

neOCampus IoT API for end-devices

End-devices API and MQTT rules
Dr Thiebolt François, thiebolt@irit.fr

Modifications table

| Date | Note |
|--------|--|
| feb.21 | replaced 'location' with 'site' in our terminology |
| Sep.20 | expanded 'display' class with TXT and IMG messages |
| Aug.20 | Updated 'airquality' class |
| May.20 | added 'value' field specifications; added 'airquality' class |
| Apr.20 | minor correction about weather topic → outside/ambient/<rain wind...> clarify things about 'outside' MQTT topics, add attendance class topic |
| Mar.20 | additional topics conventions |
| Feb.20 | added access class topic |
| Jan.20 | added pressure, rain, wind ... classes (metropole weather station) |
| Mar.18 | added display class messages |
| Nov.17 | initial release |

Abstract

This guide explains how to authenticate against the neOCampus IoT chain and then how to publish / subscribe data through the neOCampus **MQTT broker** and its associated device management application named **sensOCampus**.

Table of contents

| | |
|--|----------|
| Abstract | 1 |
| End-devices | 4 |
| sensOCampus credentials API | 5 |
| get_credentials | 5 |
| get_config | 5 |
| summary | 6 |
| MQTT topics conventions | 7 |
| msg to a command topic | 7 |
| msg to a class topic | 8 |
| unitID and subID | 9 |
| scenario example | 9 |
| sensors auto detection and messages publishing | 10 |
| sensors and actuators uniqueness | 10 |
| neOCampus MQTT sandbox | 10 |
| example topics | 10 |
| special case topics | 11 |
| optional 'location' field | 11 |
| outside/<xxx> topics rules | 11 |
| Class topics and command topics | 13 |
| device | 15 |
| publish | 15 |
| subscribe | 15 |
| temperature / luminosity / co2 / humidity / pressure / weight / uv | 17 |
| publish | 17 |
| subscribe | 17 |
| rain | 19 |
| publish | 19 |
| subscribe | 19 |
| wind | 21 |
| publish | 21 |
| subscribe | 22 |
| energy | 23 |
| publish | 23 |
| subscribe | 24 |
| camera | 25 |
| digital | 26 |
| publish | 26 |
| subscribe | 26 |
| noise | 28 |

| | |
|--|-----------|
| publish | 28 |
| subscribe | 28 |
| attendance | 30 |
| publish | 30 |
| subscribe | 30 |
| airquality | 31 |
| publish | 31 |
| subscribe | 31 |
| lighting | 33 |
| publish | 33 |
| subscribe | 33 |
| dali | 34 |
| shutter | 35 |
| publish | 35 |
| subscribe | 35 |
| display | 36 |
| publish | 36 |
| subscribe | 36 |
| access | 39 |
| publish | 39 |
| subscribe | 39 |
| Annexe - A | 41 |
| A - 1 ESP8266 credentials sample code | 41 |
| A - 2 sensOCampus configuration U4/302 | 43 |
| A - 3 sensOCampus LCC's AirQuality configuration | 50 |

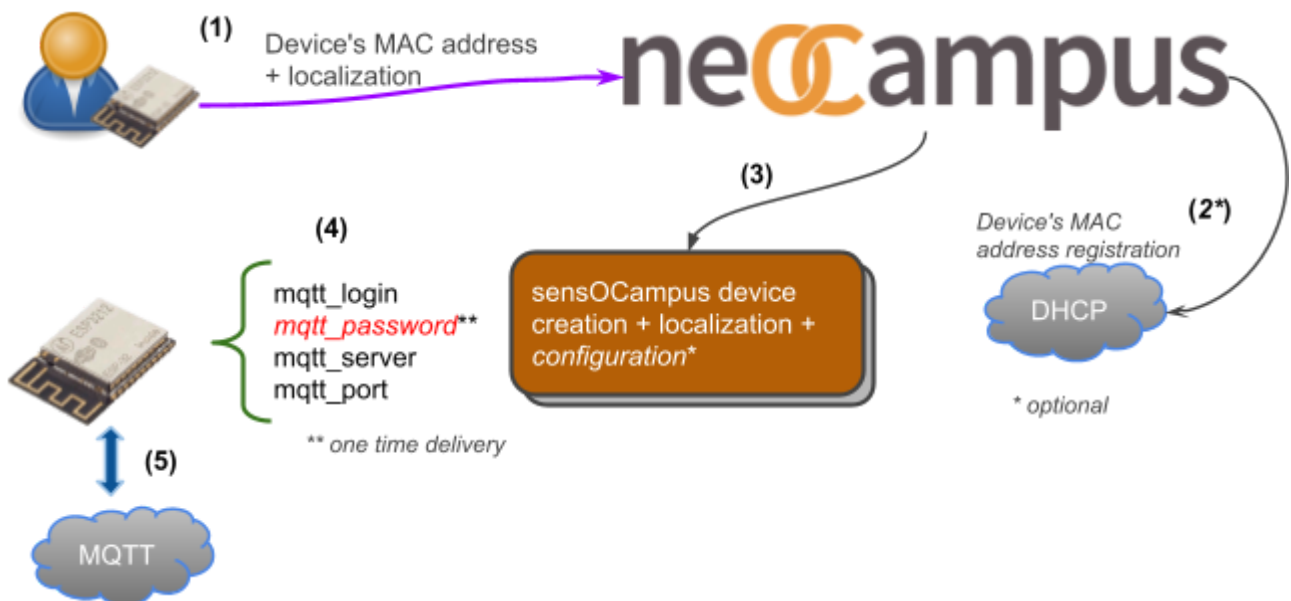
End-devices

What we call a **device (or end-device)** is a piece of **hardware connected to a network**. Such a device may encompass one to several sensors / actuators. It is devices' firmware responsibility to publish sensors values to the proper topic and to subscribe to relevant topics.



In the upper example, the device is a Raspberry Pi that could be connected to either a wired / wireless network. Each kind of sensor / actuators map to a topic class. However, to be able to publish / subscribe to the MQTT broker, the device's client needs credentials. To obtain these credentials, you first need to:

1. declare device's MAC address to the neOCampus technical staff,
2. interact with the sensOCampus application that will give you your credentials.



sensOCampus is the main end-devices management application developed for neOCampus. It takes care of managing the device's own specific setup (MAC, configuration, status, topics etc).

sensOCampus credentials API

We below describe the various involved steps that end-devices need to undertake with the sensOCampus application to retrieve their credentials.

1. `get_credentials` → will give your device its mqtt related credentials,
2. `get_config` → your device will be given a **MQTT base topic** along with optional configuration registered for each peculiar device at the sensOCampus level.

get_credentials

```
https://sensocampus.univ-tlse3.fr/device/credentials?mac=<device_mac_addr>
```

... response will be in JSON format

```
{
  "login" : "<mqtt_login>",
  "password" : "<mqtt_password>",
  "server" : "neocampus.univ-tlse3.fr",
  "port" : 1883
}
```

Please pay attention to the facts that:

- **password** field is a one-time delivery parameter → if you lose it, you need to apply for new credentials at the neOCampus technical staff,
- **"server"** and **"port"** fields are **optionals** → you ought to have these default values in your code if sensOCampus does not deliver them to your device.

It's the device's own responsibility to save these credentials in some non volatile hardware. If you apply for a credentials renewal operation, both login and password will change.

get_config

prepare HTTPS request with previously delivered credentials (https_auth) ...

```
get https://sensocampus.univ-tlse3.fr/device/config
```

... response will be in JSON format

```
{
  'zones': [],
  'topics': ['bu/hall']
}
```

In this simple example, we have no specific configuration (empty zones) and we must take into account the **topics** field as **BASE_TOPIC**.

Note: in case 'topics' contains multiple fields, just select the first one

summary

You now have the following:

| | |
|------------|---------------------------------|
| login | "<mqtt_login>" |
| password | "<mqtt_password>" |
| server | neocampus.univ-tlse3.fr |
| port | 1883 or 8883 (tls) |
| BASE_TOPIC | <building>/<room> (e.g bu/hall) |

Later, this BASE_TOPIC means that you've been granted the following topics rules:

| | |
|-------------------|--|
| bu/hall/+ | publish & subscribe (i.e write & read) |
| bu/hall/+/command | subscribe (i.e read) |

In the next section, we'll start to talk about the MQTT conventions that apply to neOCampus.

MQTT topics conventions

The following describes various rules about topics conventions that apply to the neOCampus IoT.

- Each **end-device** get specified a **BASE_TOPIC** through its `get_config` procedure,
- A **BASE_TOPIC** is named accordingly to `<building>/<room>`, eg. `u4/302`
- Each sensor / actuator belongs to a **class** (e.g temperature, co2, shutter ...) that is **appended** to the device's **BASE_TOPIC** (e.g `bu/hall/temperature`), named a **class topic**,
- Each sensor / actuator subscribe to a **command topic** with a **command** token appended to the class topic (e.g `bu/hall/temperature/command`)
- The **end-device** itself publish to a **class topic** (e.g `bu/hall/device`) and subscribe to a **command topic** (e.g `bu/hall/device/command`)
- Each **end-device** is identified by its **MAC_ADDRESS**,
- A **sensor** is either identified by an ID specified at sensOCampus server or automatically discovered at startup (e.g i2c scan). Sensors automatically discovered have an ID prefixed with `auto` (e.g `'auto_C32F'` with last 2 digits being end of device's `MAC_ADDR`),
- An **actuator** is identified by an ID specified at sensOCampus server (e.g `u4/302/shutter` with 3 shutters identified as "front", "center" and "back"),
- topics prefixed with `'_'` are of specific use (e.g `_lora/...` or `_weather/...`),
- For outside end-devices located **within** the campus (e.g toulouse metropole weather station), **BASE_TOPIC** is `outside/<class>` (e.g `outside/ambient/wind` or `outside/access` ---access control gates of our university),
- For abroad end-devices (i.e not located within the campus) **BASE_TOPIC** is `abroad/<location>` e.g `abroad/carcassonne/<building>/<room>/<kind>`
- JSON frames' keys are mostly lower case ;)



Hence, for each device, each sensor and each actuator, there's:

- a **class topic** to publish to → `BASE_TOPIC / CLASS`
- a **command topic** to subscribe to → `BASE_TOPIC / CLASS / command`

msg to a command topic

Whenever a message is sent to a **command topic** (e.g `bu/hall/shutter/command`), the JSON frame **OUGHT** to contain a **'dest'** field.

- `'dest' : "all"` → message is for all of those that subscribed to this command topic,
- `'dest' : "<ID>"` → message is only for those whose ID matches

Example, if you wish to send an order to a specific **device**, `'dest'` will contain its **MAC_ADDRESS**. If you wish to send an order to a shutter, `'dest'` will contain its ID specified at the sensOCampus interface ("front" for example).

✓ Sending order to a shutter (with proper mqtt login / passwd)

```
order: "up"
dest: "all"
    or "<shutter_ID>"
```

Json frame as mqtt payload

u4 / campusfab / shutter / command

In example above, all shutters from u4 / campusfab will receive the "up" order thus opening all of them.

msg to a class topic

Whenever a device, a sensor or an actuator send a message to a **class topic** (e.g bu/hall/noise or u4/cfab/device), associated JSON frame **OUGHT** to contains a 'unitID' field whose value reflect the sender's identity:

- 'unitID' : "<ID>"

| 'unitID' : "00:08:a2:1f:cb:3f" | 'unitID' : 'auto_CB3F' | 'unitID' : "front" |
|---|--|---|
| sender is a device. <i>Note</i> that for compatibility, a 'unit' key with the same value may be added. | sender is an auto-detected sensor usually associated with 'subID' : '<i2c_addr>' <i>Note</i> the last 2 digits are from device's MAC_ADDR | sender may be a sensor or actuator declared at sensOCampus level. <i>Note</i> a 'subID' will be added if it has been declared. |

```
order: "idle"
unitID: "<shutter_ID>"
status: "open"
```

Json frame as mqtt payload

u4 / campusfab / shutter

In the example above, a shutter identified by its 'unitID' sent back its status through the class topic.



REMEMBER: when declaring several devices in the same room (e.g u4/campusfab), it is the users' responsibility to manage identity uniqueness of sensors or actuators declared at the sensOCampus level.

unitID and subID

Whenever a sensor is automatically detected at startup (e.g i2c scan), it gets automatically attributed a 'unitID' (identity) and a 'subID' (informative only field ---e.g i2c addr)

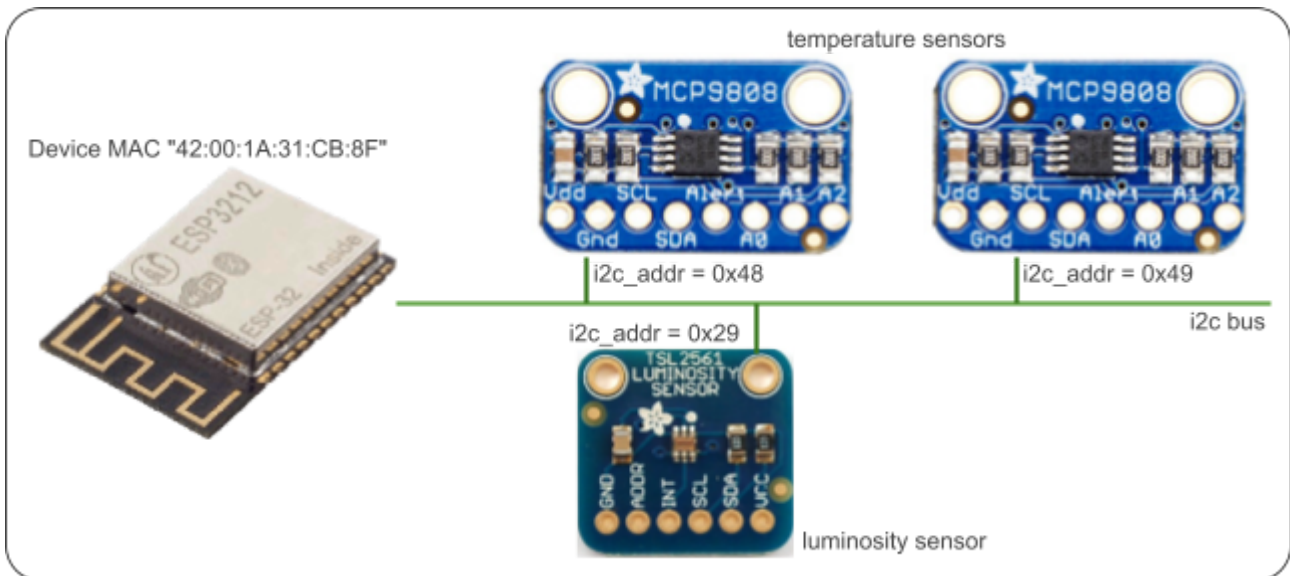
Example:

| unitID | subID | Note |
|------------------------|-------------------|---|
| 'unitID' : 'auto_CB8F' | 'subID' : '32' | At least one auto-detected sensor with i2c addr 0x20. Device's MAC_ADDR end with CB8F |
| 'unitID' : 'inside' | 'subID' : 'ilot1' | A sensor or actuator declared at sensOCampus |

Note: the nature of the sensor will be revealed according to it class topic publishing

scenario example

Considering the following device featuring 3 x i2c sensors.



This will result in the following identity of sensors:

| 'unitID' | 'subID' | class & command topics |
|-----------|---------|--|
| auto_CB8F | 41 | BASE_TOPIC/luminosity BASE_TOPIC/luminosity/command |
| auto_CB8F | 72 | BASE_TOPIC/temperature BASE_TOPIC/temperature/command |
| auto_CB8F | 73 | BASE_TOPIC/temperature BASE_TOPIC/temperature/command |

sensors auto detection and messages publishing

Each sensor value is sent as a separate message. It means for example that if you feature 8 temperature sensors on the same device, you'll have 8 different messages when it comes to pushing the data.

sensors and actuators uniqueness

It is the IoT manager's responsibility to ensure unitID uniqueness at the room-level.

Of course, if you add to the same room two devices whose MAC_ADDR last 2 digits are the same ... use another device ;)

neOCampus MQTT sandbox

To ease testing of your sensor / actuator, you may give a try to the neOCampus MQTT sandbox:

| | |
|-------------------|-------------------------|
| login | test |
| passwd | <ask for it!> |
| server | neocampus.univ-tlse3.fr |
| port | 1883 |
| BASE_TOPIC | TestTopic/# |

Hence, you won't need the sensOCampus credentials and you are free to create / read / write in any topic you want considering it is BASE_TOPIC biased.

example topics



Base : defined at device registration time according to location

e.g u4 / 300 or bu / hall ...

Type : kind of sensor / actuator (module) defined by sensOCampus or automagically detected

e.g shutter, luminosity, temperature, sound, lighting ...

Command : to send orders to a sensor / actuator (module)

e.g orders to shutter like UP, STOP, DOWN

special case topics

| BASE_TOPIC | Description |
|---|--|
| _lora | raw LoRaWAN messages from our LoRaWAN server |
| _weather | Toulouse metropole raw values |
| domoticz | [DEPRECATED] only intended to communicate with domoticz UI |
| TestTopic | neOCampus sandbox |
| outside | <u>outdoor UT3</u> end-devices e.g outside/ambient/rain or outside/access for our access control system |
| 'abroad/<site>' MQTT topic prefix for <u>non UT3 end-devices</u> | |
| abroad | <p><u>non UT3</u> end-devices abroad/<site>/...</p> <p>e.g for indoor sensors at Carcassonne site abroad/carcassonne/<building>/<room>/<kind></p> <p>e.g for outdoor sensors at Carcassonne site abroad/carcassonne/outside/...</p> |

optional 'site' field

Whenever a MQTT message is received **without** a 'site' key in payload nor its topic beginning with 'abroad', it will get considered as an **UT3 site end-device**.

outside/<xxx> topics rules

When it's about out-of-building sensors, we consider them as 'outside'. We present below the various MQTT topics you may encounter :

| Note | 'outside' related MQTT topics @ neOCampus |
|-------------------|--|
| deprecated | <i>case for sensors attached to a conCentratOr</i> u4/302/temperature { 'unitID': 'outside', subID: 'ouest', ... } |
| | <i>case for Toulouse metropole weather station @ UT3 site</i> outside/ambient/rain { 'unitID': 'metropole', ... } |
| | <i>case for abroad out-of-buildings sensors</i> abroad/carcassonne/outside/ambient/rain { 'unitID': 'davis', ... } |
| | <i>case for access control gates for our campus</i> outside/access { 'unitID': 'carGate_main', ... } |

Whenever an equipment is located out-of a building (e.g beehives) we'll try to consider them as being part of a **virtual building**. For example, let's consider our Apiary: all beehives are part of the Apiary, even wherever those beehives are spread all over the campus.

```
e.g apiary/beehive1/temperature  
building=apiary,room=beehive1,kind=temperature,site=ut3
```

Class topics and command topics

Below is a description of the currently existing classes:

| Class | Publish | Subscribe |
|-------------|------------------------|--------------------------------|
| device | BASE_TOPIC/device | BASE_TOPIC/device/command |
| temperature | BASE_TOPIC/temperature | BASE_TOPIC/temperature/command |
| luminosity | BASE_TOPIC/luminosity | BASE_TOPIC/luminosity/command |
| humidity | BASE_TOPIC/humidity | BASE_TOPIC/humidity/command |
| co2 | BASE_TOPIC/co2 | BASE_TOPIC/co2/command |
| energy | BASE_TOPIC/energy | BASE_TOPIC/energy/command |
| camera | BASE_TOPIC/camera | BASE_TOPIC/camera/command |
| digital | BASE_TOPIC/digital | BASE_TOPIC/digital/command |
| noise | BASE_TOPIC/noise | BASE_TOPIC/noise/command |
| weight | BASE_TOPIC/weight | BASE_TOPIC/weight/command |
| uv | BASE_TOPIC/uv | BASE_TOPIC/uv/command |
| rain | BASE_TOPIC/rain | BASE_TOPIC/rain/command |
| wind | BASE_TOPIC/wind | BASE_TOPIC/wind/command |
| attendance | BASE_TOPIC/attendance | BASE_TOPIC/attendance/command |
| airquality | BASE_TOPIC/airquality | BASE_TOPIC/airquality/command |
| | | |
| lighting | BASE_TOPIC/lighting | BASE_TOPIC/lighting/command |
| dali | BASE_TOPIC/lighting | BASE_TOPIC/lighting/command |
| shutter | BASE_TOPIC/shutter | BASE_TOPIC/shutter/command |
| display | BASE_TOPIC/display | BASE_TOPIC/display/command |
| access | BASE_TOPIC/access | BASE_TOPIC/access/command |
| | | |

e.g temperature sensor PUBLISH its value in BASE_TOPIC/temperature

... and it also SUBSCRIBE to BASE_TOPIC/temperature/command to receive orders (e.g frequency acquisition change)

device

Basis of all sensors / actuators, end-devices are connected to a network and are identified via their MAC address.

Each device ought to be able to:

- 'publish' some information (e.g status)
- 'subscribe' to a command topic

publish

| BASE_TOPIC/device | JSON frame |
|-------------------|---|
| status | is automatically published every 30mn (default) <pre>{ 'unitID' : <MAC_ADDR>, 'status': "OK", <optional fields> }</pre> |

Note: there's no 'values' because a device is not supposed to deliver such items.

The 'status' key:

| | |
|-------------|-------------------|
| OK | normal operation |
| FAIL | an error occurred |

Note: since this is only a user informative message, you can send any string you want!

subscribe

| BASE_TOPIC/device/command | JSON frame |
|--------------------------------|--|
| order | <pre>{ 'dest' : <MAC_ADDR>, 'order' : "<action>", <optional fields> }</pre> |
| upgrade (firmware/application) | <pre>{ 'dest' : <MAC_ADDR>, 'order' : "upgrade", <optional fields> }</pre> <p>optional fields may contain - 'value' → url to firmware (e.g 'value' : 'http://xxx.bin')</p> |
| frequency change order | <pre>{ 'dest' : <MAC_ADDR>, 'order' : "frequency", 'value' : <integer seconds> }</pre> |

Note: 'frequency' is about 'status' delivery, not 'values' (whose message does not exists).

'order' command possible actions:

| | |
|------------------|---|
| reset | reset application configuration and restart app. |
| restart | restart application |
| reboot | reboot the whole board |
| update | update application configuration (i.e json config from sensOCampus) |
| upgrade | upgrade firmware / application and restart |
| reinstall | [Raspberry Pi] start whole SDCard reinstallation |
| status | force immediate delivery of a status report to its class topic |
| frequency | change frequency of status report delivery (min. 10mn, max 6h) |



Note that status report is automatically published for each device while it is only published on explicit request for the sensors and actuators.

temperature / luminosity / co2 / humidity / pressure / weight / uv

These classes of sensors send back ambient parameters. They are able to change their acquisition frequency and they transmit both 'value' of the sensor along with its physical unit (e.g 'value_units' : 'celsius' or '%r.H.' or ...)

We describe below the various types of the 'value' field:

| CLASS | 'value' field |
|-------------------------------|---------------|
| temperature, pressure, weight | FLOAT |
| <others> | INT |

publish

| BASE_TOPIC / CLASS | JSON frame |
|--------------------|--|
| status | <p>is published on request</p> <pre>{ 'unitID' : <ID>, 'frequency': <acquisition frequency seconds>, <optional fields> }</pre> <p>optional fields may contain</p> <ul style="list-style-type: none"> - 'sensors' → declared sensors - 'i2c_sensors' → automatically discovered sensors |
| value | <p>is automatically published every <freq> seconds</p> <pre>{ 'unitID' : <ID>, 'value': <value>, 'value_units' : "<string>" <optional fields> }</pre> <p>optional fields may contain</p> <ul style="list-style-type: none"> - 'subID' → either i2c addr of sensor or explicit value set at sensOCampus level |

subscribe

| BASE_TOPIC/CLASS/command | JSON frame |
|--------------------------|--|
| send status order | <pre>{ 'dest' : <ID>, 'order' : "status" }</pre> |

| | |
|-----------------------------|--|
| frequency change order | <pre>{ 'dest' : <ID>, 'order' : "frequency", 'value' : <integer seconds> }</pre> |
| immediate acquisition order | <pre>{ 'dest' : <ID>, 'order' : "acquire" }</pre> |

'order' command possible *actions*:

| | |
|------------------|--|
| status | force immediate delivery of a status report to its class topic |
| frequency | change frequency of status report delivery (min. 10mn, max 6h) |
| acquire | force immediate delivery of sensor value(s). |

rain

This class of sensor sends back a broad range of values and values units (like [energy](#)). Beware that you could face changes in either name of units or number of items in lists (of course both will get consistent anyway).

- 'value' : ['0.0', '0.0', '0.0', '0.0', '0.0', '0.0']
- 'value_units' : ['stormRain_cm', 'dayRain_cm', 'rain24_cm', 'hourRain_cm', 'rainRate_cm_per_hour', 'monthRain_cm']

We describe below the various types of the 'value' field:

| CLASS | 'value' field |
|-------|---------------|
| rain | LIST |

publish

| BASE_TOPIC / rain | JSON frame |
|-------------------|---|
| status | <p><i>is published on request</i></p> <pre>{ 'unitID': <ID>, 'frequency': <acquisition frequency seconds>, <optional fields> }</pre> <p><i>optional fields may contain</i></p> <ul style="list-style-type: none"> - 'sensors' → declared sensors - 'i2c_sensors' → automatically discovered sensors |
| value | <p><i>is automatically published every <freq> seconds</i></p> <pre>{ 'unitID': <ID>, 'value': [<value>, <value> ...], 'value_units': ["<string>", "<string>" ...] <optional fields> }</pre> <p><i>optional fields may contain</i></p> <ul style="list-style-type: none"> - 'subID' → either i2c addr of sensor or explicit value set at sensOCampus level |

subscribe

| BASE_TOPIC/rain/command | JSON frame |
|-------------------------|--|
| send status order | <pre>{ 'dest': <ID>, 'order': "status" }</pre> |

| | |
|-----------------------------|--|
| frequency change order | <pre>{ 'dest' : <ID>, 'order' : "frequency", 'value' : <integer seconds> }</pre> |
| immediate acquisition order | <pre>{ 'dest' : <ID>, 'order' : "acquire" }</pre> |

'order' command possible *actions*:

| | |
|------------------|--|
| status | force immediate delivery of a status report to its class topic |
| frequency | change frequency of status report delivery (min. 10mn, max 6h) |
| acquire | force immediate delivery of sensor value(s). |

wind

This class of sensor sends back a broad range of values and values units (like [energy](#)). Beware that you could face changes in either name of units or number of items in lists (of course both will get consistent anyway).

- 'value' : [326.25, 3.79, 9.66, 315.0]
- 'value_units' : ['windDir', 'windSpeed_kph', 'windGust_kph', 'windGustDir']

Unless otherwise stated, wind directions are degrees (direction where the wind is blowing to) and wind speeds are kilometers per hour.

We describe below the various types of the 'value' field:

| CLASS | 'value' field |
|-------|---------------|
| wind | LIST |

publish

| BASE_TOPIC / wind | JSON frame |
|-------------------|---|
| status | <p>is published on request</p> <pre>{ 'unitID' : <ID>, 'frequency': <acquisition frequency seconds>, <optional fields> }</pre> <p>optional fields may contain</p> <ul style="list-style-type: none"> - 'sensors' → declared sensors - 'i2c_sensors' → automatically discovered sensors |
| value | <p>is automatically published every <freq> seconds</p> <pre>{ 'unitID' : <ID>, 'value': [<value>, <value> ...], 'value_units' : ["<string>", "<string>" ...] <optional fields> }</pre> <p>optional fields may contain</p> <ul style="list-style-type: none"> - 'subID' → either i2c addr of sensor or explicit value set at sensOCampus level |

subscribe

| BASE_TOPIC/wind/command | JSON frame |
|-----------------------------|---|
| send status order | { 'dest' : <ID>, 'order' : "status" } |
| frequency change order | { 'dest' : <ID>, 'order' : "frequency", 'value' : <integer seconds> } |
| immediate acquisition order | { 'dest' : <ID>, 'order' : "acquire" } |

'order' command possible actions:

| | |
|------------------|--|
| status | force immediate delivery of a status report to its class topic |
| frequency | change frequency of status report delivery (min. 10mn, max 6h) |
| acquire | force immediate delivery of sensor value(s). |

energy

Power and energy consumption is usually gathered from Modbus energy meters. This kind of sensor can't be automatically detected, hence requiring a sensOCampus definition.

Moreover, each sensor gives a bunch a data (power, freq, energy, power_factor, intensity, voltage ...) all packed as a list in the 'value' field along with their corresponding units in 'value_units':

- 'value' : ['158426.00', '158420.00', '235.22', '0.14', '20.00', '20.00', '30.00', '0.70']
- 'value_units' : ['Wh', 'Ea+', 'V', 'A', 'W', 'VAR', 'VA', 'cosPhi']

We describe below the various types of the 'value' field:

| CLASS | 'value' field |
|--------|---------------|
| energy | LIST |

publish

| BASE_TOPIC / energy | JSON frame |
|---------------------|--|
| status | <p><i>is published on request</i></p> <pre>{ 'unitID' : <ID>, 'frequency' : <acquisition frequency seconds>, 'backend' : <backend type>, <optional fields> }</pre> <p><i>optional fields may contain</i></p> <ul style="list-style-type: none"> - 'link' : "/dev/usb0" (for example), - 'link_speed' : 9600, - 'nodes' : [(<modbus addr>, <meter_name>), ...] |
| value | <p><i>is automatically published every <freq> seconds</i></p> <pre>{ 'unitID' : <ID>, 'value': [<value>, <value> ...], 'value_units' : ["<string>", "<string>" ...] }</pre> <p><i>Note: see above for a description of 'value' and 'value_units'</i></p> |

<backend type> possible values:

| | |
|----------------|--|
| modbus | either RS-485 or TCP modbus energy meter |
| rf868 | 868MHz energy meter (from consOCampus project) |
| unknown | as you guess ;) |

subscribe

| BASE_TOPIC/energy/command | JSON frame |
|-----------------------------|---|
| send status order | { 'dest' : <ID>, 'order' : "status" } |
| frequency change order | { 'dest' : <ID>, 'order' : "frequency", 'value' : <integer seconds> } |
| immediate acquisition order | { 'dest' : <ID>, 'order' : "acquire" } |

'order' command possible **actions**:

| | |
|------------------|--|
| status | force immediate delivery of a status report to its class topic |
| frequency | change frequency of status report delivery (min. 10mn, max 6h) |
| acquire | force immediate delivery of sensor value(s). |

camera



digital

This class of sensor is related to everything that is relevant to digital inputs (e.g open window detector, motion sensor etc). This kind of sensor ought to get declared at the sensOCampus level.

For each event on a digital input (i.e rising_edge and falling_edge), a message will be sent immediately (i.e no timer involved but direct hardware events management).

We describe below the various types of the 'value' field:

| CLASS | 'value' field |
|---------|---------------|
| digital | INT |

publish

| BASE_TOPIC / digital | JSON frame |
|----------------------|--|
| status | <p><i>is published on request</i></p> <pre>{ 'unitID' : <ID>, 'sensors': [[101, "button", "ilot1"], [102, "presence", "ilot1"]] }</pre> <p><i>'sensors' : [(input, type, subID), ...] these values are coming from sensOCampus definitions</i></p> |
| value | <p><i>is automatically published for each event on input(s)</i></p> <pre>{ 'unitID' : <ID>, 'value': 1, 'input' : 102, 'type' : "presence" or "on_off" or ... 'subID' : "ilot1" }</pre> |

subscribe

| BASE_TOPIC/digital/command | JSON frame |
|----------------------------|--|
| send status order | <pre>{ 'dest' : <ID>, 'order' : "status" }</pre> |

'order' command possible [actions](#):

| | |
|---------------|--|
| status | force immediate delivery of a status report to its class topic |
|---------------|--|

noise

This sensor amplifies sound from a microphone and sets a threshold on a comparator delivering pulses when sound intensity goes beyond. Pulses count are recorded over a sliding window giving their total number for an elapsed time (default 5s). If this total number of pluses is higher than a user defined threshold, then a noise message is sent.

Thus, this sensor is driven by two threshold:

- **sensitivity** → 0 to 100%. Set DAC output to the comparator,
- **threshold** → noise limit (pulses count).

We describe below the various types of the 'value' field:

| CLASS | 'value' field |
|-------|---------------|
| noise | INT |

publish

| BASE_TOPIC / noise | JSON frame |
|--------------------|--|
| status | <p><i>is published on request</i></p> <pre>{ 'unitID' : <ID>, 'sensitivity' : <0 to 100%>, 'threshold' : <integer>, <optional fields> }</pre> <p><i>optional fields may contain</i></p> <ul style="list-style-type: none"> - '_scan_window' → size of sliding window |
| value | <p><i>is automatically published upon noise limit reached</i></p> <pre>{ 'unitID' : <ID>, 'value' : <sum pulses count>, 'value_units' : "pulses" <optional fields> }</pre> <p><i>optional fields may contain</i></p> <ul style="list-style-type: none"> - 'input' → pin used to count pulses - 'subID' → DAC i2c addr |

subscribe

| BASE_TOPIC/noise/command | JSON frame |
|--------------------------|--|
| send status order | <pre>{ 'dest' : <ID>, 'order' : "status" }</pre> |

| | |
|---|---|
| change sensitivity order | { 'dest' : <ID>, 'order' : "sensitivity", 'value' : <integer 0 to 100%> } |
| change noise limit threshold order (i.e pulse count limit) | { 'dest' : <ID>, 'order' : "threshold", 'value' : <integer> } |
| immediate acquisition order | { 'dest' : <ID>, 'order' : "acquire" } |

'order' command possible actions:

| | |
|--------------------|---|
| status | force immediate delivery of a status report to its class topic |
| sensitivity | set new value to DAC output used as input to the comparator |
| threshold | set noise limit through a maximum number of pulses count across the whole sliding windows |
| acquire | force immediate delivery of sensor value(s). |

Note: there's no 'frequency' order because value delivery is not dependent on a timer.

attendance

This class of sensor is related to everything that is relevant to flow of people, bicycles, cars etc. These sensors give an estimation about a flow e.g peoples that are going in or out of a place. This can also be used to simply count a number of people in a room.

We describe below the various types of the 'value' field:

| CLASS | 'value' field |
|------------|---------------|
| attendance | INT |

publish

| BASE_TOPIC / attendance | JSON frame |
|-------------------------|---|
| status | <i>is published on request</i> <pre>{ 'unitID' : <ID> }</pre> |
| value | <i>is automatically published for each event on input(s)</i> <pre>{ 'unitID' : <ID>, 'value' : <int>, 'value_units' : "people" or "car" or "bicycle" or ..., 'subID' : [<optional>] }</pre> |

subscribe

| BASE_TOPIC / attendance / command | JSON frame |
|-----------------------------------|--|
| send status order | <pre>{ 'dest' : <ID>, 'order' : "status" }</pre> |

'order' command possible **actions**:

| | |
|---------------|--|
| status | force immediate delivery of a status report to its class topic |
|---------------|--|

airquality

This class of sensors may send back a broad range of values and values units. However, unlike [energy](#), we'll keep the various values as standalone ones (i.e no lists of values and value_units). Since concentration of most organic components are given as 'ppm' units, we'll make use of the 'subID' field to create a distinguish between the various components:

```
{ "unitId":"i2c_sensor", "subId":"NO", "value":xxx, "value_units":"ppm" }
```

We describe below the various types of the 'value' field:

| CLASS | 'value' field |
|------------|---------------|
| airquality | INT |

publish

| BASE_TOPIC / rain | JSON frame |
|-------------------|--|
| status | <p>is published on request</p> <pre>{ 'unitID' : <ID>, 'frequency': <acquisition frequency seconds>, <optional fields> }</pre> <p>optional fields may contain</p> <ul style="list-style-type: none"> - 'sensors' → declared sensors - 'i2c_sensors' → automatically discovered sensors - others items relevant to your kinds of sensors |
| value | <p>is automatically published every <freq> seconds</p> <pre>{ 'unitID' : <ID>, 'value': <value>, 'value_units' : "ppm", 'subID' : "<string>", <optional fields> }</pre> <p>- 'subID' → either i2c addr of sensor or explicit value set at sensOCampus level</p> |

subscribe

| BASE_TOPIC/rain/command | JSON frame |
|-------------------------|--|
| send status order | <pre>{ 'dest' : <ID>, 'order' : "status" }</pre> |

| | |
|-----------------------------|---|
| frequency change order | { 'dest' : <ID>, 'order' : "frequency", 'value' : <integer seconds> } |
| immediate acquisition order | { 'dest' : <ID>, 'order' : "acquire" } |

'order' command possible *actions*:

| | |
|------------------|--|
| status | force immediate delivery of a status report to its class topic |
| frequency | change frequency of values messages. |
| acquire | force immediate delivery of sensor value(s). |

lighting

Like all actuators, its setup is defined at the sensOCampus level.

This actuator drives various lighting command systems like **telerupteur** or **directly connected** lights sources.

We describe below the various types of the 'value' field:

| CLASS | 'value' field |
|----------|---------------|
| lighting | NONE |

publish

| BASE_TOPIC / lighting | JSON frame |
|-----------------------|---|
| status | <p>is published on request and upon light event (on, off)</p> <pre>{ 'unitID' : <ID>, 'status': '[ON OFF unknown]' }</pre> |

subscribe

| BASE_TOPIC/lighting/command | JSON frame |
|-----------------------------|--|
| order | <pre>{ 'dest' : <ID>, 'order' : '[ON OFF] or status' <optional fields> }</pre> <p>optional fields may contain - 'value': '<0 to 100>' → percentage of luminosity</p> |

'order' command possible **actions**:

| | |
|------------------|---|
| status | force immediate delivery of a status report to its class topic |
| ON or OFF | <p>set output to ON or OFF.</p> <p><i>Note: if teleruptor type, any order will just toggle the output (i.e we don't know whether it is on or off)</i></p> <p><i>Note: for variable lighting systems, an additional 'value' field may contains an integer ranging from 0 to 100 (percents)</i></p> |

dali



shutter

Like all actuators, its setup is defined at the sensOCampus level.

This actuator is able to drive two kinds of shutters (blinds): **wired** and **wireless blinds** (difference is the way outputs are activated ---i.e short pulses for wireless).

We describe below the various types of the 'value' field:

| CLASS | 'value' field |
|---------|---------------|
| shutter | NONE |

publish

| BASE_TOPIC / shutter | JSON frame |
|----------------------|---|
| status | <p>is published on request and upon shutter event</p> <pre>{ 'unitID' : <ID>, 'status' : '[CLOSED OPENED UNKNOWN]', 'order' : '[UP DOWN STOP UNKNOWN]' }</pre> <p>'status' field reflect current state of the shutter 'order' field is the action currently undertaken</p> |

subscribe

| BASE_TOPIC/shutter/command | JSON frame |
|----------------------------|--|
| order | <pre>{ 'dest' : <ID>, 'order' : '[UP DOWN STOP] or status' }</pre> |

'order' command possible **actions**:

| | |
|-----------------------------|---|
| status | force immediate delivery of a status report to its class topic |
| UP DOWN STOP | Set action to open, close or stop shutter in its current position |

display

This kind of actuator is able to display some web pages. Users can send a list of web pages that will get displayed according to a scheduling based on a timer value. This timer value may get changed along with the others parameters like time_on and time_off that define when to switch ON and when to switch OFF the display itself.

We describe below the various types of the 'value' field:

| CLASS | 'value' field |
|---------|---------------|
| display | NONE |

publish

| BASE_TOPIC / display | JSON frame |
|----------------------|--|
| status | <p>is published on request and upon page change and upon order received and upon power mode change</p> <pre>{ 'unitID' : <ID>, 'status' : '[OK KO]', 'pwr_status' : '[ON OFF UNKNOWN FORCEON FORCEOFF]', 'frequency' : '<web pages timeout>', 'time_on' : '<when to set ON>', 'time_off' : '<when to set OFF>', 'days_on' : '<list of active days>', 'cur_url' : '<currently displayed url>', 'def_url' : '<True False>' }</pre> <p>'status' field reflect browser's status (alive/dead) 'pwr_status' field reflect video output status 'frequency' field is the time a web page is displayed 'time_on' field is when we switch ON video (minutes of day) 'time_off' field is when we switch OFF video (minutes of day) 'days_on' list that starts with 1 --> monday 'cur_url' field is the currently displayed web page (will get truncated to a maximum value) 'def_url' tells whether we display the default urls list</p> |

subscribe

| BASE_TOPIC/display/command | JSON frame |
|----------------------------|--|
| send status order | <pre>{ 'dest' : <ID>, 'order' : 'status'</pre> |

| | |
|----------------------------------|--|
| | } } |
| order to set mode | { 'dest': <ID>, 'order': 'mode', 'value': '[NORMAL FORCEON FORCEOFF]' } |
| order to set web pages frequency | { 'dest': <ID>, 'order': 'frequency', 'value': <integer> } |
| order to set URLs list | { 'dest': <ID>, 'order': 'url', 'value': 'url' or [url1, url2, ... urlx] } |
| order to display a TXT msg | { 'dest': <ID>, 'order': 'txt', 'value': '<text to display>' } |
| order to display an image | { 'dest': <ID>, 'order': 'img', 'value': '<binary image to display*>' } <i>* image format will be specified later</i> |
| order to set 'time_on' events | { 'dest': <ID>, 'order': 'time_on', 'value': "06:45" or "1-5 06:45" or "1,3,4 09:00" ... } |
| order to set 'time_off' events | { 'dest': <ID>, 'order': 'time_off', 'value': "19:30" } |

'order' command possible *actions*:

| | |
|---------------|---|
| status | force immediate delivery of a status report to its class topic |
| mode | Set display mode. At startup, we're in NORMAL mode (i.e ON or OFF). If mode is set to FORCEON , the display will switch ON and will then automatically switch back to normal mode on next 'time_on' event. If set to FORCEOFF , the display will stay OFF until the next reboot! or |

| | |
|------------------|---|
| | if you switch back to another mode. |
| frequency | set timeout (seconds) before changing web page to display |
| url(s) | Either set a single url (i.e string) or a list of urls |
| time_on | Kind of 'crontab' value: <ul style="list-style-type: none">- hours:minutes- time of days (e.g 1-5) + hours:minutes e.g: "1-5 7:30" |
| time_off | hours:minutes to switch OFF video output e.g: "19:30" |

access

This topic is dedicated to access control systems. These systems are usually installed on front doors for rooms access security or related to external gates when they're involved in vehicle access authorisation.

- end-device: **publish** to **BASE_TOPIC / access**
- end-device: **subscribe** to **BASE_TOPIC / access / command**

We describe below the various types of the 'value' field:

| CLASS | 'value' field |
|--------|---------------|
| access | NONE |

publish

| BASE_TOPIC / access | JSON frame |
|---------------------|---|
| status | <p>is published upon status order received</p> <pre>{ 'unitID' : <ID>, 'status' : [OK KO], <others> }</pre> <p><others> could be a detected camera, a NFC reader etc. See accessOCampus developer guide.</p> |
| access | <p>is published upon a USER applying to an access</p> <pre>{ 'unitID' : <ID>, 'seq_id' : '<access seq. number>', 'auth_type' : <see accessOCampus doc>, 'nfc_uid' : <see accessOCampus doc>, 'passcode' : <see accessOCampus doc>, 'thermal_detect' : <see accessOCampus doc>, 'image' : <see accessOCampus doc>, }</pre> <p>See accessOCampus developer guide for fields details.</p> |

subscribe

| BASE_TOPIC/access/command | JSON frame |
|---------------------------|--|
| send status order | <pre>{ 'dest' : <ID>, 'order' : 'status' }</pre> |

| | |
|--|---|
| grant access order (i.e door/gate will open) | { 'dest' : <ID>, 'seq_id' : <from access request seq_id field>, 'order' : 'grant' } |
| deny access order (i.e door/gate will remain closed) | { 'dest' : <ID>, 'seq_id' : <from access request seq_id field>, 'order' : 'deny' } |
| ask_code order (people has not been recognized by the camera → ask for code) | { 'dest' : <ID>, 'order' : 'ask_code' } |
| force_open order | { 'dest' : <ID>, 'order' : 'force_open' } |
| force_close order | { 'dest' : <ID>, 'order' : 'force_close' } |
| normal mode order (to cancel previous force_xxx orders) | { 'dest' : <ID>, 'order' : 'normal' } |

'order' command possible actions:

| | |
|--------------------|---|
| status | force immediate delivery of a status report to its class topic |
| grant | subsequently to an 'access request', will authorize access |
| deny | subsequently to an 'access request', will deny access |
| ask_code | people not recognized (camera), ask for an additional code |
| force_open | special_mode: system remains open (e.g journée portes ouvertes) |
| force_close | special_mode: system remains closed (e.g burglar emergency) |
| normal | end of 'special_mode': go back to regular ops |

Annexe - A

A - 1 ESP8266 credentials sample code

```
bool _http_get( const char *url, char *buf, size_t bufsize, const char *login, const char
*passwd ) {

    HTTPClient http;

    http.begin(url);
    log_debug(F("\n[HTTP] GET url : ")); log_debug(url); log_flush();

    // authentication ?
    if( login!=NULL and passwd!=NULL ) {
        http.setAuthorization( login, passwd);
    }

    // perform GET
    int httpCode = http.GET();

    // connexion failed to server ?
    if( httpCode < 0 ) {
        log_error(F("\n[HTTP] connexion error code : ")); log_debug(httpCode,DEC); log_flush();
        return false;
    }

    // check for code 200
    if( httpCode == HTTP_CODE_OK ) {
        String payload = http.getString();
        snprintf( buf, bufsize, "%s", payload.c_str() );
    }
    else {
        log_error(F("\n[HTTP] GET retcode : ")); log_debug(httpCode,DEC); log_flush();
    }

    // close connexion established with server
    http.end();

    yield();

    return ( httpCode == HTTP_CODE_OK );
}
```

```
// HTTP get
bool http_get( const char *url, char buf[], size_t bufsize ) {
    return _http_get( url, buf, bufsize, NULL, NULL );
}

// HTTP get with credentials
bool http_get( const char *url, char *buf, size_t bufsize, const char *login, const char
*passwd ) {
    return _http_get( url, buf, bufsize, login, passwd );
}
```

A - 2 sensOCampus configuration U4/302

- [2020, March] sensOCampus configuration for u4/302 located end-device

```
[
  {
    "topic": "u4/302",
    "modules":
    [
      {
        "module": "Energy",
        "unit": "modbus_rs485",
        "params":
        [
          {
            "param": "frequency",
            "value": 0
          },
          {
            "param": "backend",
            "value": "modbus"
          },
          {
            "param": "link",
            "value": "/dev/ttyUSB0"
          },
          {
            "param": "link_speed",
            "value": 9600
          },
          {
            "param": "subIDs",
            "value": [
              "chauffage",
              "prises1",
              "prises2",
              "prises3"
            ]
          },
          {
            "param": "addr",
            "value": [
              1,
              88,
              65,
              62
            ]
          }
        ]
      },
      {
        "module": "Shutter",
        "unit": "front",
        "params":
        [
          {
            "param": "shutterType",
            "value": "wired"
          },
          {
            "param": "courseTime",
            "value": 20
          },
          {
            "param": "upOutput",
            "value": 100
          },
          {
            "param": "downOutput",
            "value": 101
          }
        ]
      }
    ]
  }
]
```

```

    ],
    {
      "module": "Shutter",
      "unit": "center",
      "params": [
        {
          "param": "shutterType",
          "value": "wired"
        },
        {
          "param": "courseTime",
          "value": 20
        },
        {
          "param": "upOutput",
          "value": 102
        },
        {
          "param": "downOutput",
          "value": 103
        }
      ]
    },
    {
      "module": "Shutter",
      "unit": "back",
      "params": [
        {
          "param": "shutterType",
          "value": "wired"
        },
        {
          "param": "courseTime",
          "value": 20
        },
        {
          "param": "upOutput",
          "value": 104
        },
        {
          "param": "downOutput",
          "value": 105
        }
      ]
    },
    {
      "module": "Digital",
      "unit": "inside",
      "params": [
        {
          "param": "frequency",
          "value": 0
        },
        {
          "param": "subIDs",
          "value": [
            "ilot1",
            "ilot2",
            "ilot3",
            "window"
          ]
        },
        {
          "param": "inputs",
          "value": [
            101,
            106,
            111,
            115
          ]
        }
      ]
    }
  ]
}

```

```

    ],
    {
        "param": "types",
        "value": [
            "presence",
            "presence",
            "presence",
            "open_close"
        ]
    }
],
{
    "module": "Luminosity",
    "unit": "inside",
    "params": [
        {
            "param": "frequency",
            "value": 0
        },
        {
            "param": "subIDs",
            "value": [
                "ilot1",
                "ilot2",
                "ilot3"
            ]
        },
        {
            "param": "inputs",
            "value": [
                100,
                105,
                110
            ]
        },
        {
            "param": "min",
            "value": [
                0,
                0,
                0
            ]
        },
        {
            "param": "max",
            "value": [
                1000,
                1000,
                1000
            ]
        },
        {
            "param": "units",
            "value": [
                "",
                "",
                ""
            ]
        }
    ]
},
{
    "module": "Luminosity",
    "unit": "outside",
    "params": [
        {
            "param": "frequency",
            "value": 60
        },
    ]
}

```

```

        {
            "param": "subIDs",
            "value": [
                "ouest"
            ]
        },
        {
            "param": "inputs",
            "value": [
                119
            ]
        },
        {
            "param": "min",
            "value": [
                0
            ]
        },
        {
            "param": "max",
            "value": [
                1400
            ]
        },
        {
            "param": "units",
            "value": [
                "w/m2"
            ]
        }
    ],
    },
    {
        "module": "Temperature",
        "unit": "inside",
        "params": [
            {
                "param": "frequency",
                "value": 0
            },
            {
                "param": "subIDs",
                "value": [
                    "ilot1",
                    "ilot2",
                    "ilot3"
                ]
            },
            {
                "param": "inputs",
                "value": [
                    103,
                    108,
                    113
                ]
            },
            {
                "param": "min",
                "value": [
                    5,
                    5,
                    5
                ]
            },
            {
                "param": "max",
                "value": [
                    40,
                    40,
                    40
                ]
            },
        ],
    },

```

```

        {
            "param": "units",
            "value": [
                "",
                "",
                ""
            ]
        }
    ],
},
{
    "module": "Temperature",
    "unit": "outside",
    "params":
    [
        {
            "param": "frequency",
            "value": 0
        },
        {
            "param": "subIDs",
            "value": [
                "ouest"
            ]
        },
        {
            "param": "inputs",
            "value": [
                117
            ]
        },
        {
            "param": "min",
            "value": [
                0
            ]
        },
        {
            "param": "max",
            "value": [
                50
            ]
        },
        {
            "param": "units",
            "value": [
                ""
            ]
        }
    ]
},
{
    "module": "Humidity",
    "unit": "inside",
    "params":
    [
        {
            "param": "frequency",
            "value": 0
        },
        {
            "param": "subIDs",
            "value": [
                "i1ot1",
                "i1ot2",
                "i1ot3"
            ]
        },
        {
            "param": "inputs",
            "value": [
                104,
                109,
            ]
        }
    ]
}

```

```

    ],
    {
      "module": "Humidity",
      "unit": "outside",
      "params": [
        {
          "param": "frequency",
          "value": 0
        },
        {
          "param": "subIDs",
          "value": [
            "ouest"
          ]
        },
        {
          "param": "inputs",
          "value": [
            118
          ]
        },
        {
          "param": "min",
          "value": [
            0
          ]
        },
        {
          "param": "max",
          "value": [
            100
          ]
        },
        {
          "param": "units",
          "value": [
            ""
          ]
        }
      ]
    },
    {
      "module": "CO2",
      "unit": "inside",

```



```
    "params":  
    [  
      {  
        "param": "frequency",  
        "value": 0  
      },  
      {  
        "param": "subIDs",  
        "value": [  
          "ilot1",  
          "ilot2",  
          "ilot3"  
        ]  
      },  
      {  
        "param": "inputs",  
        "value": [  
          102,  
          107,  
          112  
        ]  
      },  
      {  
        "param": "min",  
        "value": [  
          0,  
          0,  
          0  
        ]  
      },  
      {  
        "param": "max",  
        "value": [  
          2000,  
          2000,  
          2000  
        ]  
      },  
      {  
        "param": "units",  
        "value": [  
          "",  
          "",  
          ""  
        ]  
      }  
    ]  
  }  
]  
]
```

A - 3 sensOCampus LCC's AirQuality configuration

- [2020, Aug.] sensOCampus configuration for LCC's AirQuality sensor

```
[
  {
    "topic": "irit2/366",
    "modules":
    [
      {
        "module": "airquality",
        "unit": "lcc_sensor",
        "params":
        [
          {
            "param": "frequency",
            "value": 0
          },
          {
            "param": "subIDs",
            "value": [
              "NO2",
              "CO",
              "CH20",
              "NO2_alt"
            ]
          },
          {
            "param": "inputs",
            "value": [
              [ 16, 17, 5, 18, 35 ],
              [ 19, 21, 22, 23, 34 ],
              [ 13, 12, 14, 27, 33 ],
              [ 15, 2, 0, 4, 32 ]
            ]
          },
          {
            "param": "outputs",
            "value": [
              -1,
              -1,
              25,
              26
            ]
          }
        ]
      }
    ]
  }
]
```

Note: 'input' parameters are specified as lists; latest element of the list is the analog input from the amplifier.