

Base de données environnementales

Structuration des données environnementales

Voilà les données renseignées dans la base de données opensensemap

<https://docs.opensensemap.org/#api-Measurements>

A priori la variable physico-chimique mesurée est renseignée dans le champ "phenomenon"

<https://docs.opensensemap.org/#api-Measurements-getDataMulti>

Et chaque "sensor" mesure un "phenomenon" renseigné dans le champ "title"

<https://docs.opensensemap.org/#api-Measurements-getLatestMeasurementOfSensor>

Bref toute la réflexion qu'on pensait devoir avoir sur la structuration des données environnementales mesurées dans une base est dispo dans cette doc'. Y'a qu'à rejoindre le projet SenseBox / OpenSensemap je dirais. Et exploiter les résultats pour valider les modèles de Roland. On peut éventuellement créer notre propre "SenseBox" si les leur ne répondent pas à notre cahier des charges (type et qualité des données valeurs mesurées).

Installation et configuration d'un Serveur MQTT

Installer l'application Mosquitto sur le serveur YunoHost du FabLab fan.ynh.fr

<https://innovation.iha.unistra.fr/books/1-fablab-vos-projets/page/installer-un-serveur-avec-la-distribution-linux-yunohost>

Installer le paquet mosquitto_clients sur un PC (Terminal Linux bash) pour tester le serveur/broker MQTT

<https://shape.host/resources/comment-installer-mosquitto-mqtt-sur-debian-12>

<https://forum.chirpstack.io/t/mosquitto-sub-with-connection-refused-not-authorised/666>

Tester la communication sur le réseau local

Tester la communication sur internet

https://github.com/YunoHost-Apps/mosquitto_ynh/issues/22

Créer une nouvelle station via REST HTTP API

- Types de stations supportées : <https://tutorials.opensensemap.org/category/devices/>
- Ajouter une station supportée ou non : <https://docs.opensensemap.org/#api-Boxes-postNewBox>

A priori l'approche la plus propre serait d'utiliser un type de station existant ou de créer un nouveau type de station "communautaire". Sinon on peut simplement envoyer de la donnée sans créer de type de station, mais c'est plus pour du test.

Créer un nouveau type de station via Sensor.Community

<https://tutorials.opensensemap.org/devices/devices-luftdaten/#3-anpassung-bestehender-ger%C3%A4te>

Publier les données d'un ESP32 vers OpenSenseMap via MQTT

<https://www.elektormagazine.fr/review/afficher-des-donnees-de-capteurs-esp32-sur-la-plateforme-opensensemap>

<https://edu.books.sensebox.de/en/>

Advanced

MQTT ➤

openSenseMap offers a [MQTT](#) client for connecting to public brokers. Documentation for the parameters is provided [in the docs](#). Please note that it's only possible to receive measurements through MQTT.

☐ Enable MQTT

Uri*

Topic*

Message format*
☐ json
☐ csv

Decoding options

Connection options

- On peut envoyer des données vers les serveurs openSenseMap en publiant des messages MQTT au format `.json` (array ou object) ou `.csv` sur un broker (serveur MQTT) public
- La documentation de l'API MQTT se trouve là : <https://docs.opensensemap.org/#api-Boxes-postNewBox>

Paramètres pour une senseBox connectée via MQTT

| Champ | Type | Description |
|---------------|---------|---|
| enabled | Boolean | enable or disable mqtt Valeur par défaut : <code>false</code> |
| url | String | the url to the mqtt server. |
| topic | String | the topic to subscribe to. |
| messageFormat | String | the format the mqtt messages are in. Valeurs autorisées : <code>"json"</code> , <code>"csv"</code> |

| | | |
|-------------------|--------|--|
| decodeOptions | String | a json encoded string with options for decoding the message. 'jsonPath' for 'json' messageFormat. |
| connectionOptions | String | a json encoded string with options to supply to the mqtt client (https://github.com/mqttjs/MQTT.js#client) |

Publication en MQTT

- Le paramètre `messageFormat` spécifie à l'API sous quel format les mesures sont envoyées.
- Un maximum de 2500 valeurs peuvent être envoyées à la fois (Maximum count of values)
- Les formats acceptés sont listés dans [Measurements/Post multiple new Measurements](#) et expliqués ci-dessous

Pour le csv

- Envoyer d'abord un header `content-type: text/csv`
- Puis envoyer une valeur par ligne au format `sensorId, value, [createdAt]` (pas de header, `[createdAt]` est un timestamp optionnel au format RFC 3339)

```
sensorID, value
anotherSensorId, value, RFC 3339- timestamp
sensorIDtheThird, value
anotherSensorId, value, RFC 3339- timestamp, longitude, latitude
anotherSensorId, value, RFC 3339- timestamp, longitude, latitude, height
...
```

Pour le JSON

- JSON Array
 - objects with the keys sensor, value and optionally createdAt and location
 - Specify the header `content-type: application/json`
 - If Location Values are posted, the Timestamp becomes obligatory

```
[
  {"sensor": "sensorID", "value": "value"},
  {"sensor": "anotherSensorId", "value": "value", "createdAt": "RFC 3339- timestamp", "location":
    [lng, lat, height]}
  ...
]
```

```
}
```

- JSON Object :

- the keys of the object are the sensorIds
- the values of the object are either
 - just the `value` of your measurement
 - or an array of the form `[value, createdAt, location]`. `createdAt` and `location` values are optional.

```
{
  "sensorID": "value",
  "anotherSensorID": ["value"]
  "sensorID3": ["value", "createdAt as RFC 3339-timestamp"],
  "sensorID4": ["value", "createdAt as RFC 3339-timestamp", "location latlng-object or
array"],
}
```

Header

| Champ | Type | Description |
|---------------|--------|---|
| Authorization | String | Box' unique access_token. Will be used as authorization token if box has auth enabled (e.g. useAuth: true) |

Paramètres pour la communication des mesures

| Champ | Type | Description |
|----------------------------------|--------|---|
| luftdaten optionnel | String | Specify whatever you want (like <code>luftdaten=1</code> ou <code>luftdaten=true</code>). Signals the api to treat the incoming data as luftdaten.info formatted json. |
| hackair optionnel | String | Specify whatever you want (like <code>hackair=1</code> ou <code>hackair=true</code>). Signals the api to treat the incoming data as hackair formatted json. |
| senseBoxId | String | the ID of the senseBox you are referring to. |

- Standard de format JSON (Object) par luftdaten.info

The API now tries to convert the `JSON objects` in the `sensordatavalues` key to the openSenseMap JSON Array format. Sensors are matched by the key `value_type` against the `title` of the sensors of this box. `SDS_P1` matches sensors with title `PM10`, `SDS_P2` matches sensors with title `PM2.5`. You can find all matchings in the source code of the openSenseMap-API (

`lib/decoding/luftdatenHandler.js`)

```
{
  "sensordatavalues": [
    {
      "value_type": "SDS_P1",
      "value": "5.38"
    },
    {
      "value_type": "SDS_P2",
      "value": "4.98"
    }
  ]
}
```

- Standard de format JSON (Values) par hackAIR

The API now tries to convert the `JSON values` in the `reading` key to the openSenseMap JSON Array format. Sensors are matched by the key `sensor_description` against the `title` of the sensors of this box. `PM2.5_AirPollutantValue` matches sensors with title `PM2.5`, `PM10_AirPollutantValue` matches sensors with title `PM10`. You can find all matchings in the source code of the openSenseMap-API (`lib/decoding/hackAirHandler.js`)

```
{
  "reading": {
    "PM2.5_AirPollutantValue": "7.93",
    "PM10_AirPollutantValue": "32.63"
  },
  "battery": "5.99",
  "tamper": "0",
  "error": "4"
}
```

Formats acceptés pour la localisation

| Champ | Type | Description |
|-------|------|-------------|
|-------|------|-------------|

| | | |
|-------------------------------|--------|--------------------------------|
| lat | Number | Latitude between -90 and 90 |
| lng | Number | Longitude between -180 and 180 |
| height optionnel | Number | Height above ground in meters. |

- Location Object

```
{ "lng": 7.684, "lat": 51.972, "height": 66.6 }
```

- Location Array

```
[ 7.684, 51.972, 66.6 ]
```

senseBox Bytes Format

Submit measurements as raw bytes. Set the header to `content-type: application/sbx-bytes`. Send measurements as 12 byte sensor Id with most significant byte first followed by 4 byte float measurement in little endian (least significant byte first) notation. A valid measurement could look like this:

```
[ 0x59, 0x5f, 0x9a, 0x28, 0x2d, 0xcb, 0xee, 0x77, 0xac, 0x0e, 0x5d, 0xc4, 0x9a, 0x99, 0x89, 0x40 ]
```

but encoded as raw bytes. Multiple measurements are just multiple tuples of id and value. The number of bytes should be a multiple of 16.

senseBox Bytes with Timestamp Format

Submit measurements with timestamp as raw bytes. Set the header to `content-type: application/sbx-bytes-ts`. Send measurements as 12 byte sensor Id with most significant byte first followed by 4 byte float measurement in little endian (least significant byte first) notation followed by a 4 byte uint32_t unix timestamp in little endian (least significant byte first) notation. A valid measurement could look like this:

```
[ 0x59, 0x5f, 0x9a, 0x28, 0x2d, 0xcb, 0xee, 0x77, 0xac, 0x0e, 0x5d, 0xc4, 0x9a, 0x99, 0x89, 0x40, 0x34, 0x0c, 0x60, 0x59 ]
```

but encoded as raw bytes. Multiple measurements are just multiple tuples of id, value and timestamp. The number of bytes should be a multiple of 20.

Ressources

<https://www.urbanheatislands.com/uhi-web-maps>

Projet Capt'air : <https://wp.unil.ch/captographies/>

Revision #11

Created 30 April 2025 13:55:17 by admin_idf

Updated 7 May 2025 14:21:55 by admin_idf