x00\xa5\x00H\xe6\x0f\xab'
PM 2.5: 6.0 g/m^3  PM 10: 16.5 g/m^3
debug: Dust finally
debug: is:b'\xff\xff'
debug: driver mode:0
debug: data:[1, 0] dest:b'\xff\xff'
debug: Resized data:[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
debug: > aab406010000000000000000000000ffff05ab
debug: <first byte:b'\xaa':<class 'bytes'>:1
debug: < c50601000048e635ab
debug: END exit_val:0
debug: process_result res:0
```

CHAPTER

FOUR

API DOCUMENTATION

pysds011	
pysds011.driver	Module that implements low level communication with Nove SDS011 sensor.
pysds011.cli	Module that contains the command line app.

4.1 Driver API

Low level layer that is in charge to manage communication with the sensor. This is the most reusable part of this package

Module that implements low level communication with Nove SDS011 sensor.

class pysds011.driver.**SDS011**(*ser*, *log*)

Main driver class

cmd_firmware_ver(*id=b'\xff\xff'*)

Get FW version

Returns version description dictionary or None if error with fields: 'year', 'month', 'day', 'id', 'pretty'

Return type dict

cmd_get_mode(*id=b'\xff\xff'*)

Get active reporting mode

Parameters **id**(*2 bytes, optional*) – sensor id to request mode, defaults to b'ÿÿ' that is 'all'

Returns mode if it is ok, None if error

Return type int

cmd_get_sleep(*id=b'\xff\xff'*)

Get active sleep mode

Take care that sensor will not response to it if it is sleeping. So mainly this API will return 0 or None. None could means: - I'm sleeping, so I cannot reply - I'm not sleeping but something went wrong as responding

Parameters **id**(*2 bytes, optional*) – sensor id to request sleep status, defaults to b'ÿÿ' that is 'all'

Returns True if it is sleeping, False if wakeup, None in case of communication error

Return type bool

cmd_get_working_period(*id=b'\xff\xff'*)
Get current working period

Returns working period in minutes: work 30 seconds and sleep n*60-30 seconds

Return type int

cmd_query_data(*id=b'\xff\xff'*)
Read dust values from the sensor

Parameters **id** (*2 bytes, optional*) – Sensor ID, defaults to b'ÿÿ'

Returns dust data as dictionary

Return type dict

cmd_set_id(*id*, *new_id*)
Set a device ID to a specific sensor

Parameters

- **id** (*2 bytes*) – ID of sensor that need a new ID (FF FF is in theory allowed too, but be carefull)
- **new_id** (*2 bytes*) – new ID to be assigned

Returns operation result

Return type bool

cmd_set_mode(*mode=1*, *id=b'\xff\xff'*)
Set data reporting mode. The setting is still effective after power off

Parameters

- **mode** (*int, optional*) – 0report active mode 1report query mode, defaults to 1
- **id** (*2 bytes, optional*) – sensor id to request mode, defaults to b'ÿÿ' that is 'all'

Returns True is set is ok

Return type bool

cmd_set_sleep(*sleep=1*, *id=b'\xff\xff'*)
Set sleep mode

Parameters

- **sleep** (*int, optional*) – 1:enable sleep mode, 0:wakeup, defaults to 1
- **id** (*2 bytes, optional*) – sensor id to request mode, defaults to b'ÿÿ' that is 'all'

Returns True is set is ok

Return type bool

cmd_set_working_period(*period=0*, *id=b'\xff\xff'*)

Set working period The setting is still effective after power off, factory default is continuous measurement. The sensor works periodically and reports the latest data.

Parameters **period** (*int*) – 0continuous(default), 1-30 minute work 30 seconds and sleep n*60-30 seconds

Returns result

Return type bool

4.2 CLI app API

Command line interface documentation

4.2.1 pysds011

pysds011 cli app entry point

```
pysds011 [OPTIONS] COMMAND [ARGS]...
```

Options

--port <port>
UART port to communicate with dust sensor.

--id <id>
ID of sensor to use. If not provided, the driver will internally use FFFF that targets all.

-v, --verbosity <LVL>
Either CRITICAL, ERROR, WARNING, INFO or DEBUG

dust

Get dust value

```
pysds011 dust [OPTIONS]
```

Options

--warmup <warmup>
Time in sec to warm up the sensor

--format <format>
result format (PRETTY|JSON|PM2.5|PM10)

fw-version

Get SDS011 FW version

```
pysds011 fw-version [OPTIONS]
```

Options

--format <format>
result format (PRETTY|JSON|PM2.5|PM10)

help

Get specific help of a command

```
pysds011 help [OPTIONS] SUBCOMMAND
```

Arguments

SUBCOMMAND
Required argument

id

Get and set the sensor address

```
pysds011 id [OPTIONS] [ID]
```

Arguments

ID
Optional argument

mode

Get and Set acquisition MODE [0,1] 1: QUERY modeSensor received query data command to report a measurement data. 0: ACTIVE modeSensor automatically reports a measurement data in a work period.

```
pysds011 mode [OPTIONS] [[0|1]]
```

Arguments

MODE
Optional argument

sleep

Get and Set sleep MODE 1:sleep 0:wakeup Just 'sleep' without a number result in querying the actual value applied in the sensor

```
pysds011 sleep [OPTIONS] [[0|1]]
```

Arguments

MODE
Optional argument

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When reporting a bug please include:

- All known details about used dust sensor (vendor, model, revision, sn when available)
- Environment informations: Operating system name and version, python version where you are using the module or the cli.
- Detailed steps to reproduce the bug indicating your expected behaviour

5.2 Documentation improvements

pysds011 could always use more documentation, whether as part of the official pysds011 docs or in docstrings.

In order to build the documentation locally

1. Create once a developement environment (here a Windows procedure, but linux one is almost the same):

```
virtualenv venv_doc
venv_doc\Script\activate.bat
pip install -r docs\requirements.txt
```

2. Build the documentation:

```
cd docs
make clean
make html
start _build\html\index.html
```

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/michelepagot/pysds011/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 code contributions are welcome :)

5.4 Development

5.4.1 GIT machinery

This repo uses *main* in place of *master* as primary branch name. This repo uses *git flow*, so developement mostly take place on *develope* branch. To set up *pysds011* for local development:

1. Fork pysds011 (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:yourfork/pysds011.git
```

3. Create a branch for local development (maybe from *develope* branch):

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes and run all *tests* and checks.
5. 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
```

6. Submit a pull request through the GitHub website.

5.4.2 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (`tox` will come soon)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

5.4.3 Run unit testing

This project uses pytest. Tests are in *tests/* folder. This project is *virtualenv* friendly. To create the test environment, run once:

```
virtualenv venv
source venv/bin/activate
pip install -e .
pip install -r tests/requirements.txt
```

To run tests:

```
pytest tests/
```

CHAPTER

SIX

AUTHORS

Michele Pagot - https://michelepagot.github.io/

CHAPTER

SEVEN

CHANGELOG

7.1 0.0.4 (2021-2-7)

- add more commands to cli
- Testing and documentation

7.2 0.0.3 (2021-1-8)

- cli get subcommands and produce meaningful results
- robustness about error handling
- Testing and documentation

7.3 0.0.2 (2021-1-4)

- Improved cli (first functional).

7.4 0.0.1 (2020-12-19)

- First release on PyPI.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

Symbols

C

I

M

P

S