# PySMA

**Johann Kellerman, René Klomp**

**Jul 05, 2021**

# CONTENTS:


1 Getting Started 1

2 PySMA Main Class 3

3 Submodules 5

4 Indices and tables 17

Python Module Index 19

Index 21

CHAPTER

ONE

# GETTING STARTED

## 1.1 Install

PySMA is available on pypi and can be installed using pip.

```
pip install pysma
```

## 1.2 Create SMA instance

The *SMA* class requires a ClientSession object, an URL and a password. The default user group is "user", but can be changed by passing it as the fourth `group` parameter.

```
session = aiohttp.ClientSession(connector=aiohttp.TCPConnector(ssl=False))
url = "https://device-hostname"
password = "MyPassword!"

sma = pysma.SMA(session, url, password)
```

## 1.3 Create Sensors

To retrieve values from the device you need a *Sensors* object. The easiest way is to generate one using the *get_sensors()* method. This will query the device to figure out the device class and returns a *Sensors* object containing specific *Sensor* objects for your device.

```
sma_sensors = sma.get_sensors()
```

Alternatively this can be manually created by initializing an empty *Sensors* object and adding new *Sensor* objects to it using the *add()* method. Predefined sensors can be found in *pysma.definitions*. See also definitions.py at Github.

```
sma_sensors = Sensors()
my_sensor = Sensor("6300_12345678_0", "dummy_sensor") # This key won't work!
sma_sensors.add(my_sensor)
sma_sensors.add(pysma.definitions.pv_power_a)
```

## 1.4 Read Sensor values

Now you have a *Sensors* object, you can pass this to *read()* to read the values from the device. The retrieved values are stored in the respective *Sensor*.

```
sma.read(sma_sensors)

for sma_sensor in sma_sensors:
    print(f"{sma_sensor.name}: {sma_sensor.value}")
```

## 1.5 Complete Example

A full example can be found in the Github repository

CHAPTER

# TWO

# PYSMA MAIN CLASS

SMA WebConnect library for Python.

See: http://www.sma.de/en/products/monitoring-control/webconnect.html

Source: http://www.github.com/kellerza/pysma

**class** pysma.**SMA**(*session: aiohttp.client.ClientSession*, *url: str*, *password: Optional[str] = None*, *group: str = 'user'*, *uid: Optional[str] = None*)

Bases: `object`

Class to connect to the SMA webconnect module and read parameters.

Init SMA connection.

**Parameters**

- **session** (*ClientSession*) – aiohttp client session
- **url** (*str*) – Url or IP address of device
- **password** (*str, optional*) – Password to use during login. Defaults to None.
- **group** (*str, optional*) – Username to use during login. Defaults to "user".
- **uid** (*str, optional*) – uid used for data extraction. Defaults to None.

**Raises** **KeyError** – User was not in USERS

**async close_session**() → None
Close the session login.

**async device_info**() → dict
Read device info and return the results.

**Returns** dict containing serial, name, type, manufacturer and sw_version

**Return type** dict

**async get_devclass**(*result_body: Optional[dict] = None*) → Optional[str]
Get the device class.

**Parameters** **result_body** (*dict, optional*) – result body to extract device class from. Defaults to None.

**Raises** **KeyError** – More than 1 device class key is not supported

**Returns** The device class identifier, or None if no identifier was found

**Return type** str

**async get_sensors**() → *pysma.sensor.Sensors*
Get the sensors based on the device class.

**Returns** Sensors object containing Sensor objects

**Return type** *Sensors*

**async new_session**() → bool
Establish a new session.

**Raises** ***SmaAuthenticationException*** – Authentication failed

**Returns** authentication successful

**Return type** bool

**async read**(*sensors:* pysma.sensor.Sensors) → bool
Read a set of keys.

**Parameters** **sensors** (Sensors) – Sensors object containing Sensor objects to read

**Returns** reading was successful

**Return type** bool

**async read_dash_logger**() → dict
Read the dash loggers.

**Returns** Dictionary containing loggers returned by device.

**Return type** dict

**async read_logger**(*log_id: int*, *start: int*, *end: int*) → list
Read a logging key and return the results.

**Parameters**

- **log_id** (*int*) – The ID of the log to read. totWhOut5min: 28672 totWhOutDaily: 28704 GridMsTotWhOutDaily: 28752 GridMsTotWhInDaily: 28768 ObjLogBatCha: 28816 totWhIn5min: 28736 totWhInDaily: 28768 ObjLogBatChrg: 29344 ObjLogBatDsch: 29360
- **start** (*int*) – Start timestamp in seconds.
- **end** (*int*) – End timestamp in seconds.

**Returns** The log entries returned by the device

**Return type** list

CHAPTER

# THREE

# SUBMODULES

## 3.1 pysma.const module

Constants for SMA WebConnect library for Python.

## 3.2 pysma.definitions module

Sensor definitions for SMA WebConnect library for Python.

pysma.definitions.**battery_capacity_a = Sensor(key='6100_00499100', name='battery_capacity_a', unit='%', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**
Capacity battery A

pysma.definitions.**battery_capacity_b = Sensor(key='6100_00499100', name='battery_capacity_b', unit='%', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**
Capacity battery B

pysma.definitions.**battery_capacity_c = Sensor(key='6100_00499100', name='battery_capacity_c', unit='%', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**
Capacity battery C

pysma.definitions.**battery_capacity_total = Sensor(key='6100_00696E00', name='battery_capacity_total', unit='%', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Total battery capacity

pysma.definitions.**battery_charge_a = Sensor(key='6400_00499500', name='battery_charge_a', unit='kWh', factor=1000, path=None, enabled=False, l10n_translate=False, value=None)**
Charge battery A

pysma.definitions.**battery_charge_b = Sensor(key='6400_00499500', name='battery_charge_b', unit='kWh', factor=1000, path=None, enabled=False, l10n_translate=False, value=None)**
Charge battery B

pysma.definitions.**battery_charge_c = Sensor(key='6400_00499500', name='battery_charge_c', unit='kWh', factor=1000, path=None, enabled=False, l10n_translate=False, value=None)**
Charge battery C

pysma.definitions.**battery_charge_total = Sensor(key='6400_00496700', name='battery_charge_total', unit='kWh', factor=1000, path=None, enabled=True, l10n_translate=False, value=None)**
Total charge

pysma.definitions.**battery_charging_voltage_a = Sensor(key='6102_00493500', name='battery_charging_voltage_a', unit='V', factor=100, path=None, enabled=True, l10n_translate=False, value=None)**
Charging voltage battery A

pysma.definitions.**battery_charging_voltage_b = Sensor(key='6102_00493500', name='battery_charging_voltage_b', unit='V', factor=100, path=None, enabled=True, l10n_translate=False, value=None)**
Charging voltage battery B

pysma.definitions.**battery_charging_voltage_c = Sensor(key='6102_00493500', name='battery_charging_voltage_c', unit='V', factor=100, path=None, enabled=True, l10n_translate=False, value=None)**
Charging voltage battery C

pysma.definitions.**battery_current_a = Sensor(key='6100_40495D00', name='battery_current_a', unit='A', factor=1000, path=None, enabled=True, l10n_translate=False, value=None)**
Current battery A

pysma.definitions.**battery_current_b = Sensor(key='6100_40495D00', name='battery_current_b', unit='A', factor=1000, path=None, enabled=True, l10n_translate=False, value=None)**
Current battery B

pysma.definitions.**battery_current_c = Sensor(key='6100_40495D00', name='battery_current_c', unit='A', factor=1000, path=None, enabled=True, l10n_translate=False, value=None)**
Current battery C

pysma.definitions.**battery_discharge_a = Sensor(key='6400_00499600', name='battery_discharge_a', unit='kWh', factor=1000, path=None, enabled=False, l10n_translate=False, value=None)**
Discharge battery A

pysma.definitions.**battery_discharge_b = Sensor(key='6400_00499600', name='battery_discharge_b', unit='kWh', factor=1000, path=None, enabled=False, l10n_translate=False, value=None)**
Discharge battery B

pysma.definitions.**battery_discharge_c = Sensor(key='6400_00499600', name='battery_discharge_c', unit='kWh', factor=1000, path=None, enabled=False, l10n_translate=False, value=None)**
Discharge battery C

pysma.definitions.**battery_discharge_total = Sensor(key='6400_00496800', name='battery_discharge_total', unit='kWh', factor=1000, path=None, enabled=True, l10n_translate=False, value=None)**
Total discharge

pysma.definitions.**battery_power_charge_a = Sensor(key='6100_00499300', name='battery_power_charge_a', unit='W', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**
Charging power battery A

pysma.definitions.**battery_power_charge_b = Sensor(key='6100_00499300', name='battery_power_charge_b', unit='W', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**

Charging power battery B

pysma.definitions.**battery_power_charge_c = Sensor(key='6100_00499300', name='battery_power_charge_c', unit='W', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**

Charging power battery C

pysma.definitions.**battery_power_charge_total = Sensor(key='6100_00496900', name='battery_power_charge_total', unit='W', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**

Total charging power

pysma.definitions.**battery_power_discharge_a = Sensor(key='6100_00499400', name='battery_power_discharge_a', unit='W', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**

Disharging power battery A

pysma.definitions.**battery_power_discharge_b = Sensor(key='6100_00499400', name='battery_power_discharge_b', unit='W', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**

Disharging power battery B

pysma.definitions.**battery_power_discharge_c = Sensor(key='6100_00499400', name='battery_power_discharge_c', unit='W', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**

Disharging power battery C

pysma.definitions.**battery_power_discharge_total = Sensor(key='6100_00496A00', name='battery_power_discharge_total', unit='W', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**

Total discharging power

pysma.definitions.**battery_soc_a = Sensor(key='6100_00498F00', name='battery_soc_a', unit='%', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**

State of charge battery A

pysma.definitions.**battery_soc_b = Sensor(key='6100_00498F00', name='battery_soc_b', unit='%', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**

State of charge battery B

pysma.definitions.**battery_soc_c = Sensor(key='6100_00498F00', name='battery_soc_c', unit='%', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**

State of charge battery C

pysma.definitions.**battery_soc_total = Sensor(key='6100_00295A00', name='battery_soc_total', unit='%', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**

Total battery state of charge

pysma.definitions.**battery_status_operating_mode = Sensor(key='6180_08495E00', name='battery_status_operating_mode', unit='', factor=None, path=('"{}"[{}].val[0].tag', 'val[0].tag'), enabled=True, l10n_translate=True, value=None)**

Battery status operating mode

pysma.definitions.**battery_temp_a = Sensor(key='6100_40495B00', name='battery_temp_a', unit='C', factor=10, path=None, enabled=True, l10n_translate=False, value=None)**

Temperature battery A

pysma.definitions.**battery_temp_b = Sensor(key='6100_40495B00', name='battery_temp_b', unit='C', factor=10, path=None, enabled=True, l10n_translate=False, value=None)**
Temperature battery B

pysma.definitions.**battery_temp_c = Sensor(key='6100_40495B00', name='battery_temp_c', unit='C', factor=10, path=None, enabled=True, l10n_translate=False, value=None)**
Temperature battery C

pysma.definitions.**battery_voltage_a = Sensor(key='6100_00495C00', name='battery_voltage_a', unit='V', factor=100, path=None, enabled=True, l10n_translate=False, value=None)**
Voltage battery A

pysma.definitions.**battery_voltage_b = Sensor(key='6100_00495C00', name='battery_voltage_b', unit='V', factor=100, path=None, enabled=True, l10n_translate=False, value=None)**
Voltage battery B

pysma.definitions.**battery_voltage_c = Sensor(key='6100_00495C00', name='battery_voltage_c', unit='V', factor=100, path=None, enabled=True, l10n_translate=False, value=None)**
Voltage battery C

pysma.definitions.**current_l1 = Sensor(key='6100_40465300', name='current_l1', unit='A', factor=1000, path=None, enabled=False, l10n_translate=False, value=None)**
Current for phase 1

pysma.definitions.**current_l2 = Sensor(key='6100_40465400', name='current_l2', unit='A', factor=1000, path=None, enabled=False, l10n_translate=False, value=None)**
Current for phase 2

pysma.definitions.**current_l3 = Sensor(key='6100_40465500', name='current_l3', unit='A', factor=1000, path=None, enabled=False, l10n_translate=False, value=None)**
Current for phase 3

pysma.definitions.**current_total = Sensor(key='6100_00664F00', name='current_total', unit='A', factor=1000, path=None, enabled=True, l10n_translate=False, value=None)**
Total Current

pysma.definitions.**daily_yield = Sensor(key='6400_00262200', name='daily_yield', unit='Wh', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
The solar plant's yield for today

pysma.definitions.**device_manufacturer = Sensor(key='6800_08822B00', name='device_manufacturer', unit='', factor=None, path=('"{}"[{}].val[0].tag', 'val[0].tag'), enabled=True, l10n_translate=True, value=None)**
Device manufactorer

pysma.definitions.**device_name = Sensor(key='6800_10821E00', name='device_name', unit='', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Device name

pysma.definitions.**device_sw_version = Sensor(key='6800_00823400', name='device_sw_version', unit='', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Device software version

pysma.definitions.**device_type = Sensor(key='6800_08822000', name='device_type', unit='', factor=None, path=('"{}"[{}].val[0].tag', 'val[0].tag'), enabled=True, l10n_translate=True, value=None)**
Device type

pysma.definitions.**energy_meter = Sensor(key='6800_008AA300', name='energy_meter', unit='', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Serial number of energy meter

pysma.definitions.**frequency = Sensor(key='6100_00465700', name='frequency', unit='Hz', factor=100, path=None, enabled=True, l10n_translate=False, value=None)**
Grid frequency

pysma.definitions.**grid_apparent_power = Sensor(key='6100_40666700', name='grid_apparent_power', unit='VA', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Total Apparent Power

pysma.definitions.**grid_apparent_power_l1 = Sensor(key='6100_40666800', name='grid_apparent_power_l1', unit='VA', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**
Apparent Power for phase 1

pysma.definitions.**grid_apparent_power_l2 = Sensor(key='6100_40666900', name='grid_apparent_power_l2', unit='VA', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**
Apparent Power for phase 2

pysma.definitions.**grid_apparent_power_l3 = Sensor(key='6100_40666A00', name='grid_apparent_power_l3', unit='VA', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**
Apparent Power for phase 3

pysma.definitions.**grid_connection_status = Sensor(key='6180_0846A700', name='grid_connection_status', unit='', factor=None, path=('"{}"[{}].val[0].tag', 'val[0].tag'), enabled=False, l10n_translate=True, value=None)**
Grid connection status

pysma.definitions.**grid_power = Sensor(key='6100_40263F00', name='grid_power', unit='W', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Power supplied to the grid. grid_power = power_l1 + power_l2 + power_l3

pysma.definitions.**grid_power_factor = Sensor(key='6100_00665900', name='grid_power_factor', unit='', factor=1000, path=None, enabled=False, l10n_translate=False, value=None)**
Grid Power factor

pysma.definitions.**grid_power_factor_excitation = Sensor(key='6180_08465A00', name='grid_power_factor_excitation', unit='', factor=None, path=('"{}"[{}].val[0].tag', 'val[0].tag'), enabled=False, l10n_translate=True, value=None)**
Grid Power factor excitation

pysma.definitions.**grid_reactive_power = Sensor(key='6100_40265F00', name='grid_reactive_power', unit='var', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Total Reactive Power

pysma.definitions.**grid_reactive_power_l1 = Sensor(key='6100_40666000', name='grid_reactive_power_l1', unit='var', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**
Reactive Power for phase 1

pysma.definitions.**grid_reactive_power_l2 = Sensor(key='6100_40666100', name='grid_reactive_power_l2', unit='var', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**
Reactive Power for phase 2

pysma.definitions.**grid_reactive_power_l3 = Sensor(key='6100_40666200', name='grid_reactive_power_l3', unit='var', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**
Reactive Power for phase 3

pysma.definitions.**grid_relay_status = Sensor(key='6180_08416400', name='grid_relay_status', unit='', factor=None, path=('"{}"[{}].val[0].tag', 'val[0].tag'), enabled=False, l10n_translate=True, value=None)**
Grid relay status

pysma.definitions.**insulation_residual_current = Sensor(key='6102_40254E00', name='insulation_residual_current', unit='mA', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Insulation residual current

pysma.definitions.**inverter_condition = Sensor(key='6180_08414C00', name='inverter_condition', unit='', factor=None, path=('"{}"[{}].val[0].tag', 'val[0].tag'), enabled=False, l10n_translate=True, value=None)**
General operating status

pysma.definitions.**inverter_power_limit = Sensor(key='6800_00832A00', name='inverter_power_limit', unit='W', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Power limit of the Inverter

pysma.definitions.**inverter_system_init = Sensor(key='6800_08811F00', name='inverter_system_init', unit='', factor=None, path=('"{}"[{}].val[0].tag', 'val[0].tag'), enabled=False, l10n_translate=True, value=None)**
Inverter Condition

pysma.definitions.**metering_active_power_consumed_l1 = Sensor(key='6100_0046EB00', name='metering_active_power_consumed_l1', unit='W', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Active Power Consumed for phase 1 measured by energy meter

pysma.definitions.**metering_active_power_consumed_l2 = Sensor(key='6100_0046EC00', name='metering_active_power_consumed_l2', unit='W', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Active Power Consumed for phase 2 measured by energy meter

pysma.definitions.**metering_active_power_consumed_l3 = Sensor(key='6100_0046ED00', name='metering_active_power_consumed_l3', unit='W', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Active Power Consumed for phase 3 measured by energy meter

pysma.definitions.**metering_active_power_l1 = Sensor(key='6100_0046E800', name='metering_active_power_l1', unit='W', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Active Power for phase 1 measured by energy meter

pysma.definitions.**metering_active_power_l2 = Sensor(key='6100_0046E900', name='metering_active_power_l2', unit='W', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Active Power for phase 2 measured by energy meter

pysma.definitions.**metering_active_power_l3 = Sensor(key='6100_0046EA00', name='metering_active_power_l3', unit='W', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Active Power for phase 3 measured by energy meter

pysma.definitions.**metering_current_l1 = Sensor(key='6100_40466500', name='metering_current_l1', unit='A', factor=1000, path=None, enabled=True, l10n_translate=False, value=None)**
Current for phase 1 measured by energy meter

pysma.definitions.**metering_current_l2 = Sensor(key='6100_40466600', name='metering_current_l2', unit='A', factor=1000, path=None, enabled=True, l10n_translate=False, value=None)**
Current for phase 2 measured by energy meter

pysma.definitions.**metering_current_l3 = Sensor(key='6100_40466B00', name='metering_current_l3', unit='A', factor=1000, path=None, enabled=True, l10n_translate=False, value=None)**
Current for phase 3 measured by energy meter

pysma.definitions.**metering_frequency = Sensor(key='6100_00468100', name='metering_frequency', unit='Hz', factor=100, path=None, enabled=True, l10n_translate=False, value=None)**
Grid frequency measured by energy meter

pysma.definitions.**metering_power_absorbed = Sensor(key='6100_40463700', name='metering_power_absorbed', unit='W', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Power absorbed fromgrid measured by energy meter

pysma.definitions.**metering_power_supplied = Sensor(key='6100_40463600', name='metering_power_supplied', unit='W', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Power supplied to grid measured by energy meter

pysma.definitions.**metering_total_absorbed = Sensor(key='6400_00462500', name='metering_total_absorbed', unit='kWh', factor=1000, path=None, enabled=True, l10n_translate=False, value=None)**
Total power from the grid measured by energy meter

pysma.definitions.**metering_total_yield = Sensor(key='6400_00462400', name='metering_total_yield', unit='kWh', factor=1000, path=None, enabled=True, l10n_translate=False, value=None)**
Total power supplied to the grid measured by energy meter

pysma.definitions.**metering_voltage_l1 = Sensor(key='6100_0046E500', name='metering_voltage_l1', unit='V', factor=100, path=None, enabled=True, l10n_translate=False, value=None)**
Voltage for phase 1 measured by energy meter

pysma.definitions.**metering_voltage_l2 = Sensor(key='6100_0046E600', name='metering_voltage_l2', unit='V', factor=100, path=None, enabled=True, l10n_translate=False, value=None)**
Voltage for phase 2 measured by energy meter

pysma.definitions.**metering_voltage_l3 = Sensor(key='6100_0046E700', name='metering_voltage_l3', unit='V', factor=100, path=None, enabled=True, l10n_translate=False, value=None)**
Voltage for phase 3 measured by energy meter

pysma.definitions.**operating_status_general = Sensor(key='6180_08412800', name='operating_status_general', unit='', factor=None, path=('"{}"[{}].val[0].tag', 'val[0].tag'), enabled=False, l10n_translate=True, value=None)**
General operating status

pysma.definitions.**optimizer_current = Sensor(key='6100_40652900', name='optimizer_current', unit='A', factor=1000, path=None, enabled=False, l10n_translate=False, value=None)**
Current supplied by optimizer

pysma.definitions.**optimizer_power = Sensor(key='6100_40652A00', name='optimizer_power', unit='W', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Power supplied by optimizer

pysma.definitions.**optimizer_serial = Sensor(key='6800_10852600', name='optimizer_serial', unit='', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Serial number of optimizer

pysma.definitions.**optimizer_temp = Sensor(key='6100_40652B00', name='optimizer_temp', unit='C', factor=10, path=None, enabled=False, l10n_translate=False, value=None)**
Temperature of optimizer

pysma.definitions.**optimizer_voltage = Sensor(key='6100_40652800', name='optimizer_voltage', unit='V', factor=100, path=None, enabled=False, l10n_translate=False, value=None)**
Voltage supplied by optimizer

pysma.definitions.**power_l1 = Sensor(key='6100_40464000', name='power_l1', unit='W', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**
Power for phase 1

pysma.definitions.**power_l2 = Sensor(key='6100_40464100', name='power_l2', unit='W', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**
Power for phase 2

pysma.definitions.**power_l3 = Sensor(key='6100_40464200', name='power_l3', unit='W', factor=None, path=None, enabled=False, l10n_translate=False, value=None)**
Power for phase 3

pysma.definitions.**pv_current_a = Sensor(key='6380_40452100', name='pv_current_a', unit='A', factor=1000, path=None, enabled=True, l10n_translate=False, value=None)**
Current amperage generated by the solar panels (A side)

pysma.definitions.**pv_current_b = Sensor(key='6380_40452100', name='pv_current_b', unit='A', factor=1000, path=None, enabled=True, l10n_translate=False, value=None)**
Current amperage generated by the solar panels (B side)

pysma.definitions.**pv_gen_meter = Sensor(key='6400_0046C300', name='pv_gen_meter', unit='kWh', factor=1000, path=None, enabled=True, l10n_translate=False, value=None)**
Total kWh generated to date

pysma.definitions.**pv_power_a = Sensor(key='6380_40251E00', name='pv_power_a', unit='W', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Current power generated by the solar panels (A side)

pysma.definitions.**pv_power_b = Sensor(key='6380_40251E00', name='pv_power_b', unit='W', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Current power generated by the solar panels (B side)

pysma.definitions.**pv_voltage_a = Sensor(key='6380_40451F00', name='pv_voltage_a', unit='V', factor=100, path=None, enabled=True, l10n_translate=False, value=None)**
Current voltage generated by the solar panels (A side)

pysma.definitions.**pv_voltage_b = Sensor(key='6380_40451F00', name='pv_voltage_b', unit='V', factor=100, path=None, enabled=True, l10n_translate=False, value=None)**
Current voltage generated by the solar panels (B side)

pysma.definitions.**serial_number = Sensor(key='6800_00A21E00', name='serial_number', unit='', factor=None, path=None, enabled=True, l10n_translate=False, value=None)**
Device serial number

pysma.definitions.**status = Sensor(key='6180_08214800', name='status', unit='', factor=None, path=('"{}"[{}].val[0].tag', 'val[0].tag'), enabled=True, l10n_translate=True, value=None)**
Status of the device

pysma.definitions.**total_yield = Sensor(key='6400_00260100', name='total_yield', unit='kWh', factor=1000, path=None, enabled=True, l10n_translate=False, value=None)**
Total power yield from a solar installation

pysma.definitions.**voltage_l1 = Sensor(key='6100_00464800', name='voltage_l1', unit='V', factor=100, path=None, enabled=True, l10n_translate=False, value=None)**
Voltage for phase 1

pysma.definitions.**voltage_l2 = Sensor(key='6100_00464900', name='voltage_l2', unit='V', factor=100, path=None, enabled=True, l10n_translate=False, value=None)**
Voltage for phase 2

pysma.definitions.**voltage_l3 = Sensor(key='6100_00464A00', name='voltage_l3', unit='V', factor=100, path=None, enabled=True, l10n_translate=False, value=None)**
Voltage for phase 3

## 3.3 pysma.exceptions module

Exceptions for the pysma library.

**exception** pysma.exceptions.**SmaAuthenticationException**
Bases: *pysma.exceptions.SmaException*

An attempt to authenticate failed.

**exception** pysma.exceptions.**SmaConnectionException**
Bases: *pysma.exceptions.SmaException*

An error occurred in the connection with the device.

**exception** pysma.exceptions.**SmaException**
Bases: Exception

Base exception of the pysma library.

**exception** pysma.exceptions.**SmaReadException**
Bases: *pysma.exceptions.SmaException*

Reading the data did not return an expected result.

## 3.4 pysma.helpers module

Helper functions for the pysma library.

pysma.helpers.**version_int_to_string**(*version_integer: int*) → str
Convert a version integer to a readable string.

**Parameters** **version_integer** (*int*) – The version integer, as retrieved from the device.

**Returns** The version translated to a readable string.

**Return type** str

## 3.5 pysma.sensor module

Sensor classes for SMA WebConnect library for Python.

class pysma.sensor.**Sensor**(*key: str*, *name: str*, *unit: str = ''*, *factor: Optional[int] = None*, *path: Optional[Union[list, tuple]] = None*, *enabled: bool = True*, *l10n_translate: bool = False*)

Bases: object

pysma sensor.

Method generated by attrs for class Sensor.

**enabled**

**extract_value**(*result_body: dict*, *l10n: Optional[dict] = None*, *devclass: str = '1'*) → bool
Extract value from json body.

**Parameters**

- **result_body** (*dict*) – json body retrieved from device
- **l10n** (*dict, optional*) – Dictionary to translate tags to strings. Defaults to None.
- **devclass** (*str, optional*) – The device class of the device used to extract the value. Defaults to "1".

**Returns** Extracting value successful

**Return type** bool

**factor**

**key**

**key_idx**

**l10n_translate**

**name**

**path**

**unit**

**value**

class pysma.sensor.**Sensors**(*sensors: Optional[Union[pysma.sensor.Sensor, List[pysma.sensor.Sensor]]] = None*)

Bases: object

SMA Sensors.

Init Sensors.

**Parameters** **sensors** (*Union[Sensor, List[Sensor], None], optional*) – One or a list of sensors to add on init. Defaults to None.

**add**(*sensor: Union[*pysma.sensor.Sensor*, List[*pysma.sensor.Sensor*]]*) → None
Add a sensor, logs warning if it exists.

A copy of sensor, or the sensors in the list, will be added. If the sensor name already exists it will be overwritten. If the sensor key already exists it will not be added.

**Parameters** **sensor** (*Sensor, List[Sensor]*) – One or a list of sensors to add

**Raises** **TypeError** – Argument sensor does not match an expected type

CHAPTER

FOUR

# INDICES AND TABLES

- genindex
- modindex

# PYTHON MODULE INDEX

## p

pysma, 3
pysma.const, 5
pysma.definitions, 5
pysma.exceptions, 13
pysma.helpers, 14
pysma.sensor, 14

# INDEX


## A

add() (*pysma.sensor.Sensors method*), 15

## B

battery_capacity_a (*in module pysma.definitions*), 5
battery_capacity_b (*in module pysma.definitions*), 5
battery_capacity_c (*in module pysma.definitions*), 5
battery_capacity_total (*in module pysma.definitions*), 5
battery_charge_a (*in module pysma.definitions*), 5
battery_charge_b (*in module pysma.definitions*), 5
battery_charge_c (*in module pysma.definitions*), 5
battery_charge_total (*in module pysma.definitions*), 5
battery_charging_voltage_a (*in module pysma.definitions*), 6
battery_charging_voltage_b (*in module pysma.definitions*), 6
battery_charging_voltage_c (*in module pysma.definitions*), 6
battery_current_a (*in module pysma.definitions*), 6
battery_current_b (*in module pysma.definitions*), 6
battery_current_c (*in module pysma.definitions*), 6
battery_discharge_a (*in module pysma.definitions*), 6
battery_discharge_b (*in module pysma.definitions*), 6
battery_discharge_c (*in module pysma.definitions*), 6
battery_discharge_total (*in module pysma.definitions*), 6
battery_power_charge_a (*in module pysma.definitions*), 6
battery_power_charge_b (*in module pysma.definitions*), 6
battery_power_charge_c (*in module pysma.definitions*), 7
battery_power_charge_total (*in module pysma.definitions*), 7
battery_power_discharge_a (*in module pysma.definitions*), 7
battery_power_discharge_b (*in module pysma.definitions*), 7
battery_power_discharge_c (*in module pysma.definitions*), 7
battery_power_discharge_total (*in module pysma.definitions*), 7
battery_soc_a (*in module pysma.definitions*), 7
battery_soc_b (*in module pysma.definitions*), 7
battery_soc_c (*in module pysma.definitions*), 7
battery_soc_total (*in module pysma.definitions*), 7
battery_status_operating_mode (*in module pysma.definitions*), 7
battery_temp_a (*in module pysma.definitions*), 7
battery_temp_b (*in module pysma.definitions*), 7
battery_temp_c (*in module pysma.definitions*), 8
battery_voltage_a (*in module pysma.definitions*), 8
battery_voltage_b (*in module pysma.definitions*), 8
battery_voltage_c (*in module pysma.definitions*), 8

## C

close_session() (*pysma.SMA method*), 3
current_l1 (*in module pysma.definitions*), 8
current_l2 (*in module pysma.definitions*), 8
current_l3 (*in module pysma.definitions*), 8
current_total (*in module pysma.definitions*), 8

## D

daily_yield (*in module pysma.definitions*), 8
device_info() (*pysma.SMA method*), 3
device_manufacturer (*in module pysma.definitions*), 8
device_name (*in module pysma.definitions*), 8
device_sw_version (*in module pysma.definitions*), 8
device_type (*in module pysma.definitions*), 8

## E

enabled (*pysma.sensor.Sensor attribute*), 14
energy_meter (*in module pysma.definitions*), 8
extract_value() (*pysma.sensor.Sensor method*), 14

## F

factor (*pysma.sensor.Sensor attribute*), 14
frequency (*in module pysma.definitions*), 9

## G

get_devclass() (*pysma.SMA method*), 3

get_sensors() (*pysma.SMA method*), 3
grid_apparent_power (*in module pysma.definitions*), 9
grid_apparent_power_l1 (*in module pysma.definitions*), 9
grid_apparent_power_l2 (*in module pysma.definitions*), 9
grid_apparent_power_l3 (*in module pysma.definitions*), 9
grid_connection_status (*in module pysma.definitions*), 9
grid_power (*in module pysma.definitions*), 9
grid_power_factor (*in module pysma.definitions*), 9
grid_power_factor_excitation (*in module pysma.definitions*), 9
grid_reactive_power (*in module pysma.definitions*), 9
grid_reactive_power_l1 (*in module pysma.definitions*), 9
grid_reactive_power_l2 (*in module pysma.definitions*), 9
grid_reactive_power_l3 (*in module pysma.definitions*), 10
grid_relay_status (*in module pysma.definitions*), 10

## I

insulation_residual_current (*in module pysma.definitions*), 10
inverter_condition (*in module pysma.definitions*), 10
inverter_power_limit (*in module pysma.definitions*), 10
inverter_system_init (*in module pysma.definitions*), 10

## K

key (*pysma.sensor.Sensor attribute*), 14
key_idx (*pysma.sensor.Sensor attribute*), 14

## L

l10n_translate (*pysma.sensor.Sensor attribute*), 14

## M

metering_active_power_consumed_l1 (*in module pysma.definitions*), 10
metering_active_power_consumed_l2 (*in module pysma.definitions*), 10
metering_active_power_consumed_l3 (*in module pysma.definitions*), 10
metering_active_power_l1 (*in module pysma.definitions*), 10
metering_active_power_l2 (*in module pysma.definitions*), 10
metering_active_power_l3 (*in module pysma.definitions*), 10
metering_current_l1 (*in module pysma.definitions*), 10
metering_current_l2 (*in module pysma.definitions*), 11
metering_current_l3 (*in module pysma.definitions*), 11
metering_frequency (*in module pysma.definitions*), 11
metering_power_absorbed (*in module pysma.definitions*), 11
metering_power_supplied (*in module pysma.definitions*), 11
metering_total_absorbed (*in module pysma.definitions*), 11
metering_total_yield (*in module pysma.definitions*), 11
metering_voltage_l1 (*in module pysma.definitions*), 11
metering_voltage_l2 (*in module pysma.definitions*), 11
metering_voltage_l3 (*in module pysma.definitions*), 11
module
    pysma, 3
    pysma.const, 5
    pysma.definitions, 5
    pysma.exceptions, 13
    pysma.helpers, 14
    pysma.sensor, 14

## N

name (*pysma.sensor.Sensor attribute*), 14
new_session() (*pysma.SMA method*), 4

## O

operating_status_general (*in module pysma.definitions*), 11
optimizer_current (*in module pysma.definitions*), 11
optimizer_power (*in module pysma.definitions*), 12
optimizer_serial (*in module pysma.definitions*), 12
optimizer_temp (*in module pysma.definitions*), 12
optimizer_voltage (*in module pysma.definitions*), 12

## P

path (*pysma.sensor.Sensor attribute*), 14
power_l1 (*in module pysma.definitions*), 12
power_l2 (*in module pysma.definitions*), 12
power_l3 (*in module pysma.definitions*), 12
pv_current_a (*in module pysma.definitions*), 12
pv_current_b (*in module pysma.definitions*), 12
pv_gen_meter (*in module pysma.definitions*), 12
pv_power_a (*in module pysma.definitions*), 12
pv_power_b (*in module pysma.definitions*), 12
pv_voltage_a (*in module pysma.definitions*), 12
pv_voltage_b (*in module pysma.definitions*), 12
pysma
    module, 3

pysma.const
    module, 5
pysma.definitions
    module, 5
pysma.exceptions
    module, 13
pysma.helpers
    module, 14
pysma.sensor
    module, 14

## R

read() (*pysma.SMA method*), 4
read_dash_logger() (*pysma.SMA method*), 4
read_logger() (*pysma.SMA method*), 4

## S

Sensor (*class in pysma.sensor*), 14
Sensors (*class in pysma.sensor*), 14
serial_number (*in module pysma.definitions*), 12
SMA (*class in pysma*), 3
SmaAuthenticationException, 13
SmaConnectionException, 13
SmaException, 13
SmaReadException, 13
status (*in module pysma.definitions*), 13

## T

total_yield (*in module pysma.definitions*), 13

## U

unit (*pysma.sensor.Sensor attribute*), 14

## V

value (*pysma.sensor.Sensor attribute*), 14
version_int_to_string() (*in module pysma.helpers*), 14
voltage_l1 (*in module pysma.definitions*), 13
voltage_l2 (*in module pysma.definitions*), 13
voltage_l3 (*in module pysma.definitions*), 13