Pysensor

Python package to read modbus packages from a sensor and save to CSV and MySQL server. Runs on Raspbian OS, using pymodbus, a RS485 CAN HAT, and a serial connection.

Installation

Install the pysensor folder into your user directory (/home/{USER}/pysensor or ~/pysensor).

Configuration

Enabling sensor commands

1. Create a bin folder in your user directory if it does not exist.
2. Create a symbolic link from ~/bin to ~/pysensor/bin/sensor.

Run `find . -type l -ls` from the home directory. The symbolic link should appear as such.

```
./bin/sensor -> /home/{USER}/pysensor/bin/sensor
```

3. Run `chmod +x /home/{USER}/pysensor/bin/sensor` to enable the bash executable.
4. Run `sensor help`. If everything is working, the help message should appear with information about its commands and options.
5. Finally, run `sensor config` to set your editor (vim, nano, etc) and connection port to the sensor (such as /dev/ttyS0). The default values can be found in ~/pysensor/bin/pysensor.defaults.conf

Usage

Retrieving values from sensor

Using registermap.json (pysensor method)

There is a `.registermap.json` file in the main `pysensor` folder. It is a python dictionary in this format:

```
{"value_name":[address, count, dec]}

# address = register address
# count = number of bytes to read
# dec = n, of which the raw value obtained is divided by 10^n
```

Your sensor manual should have a table of modbus register values, known as a register map. Pysensor uses pymodbus to read these values. More can be learnt from pymodbus documentation, but here is a brief usage explanation.

`address` is the `register address` listed next to the value in the register map. It is suggested you use the value in the register map for `value_name`, for consistency.

`count` is the number of bytes to read. This value will be 1 or 2, as it will be either a single(1) byte package or a double(2) byte package. If the address of the

next value skips a number, it is a double byte package, and if it is consecutive, it is a single byte package.

`dec` is used to obtain the true value of a raw number from the sensor. The sensor does not store floats, so the raw number obtained does not have a decimal point. The register map will have a 'Multiplier' column, showing by what factor of 10 the true value has been multiplied. In order to obtain the true value from the raw number, the raw number will need to be divided by the multiplier, placing a decimal point in the correct location.

For example, the `Ia` value is listed to have a multiplier of `x100`. A multiplier of `x100` is equal to `10^2`. Hence, `dec` will be `2`. The raw number of `12345` will be converted to `123.45`

NOTE: `dec` is a custom argument for the pysensor Retrieve class. pymodbus does not have the `dec` argument, but it does have the `address` and `count` arguments for reading modbus values.

Reading consecutive values

You may notice that similar values such as Ia, Ib, Ic are next to each other on the register map table. You can read them all at once and group them under the same `value_name`, by increasing the `count` value. Here is such an example.

```
Imagine that register map values 0-5 (6 bytes) are reserved for Ia, Ib, Ic.
Hence, starting from address 0, 6 bytes are read, so all three values can be read at
the same time. address = 0 and count = 6
Furthermore, they have a multiplier of x100, so dec = 2.
The .registermap.json entry would appear as such.

{"current":[0, 6, 2]}

And the retrieved value list would appear as: [0,Ia,0,Ib,0,Ic]
```

A word about double byte packages

In the example above, a double byte package was read. However, only one byte contains the value we want (Ia), while another byte is empty. (0)

Double byte packages will typically have an empty zero value, unless both bits are used to contain numbers. It will be represented as `[0, real value]`.

The cook() function in the Retrieve class of pysensor is used to remove the empty zero values from the obtained list. For single byte packages, cookone() is used.

Using only pymodbus

More can be learnt from pymodbus documentation, but here is a simple example of how to obtain a value without relying on the Retrieve class of pysensor. This may be used for testing and debugging.

This is the basic process:

1. Build `client` object
2. Connect to sensor: `client.connect()`
3. Read holding register: `output = client.read_holding_registers(address,count,serial id)`

4. Convert to list: `output.registers`
5. Close connection: `client.close()`

```
from pymodbus.client import ModbusSerialClient
from pymodbus.transaction import ModbusRtuFramer
from pymodbus.register_read_message import ReadHoldingRegistersResponse

def retrieve(port):
        """Configures and returns serial client object. See pymodbus documentation for the arguments used. """
        client = ModbusSerialClient(
        port,
        framer=ModbusRtuFramer,
        timeout=2,
        retry_on_empty=True,
        baudrate=9600,
        strict=False,
        )
        try:
            client.connect()

            raw = client.read_holding_registers(0,2,1)
            # 0 = address, 2 = count, 1 = serial ID (usually 1)
            cooked = raw.registers
            # .registers method is used to convert the raw value output into a value list
            print(cooked)
        finally:
            client.close()

if __name__ == '__main__':
    retrieve(port)
```

Storing values from sensor

Pysensor stores records to a MySql database online, powered by Planetscale. It will also store a local CSV file daily. Each day, any record older than the last 30 days is wiped by the `sqlcleanup.py` and `csvcleanup.py` scripts, also controlled by CRON.

Planetscale web console

To quickly query the database, log into Planetscale and use the 'Console' tab. This will open a command line for you to enter MySQL queries in. Look up MySQL documentation for how to use MySQL commands.

For security's sake, some commands are not permitted to run on a "production branch", but there is a separate "development branch" for you to use. Planetscale has its own documentation about how to make serious changes to your database. If you know what you are doing, disable the "production branch" status in the Overview tab and make your changes.

Some frequently used query examples:

The last row on record

```
SELECT * FROM sensor ORDER BY PK_no DESC LIMIT 1;
```

The first row on record

```
SELECT * FROM sensor ORDER BY PK_no ASC LIMIT 1;
```

Specific time

```
SELECT * FROM sensor WHERE DATETIME(read_at) = DATETIME('2023-09-11 12:00:00');
```

Specific time range

```
SELECT * FROM sensor WHERE DATETIME(read_at) BETWEEN DATETIME('2023-09-11 12:00:00')
AND DATETIME('2023-09-12 12:00:00');
```

Notes:

- For BETWEEN - AND clause, it is BETWEEN {older time} AND {newer time}.
- The DATE() type can also be used instead of DATETIME(), if time does not have to be specified.

mysql.connector

Further documentation is at MySql page. This is a Python package meant for working with a MySql database.

The connection is made using a `.env` file, which must be named only `.env`. It stores username, password, and other server details and arguments. If you change username or generate a new password with planetscale, the change must be reflected in the `.env` file.

Native logger

Pysensor comes with a native logger, `logger.py`. It uses the python `logging` module.

The log is stored in `~/pysensor/log/pysensor.log` and can be read with `sensor readlog -n` or `sensor readlog all`, from the command line. (n is the number of lines to read)

To use in a module inside the pysensor folder, use `from logger import logger`.

If it is outside the pysensor folder, use `from pysensor.logger import logger`

sensor command

This is the help message of the sensor command, which can be obtained with `sensor -h`.

```
Options:
    -h, help Show this message again

  -s, status Show status of pysensor functions, with options as below
         sql Connect to MySQL server and retrieve time of last record
         csv Time of last record in local CSV backup
      serial Connect to sensor and retrieves sensor information
        cron CRON service status
     -a, all Show status of all of the above
```

```
    -e, edit Edit various pysensor scripts, with options as below
        main main pysensor script
      status status functions of pysensor
         csv last CSV backup file
        cron cron jobs of pysensor
        bash bash file that controls the 'sensor' command
      logger module that contains logging config
      config sensor command config file

        Misc Miscellaneous one word options
       start Starts CRON service (enables program)
        stop Stops CRON service (disables program)

     readlog
          -n View last n lines of logger
         all View all lines of logger

        mail View mail queue
        path View current path of pysensor
```