

# openSenseMap

## Dokumentation



---

# Inhaltsverzeichnis

Einleitung	1.1
Registrierung	1.2
Andere Plattformen	1.2.1
Bearbeiten einer Station	1.3
Datendownload	1.4
Datenanalyse	1.5
REST API	1.6

# senseBox



## openSenseMap

Die openSenseMap (OSeM) ist eine Webplattform, auf welcher diverse standortbezogene Sensordaten hochgeladen und visualisiert werden können. Auf der Plattform lassen sich Stationen registrieren, welche die Daten eines oder mehrerer Sensoren übertragen.

Neben einer Zeitreihenvisualisierung der Daten, ist es auch möglich diese nach verschiedenen Kriterien zu Filtern und räumliche Interpolationen zu errechnen.

Sämtliche Sensordaten stehen unter der [Public Domain Dedication and License 1.0](#) zum Download zur Verfügung, und können frei verwendet werden.

Sowohl die openSenseMap als auch die zugehörige API ist Open Source Software. Quellcode und Issuetracker sind hier zu finden:

- [openSenseMap](#)
- [openSenseMap API](#)

## Registrierung auf der OSeM

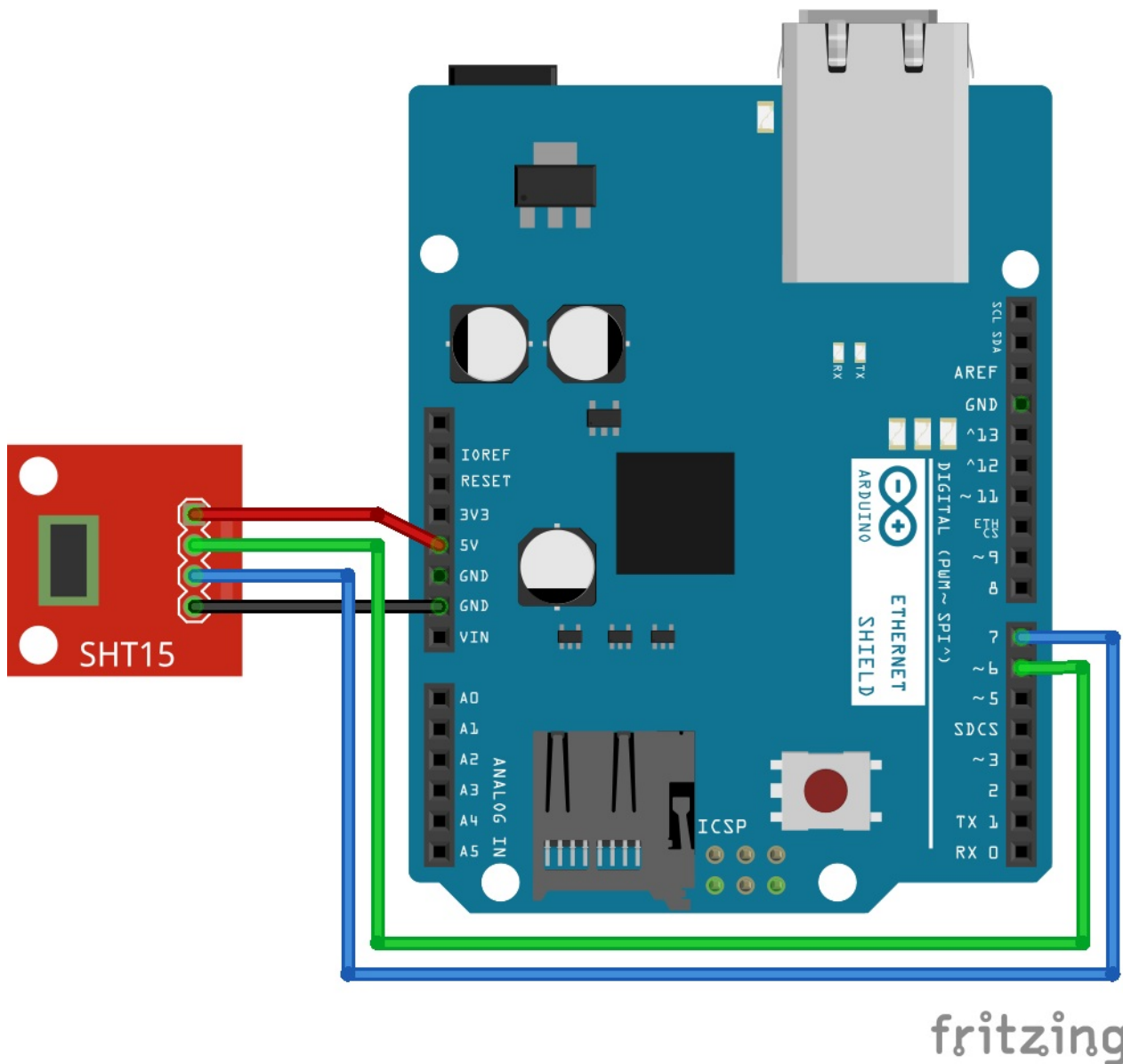
# Manuelle Konfiguration eines Sensors auf der OSeM

In dieser Anleitung wird beispielhaft die Anbindung eines SHT15 Sensors in das OSeM Netzwerk beschrieben. Für die Datenverarbeitung bzw. -übertragung nutzen wir einen Arduino Uno mit Arduino Ethernet Shield. Die REST Schnittstelle bietet aber auch jeder anderen programmierbaren und mit dem Internet verbundenen Messstation die Möglichkeit, Messungen auf der OSeM zu veröffentlichen und zu teilen.

# Materialien

- Arduino Uno R3
- Arduino Ethernet Shield R3
- [Sparkfun SHT15 Breakout](#)

# Aufbau



- VCC zu 5V
- DATA zu Pin 6

- SCK zu Pin 7
- GND zu Arduino GND

## SHT1x Bibliothek

Für Sensoren der SHT1x Serie von Sensirion gibt es bereits eine einfache Arduino-Schnittstelle. [Lade die Bibliothek von Github herunter](#) und entpacke sie in deinen `Arduino/libraries` Ordner. Importiert die Bibliothek wie gehabt in deinen Arduino Sketch, und definiere zusätzlich den Daten- und Taktpin entsprechend der Verkabelung in obiger Abbildung. Danach kannst du eine Verbindung zum Sensor herstellen:

```
#include<sht1x.h>
#define dataPin 6
#define clockPin 7
SHT1x sht1x(dataPin, clockPin);
```

Nun lassen sich über zwei Funktionen die Temperatur in Grad Celsius, sowie die relative Luftfeuchte in Prozent als Gleitkommazahl abspeichern:

```
float temp = sht1x.readTemperatureC();
float humi = sht1x.readHumidity();
```

## Manuelle Registrierung

Um einen Sensor mit der OSeM zu verbinden, musst du ihn [zuerst dort registrieren](#). Falls du dabei keinen der senseBox-Bausätze nutzt, muss in Schritt 4 der Registrierung die manuelle Konfiguration gewählt werden. Wie unten in der Abbildung dargestellt, wird dort für jedes gemessene Phänomen ein neuer Sensor angelegt:

The screenshot shows the 'SenseBox Home' interface. Under 'Manuelle Konfiguration', the 'Manuelle Konfiguration' checkbox is checked. A table lists the configured sensor:

Phänomen	Einheit	Typ	Ändern
Temperatur	°C	SHT15	[Edit] [Delete]
[Dropdown]	[Input]	[Input]	[Add] [Remove]

A dropdown menu is open for the 'Phänomen' column, showing the following options: Temperatur, Luftfeuchtigkeit (highlighted), Luftdruck, Schall, Licht, Licht (digital), UV, and Kamera. Below the table is a '+ Sensor hinzufügen' button.

## openSenseMap API

Eine REST Schnittstelle regelt den Zugang zur Datenbank auf dem OSeM Server. Intern ist jede Messstation mit ihren Sensoren (bzw. Phänomenen) verknüpft, die bei der Registrierung angegeben wurden. In unserem Falle haben wir eine senseBox ID für die Station, sowie jeweils eine Sensor ID für Temperatur- und Luftfeuchtheitsmessungen bei der Registrierung generiert. Die IDs werden dir nach der Registrierung per Mail zugeschickt. Jede Messung wird dann über das HTTP Protokoll mit der `POST` Operation an den Server gesendet. Dazu muss eine eindeutige URI angegeben werden die wie folgt aufgebaut ist:

```
https://api.opensensemap.org/senseBoxID/SensorID
```

Hinweis: Sollte der verwendete Microcontroller nicht HTTPS-kompatibel sein, gibt es derzeit noch eine HTTP Schnittstelle: `http://opensensemap.org:8000/senseBoxID/sensorID`

Jede Messung wird einzeln im JSON Format über das `value` -Attribut an den Server gesendet. Angenommen, wir wollen von unserer Station (ID 1234) einen Messwert des Thermometers (ID abcd) von `22,5` an den OSeM Server schicken, dann sähe der vollständige HTTP POST Request folgendermaßen aus:

```
POST /boxes/1234/abcd HTTP/1.1
Host:opensensemap.org
Content-Type: application/json
Connection: close
Content-Length: 14

{"value":22.5}
```

Achtung: Ab Zeile 7 werden die JSON-Daten gesendet. Der Zeilenumbruch ( `\n` ) in Zeile 6 ist notwendig um die Operation korrekt auszuführen.

## Arduino OSeM Client

Nach der Registrierung wird ein Arduino Sketch generiert, den du als Anhang in einer Bestätigungsmail zugeschickt bekommst. Diesen Sketch musst du noch anpassen, indem die SHT1x Bibliothek eingefügt, sowie die benötigten Variablen und eine Sensorinstanz erstellt werden:

```
#include <SPI.h>
#include <Ethernet.h>

#include<sht1x.h>
#define dataPin 6
#define clockPin 7
SHT1x sht1x(dataPin, clockPin);

//senseBox ID
#define senseBox_ID "1234"
//Sensor IDs
#define TEMPERATURESENSOR_ID "abcd"
#define HUMIDITYSENSOR_ID "efgh"
```

Innerhalb der `if`-Anweisung in der `loop` -Funktion, musst du nacheinander die Sensoren auslesen und mit der Hilfsfunktion `postFloatValue()` hochladen.

```
void loop()
{
  //Upload der Daten mit konstanter Frequenz
  if (millis() - oldTime >= postInterval)
  {
    oldTime = millis();
    temperature = sht1x.readTemperatureC();
    postFloatValue(temperature, 1, temperatureSensorID);
    humidity = sht1x.readHumidity();
    postFloatValue(humidity, 0, humiditySensorID);
  }
}
```

```
}
```

Falls du ein Ethernet Modul nutzt, welches nicht mit der Ethernet Bibliothek kompatibel ist muss der Sketch entsprechend angepasst werden. Solltest du weitere Fragen dazu haben, kannst du dich auch direkt [an unseren Support wenden](#).



# Bearbeiten einer Station

Von bereits registrierten Stationen lassen sich sämtliche Angaben nachträglich ändern.

Hierzu wird der bei der Registrierung erhaltene API-Key benötigt!

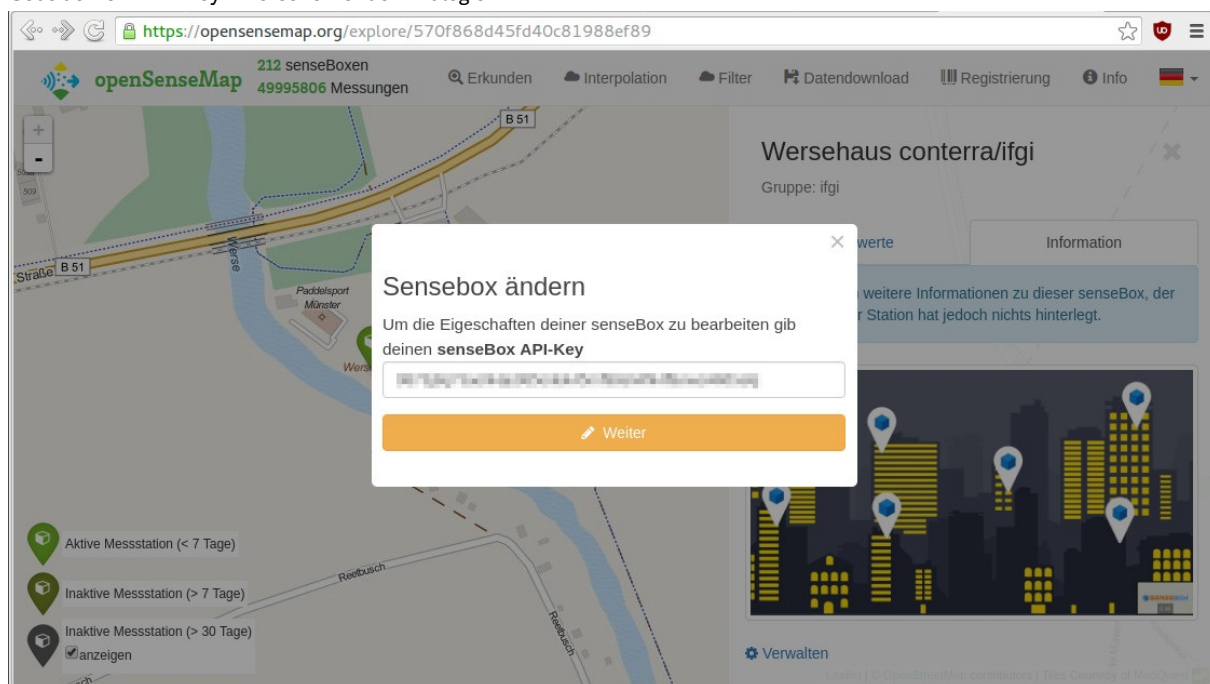
1. Wähle deine Station auf der openSenseMap durch Klick auf den entsprechenden Marker auf der Karte aus.

Alternativ kannst du auch in der folgenden URL `<senseBox-ID>` durch deine senseBox-ID (nicht der API-Key!) ersetzen:

`https://opensensemap.org/explore/<senseBox-ID>`

2. Wähle in der rechten Sidebar den Tab "Information", und klicke unten den Button "Verwalten".

3. Gebe deinen API-Key im erscheinenden Dialog ein.



4. Gebe deine Änderungen im sich öffnenden Formular ein. Neben Änderungen an Metadaten, Standort und Foto ist es auch möglich die Sensor-Konfiguration zu ändern.

Hinweis: Falls du einen neuen Sensor hinzugefügt hast und den aktualisierten Arduino-Sketch herunterladen willst, ist dies erst möglich, wenn die Seite nach dem Speichern neu geladen, und der API-Key erneut eingegeben wurde.

5. Klicke im oberen Teil des Dialogs auf "Speichern" um deine Änderungen zu speichern, oder "Abbrechen" um sie zu verwerfen.

## Löschen einer Station

Folge den Schritten unter "Bearbeiten einer Station" und gebe unter "Sensebox löschen" `DELETE` in das Textfeld ein.

Achtung: Hierdurch werden neben deiner senseBox alle hinterlegten Sensordaten unwiderruflich entfernt!

# Datendownload

Es bestehen mehrere Möglichkeiten Sensordaten von der openSenseMap herunterzuladen. Abhängig von der Fragestellung bietet sich je eine Option an.

## Daten zu einer Box

Unter <https://archive.opensensemap.org> wird ein Archiv für sämtliche Messungen in der openSenseMap Datenbank geführt. Hier sind nach Tag und Box gegliederte Messwerte als CSV beziehungsweise ZIP-Archiv verfügbar.

## Daten zu einem Phänomen

Unter dem Reiter "Datendownload" sind Funktionen zum Herunterladen der Sensordaten zu finden.

Der Datendownload bezieht sich immer auf ein ausgewähltes Phänomen (z.B. Lufttemperatur), einen Zeitraum und eine Boundingbox. Die Boundingbox bezeichnet die räumliche Auswahl der Stationen, und wird automatisch durch den aktuell sichtbaren Kartenausschnitt bestimmt.

Achtung: Je nach Auswahl der Filterparameter kann der Download sehr groß werden (mehrere 100MB)!

## Erweitertes Filtern

Zusätzlich zu den zeitlichen und räumlichen Filtern unter "Datendownload" lässt sich die Stationsauswahl weiter unter dem Reiter "Filter" einschränken. Wie das geht ist im Kapitel [Datenanalyse](#) beschrieben.

## Formate

Derzeit wird nur das Datenformat CSV unterstützt, welches problemlos mit Tabellenkalkulations-Tools wie Excel verarbeitet werden kann.

Jede Zeile enthält eine Messung einer senseBox mit dem ausgewählten Phänomen. Der Messwert ( `value` ), Standort des Sensors ( `lat` , `lng` , Referenzsystem WGS84) und ein Zeitstempel ( `createdAt` ) sind in je einer Spalte angegeben:

```
createdAt;value;lat;lng
2016-09-20T10:05:49.581Z;18.70;7.64568;51.962372
2016-09-20T10:00:52.689Z;18.62;7.64568;51.962372
2016-09-20T09:55:54.282Z;18.47;7.64568;51.962372
....
```

## API-Download

Falls die beiden genannten Möglichkeiten nicht flexibel genug sind, können über die [REST API unter /boxes/data](#) auch komplexe Anfragen gestellt werden.

Für solche anfragen bietet sich das Kommandozeilenwerkzeug `curl` an. Unter Linux ein Terminal öffnen und beispielsweise folgenden Befehl eingeben, um sämtliche Temperatur-Messungen im geographischen Bereich 51°N - 52°N, 7°E - 8°E in die Datei `measurements.csv` herunterzuladen:

```
curl "https://api.opensensemap.org/boxes/data?phenomenon=Temperatur&bbox=7,51,8,52" > measurements.csv
```

Andere geeignete Parameter (Zeitraum, Box-IDs, ...) lassen sich der verlinkten API-Dokumentation entnehmen.



# Datenanalyse

## Filter

Die angezeigten senseBoxen lassen sich nach verschiedenen Kriterien auswählen.

Hierzu können unter dem Reiter "Filter" die entsprechenden Angaben in der Sidebar gemacht werden. Nach einem Klick auf "Filter anwenden" werden die Boxen gefiltert (dies kann je nach Auswahl einen Augenblick dauern).

Anschließend wird unterhalb der Filtereinstellungen eine Auflistung der auf die Kriterien zutreffenden senseBoxen angezeigt.

Dieser Filter bezieht sich auch auf die anderen Datenanalyse-Funktionen [Interpolation](#) und [Datendownload](#)!

## Interpolation

Im Reiter "Interpolation" lassen sich die Daten mehrerer senseBoxen räumlich interpolieren.

Dies ist nützlich um die räumlichen Unterschiede eines Phänomens auf der Karte sichtbar zu machen, oder um ungefähre (!) Werte in Regionen ableiten zu können, in welchen keine Sensoren vorhanden sind.

Vorraussetzung für diese Funktion ist, dass zuvor ein Filter auf einen Zeitraum und ein bestimmtes Mess-Phänomen gewählt wurde.

Es stehen zwei Interpolationsverfahren zur Verfügung: Inverse Distance Weighting (IDW) und Thiessen Polygone.

Nach Einstellung der Parameter (s.U.) wird die Interpolation auf unserem Server berechnet. Wenn die Kalkulation abgeschlossen ist, wird das Ergebnis als Heatmap in der Karte angezeigt.

### Inverse Distance Weighting

TODO

### Thiessen Polygone

TODO

The openSenseMap provides a REST API, which can be used to query & post senseBox metadata & measurements. The endpoint is <https://api.opensensemap.org/>.

This documentation can also be found [here](#) with an improved layout.

## openSenseMap API documentation

methods to manage senseBoxes and get/post measurements

### Boxes

- **method** **GET** [Validate authorization](#)
- **method** **GET** [Get one senseBox](#)
- **method** **GET** [Get all senseBoxes](#)
- **method** **POST** [Post new senseBox](#)
- **method** **PUT** [Update a senseBox: Image and sensor names](#)
- **method** **DELETE** [Delete a senseBox and its measurements](#)
- **method** **GET** [Download the Arduino script for your senseBox](#)

### Interpolation

- **method** **GET** [Get a Inverse Distance Weighting Interpolation as FeatureCollection](#)

### Measurements

- **method** **POST** [Post new measurement](#)
- **method** **GET** [Get latest measurements of a senseBox](#)
- **method** **GET** [Get the 10000 latest measurements for a sensor](#)
- **method** **DELETE** [Delete measurements of a sensor](#)
- **method** **GET,POST** [Get latest measurements for a phenomenon as CSV](#)
- **method** **POST** [Post multiple new measurements](#)

### Misc

- **method** **GET** [Get some statistics about the database](#)

## Boxes

### Validate authorization **method** **GET**

Validate authorization through API key and senseBoxId. Will return status code 403 if invalid, 200 if valid.

```
GET /users/:senseBoxId
```

### Headers

--	--	--

Name	Type	Description
x-apikey	String	the secret API key which corresponds to the <code>senseBoxId</code> parameter.

## Parameter

Name	Type	Description
returnBox	String	optional if supplied and non-empty, returns the senseBox with the senseBoxId with hidden fields
:senseBoxId	String	the ID of the senseBox you are referring to.

## Success 200

Name	Type	Description
Response	json	<code>{"code": "Authorized", "message": "ApiKey is valid"}</code>

## Error Response

Error-Response:

```
HTTP/1.1 403 Forbidden
{"code": "NotAuthorized", "message": "ApiKey is invalid or missing"}
```

## Get one senseBox method GET

GET /boxes/:boxId

## Parameter

Name	Type	Description
format	String	optional the format the sensor data is returned in. Default value: json Allowed values: "json", "geojson"
:senseBoxId	String	the ID of the senseBox you are referring to.

## Success Response

Example data on success:

```
{
  "_id": "57000b8745fd40c8196ad04c",
  "boxType": "fixed",
  "createdAt": "2016-06-02T11:22:51.817Z",
  "exposure": "outdoor",
  "groupTag": "",
  "image": "57000b8745fd40c8196ad04c.png?1466435154159",
```

```
"loc": [
  {
    "geometry": {
      "coordinates": [
        7.64568,
        51.962372
      ],
      "type": "Point"
    },
    "type": "feature"
  }
],
"name": "Oststr/Mauritzsteinpfad",
"sensors": [
  {
    "_id": "57000b8745fd40c8196ad04e",
    "lastMeasurement": {
      "value": "0",
      "createdAt": "2016-11-11T21:22:01.675Z"
    },
    "sensorType": "VEML6070",
    "title": "UV-Intensität",
    "unit": "µW/cm²"
  },
  {
    "_id": "57000b8745fd40c8196ad04f",
    "lastMeasurement": {
      "value": "0",
      "createdAt": "2016-11-11T21:22:01.675Z"
    },
    "sensorType": "TSL45315",
    "title": "Beleuchtungsstärke",
    "unit": "lx"
  },
  {
    "_id": "57000b8745fd40c8196ad050",
    "lastMeasurement": {
      "value": "1019.21",
      "createdAt": "2016-11-11T21:22:01.675Z"
    },
    "sensorType": "BMP280",
    "title": "Luftdruck",
    "unit": "hPa"
  },
  {
    "_id": "57000b8745fd40c8196ad051",
    "lastMeasurement": {
      "value": "99.38",
      "createdAt": "2016-11-11T21:22:01.675Z"
    },
    "sensorType": "HDC1008",
    "title": "rel. Luftfeuchte",
    "unit": "%"
  },
  {
    "_id": "57000b8745fd40c8196ad052",
    "lastMeasurement": {
      "value": "0.21",
      "createdAt": "2016-11-11T21:22:01.675Z"
    },
    "sensorType": "HDC1008",
    "title": "Temperatur",
    "unit": "°C"
  },
  {
    "_id": "576996be6c521810002479dd",
    "sensorType": "WiFi",
    "unit": "dBm",
    "title": "Wifi-Stärke",
    "lastMeasurement": {
      "value": "-66",
      "createdAt": "2016-11-11T21:22:01.675Z"
    }
  }
]
```

```
    }
  },
  {
    "_id": "579f9eae68b4a2120069edc8",
    "sensorType": "VCC",
    "unit": "V",
    "title": "Eingangsspannung",
    "lastMeasurement": {
      "value": "2.73",
      "createdAt": "2016-11-11T21:22:01.675Z"
    },
    "icon": "osem-shock"
  }
],
"updatedAt": "2016-11-11T21:22:01.686Z"
}
```

## Get all senseBoxes method GET

With the optional `date` and `phenomenon` parameters you can find senseBoxes that have submitted data around that time, +/- 4 hours, or specify two dates separated by a comma.

```
GET /boxes?date=:date&phenomenon=:phenomenon&format=:format
```

### Parameter

Name	Type	Description
date	ISO8601Date	optional One or two ISO8601 timestamps at which boxes should provide measurements. Use in combination with <code>phenomenon</code> .
phenomenon	String	optional A sensor phenomenon (determined by sensor name) such as temperature, humidity or UV intensity. Use in combination with <code>date</code> .
format	String	optional the format the sensor data is returned in.  Default value: json Allowed values: "json", "geojson"
exposure	String	optional (optional) only return sensors of boxes with the specified exposure. Can be indoor or outdoor  Allowed values: "indoor", "outdoor"

## Post new senseBox method POST

Create a new senseBox. This method allows you to submit a new senseBox.

Along with the senseBox, an user is created which then owns the senseBox.

If you specify `mqtt` parameters, the openSenseMap API will try to connect to the MQTT broker specified by you. The parameter `messageFormat` tells the API in which format you are sending measurements in.



For `json`, the format is:

```
{  "sensorId": <value>,  "sensorId": [<value>,<createdAt>]  ... }
```

For `csv`, the format is:

```
sensorId,value sensorId,value,createdAt ...
```

POST /boxes

## Headers

Name	Type	Description
content-type	String	Should be <code>application/json</code> or <code>application/json; charset=utf-8</code>

## JSON request body

Name	Type	Description
user	User	the user for this senseBox.
orderId	String	the apiKey of the user for the senseBox.
name	String	the name of this senseBox.
groupTag	String	the groupTag of this senseBox.
exposure	String	the exposure of this senseBox. Allowed values: "indoor","outdoor"
boxType	String	the type of the senseBox. Currently only 'fixed' is supported. Allowed values: "fixed"
sensors	Sensor[]	an array containing the sensors of this senseBox.
loc	Location	the location of this senseBox. Must be a GeoJSON Point Feature. (RFC7946)

## Parameter for creating the user for a senseBox

Name	Type	Description
firstname	String	the firstname of the user.
lastname	String	the ths lastname of the user.
email	String	the email for the user. is used for sending the arduino sketch.
lang	String	the language of the user.

## A single sensor for the nested Sensor parameter

Name	Type	Description
title	String	the title of the phenomenon the sensor observes.
unit	String	the unit of the phenomenon the sensor observes.
sensorType	String	the type of the sensor.
icon	String	optional the visual representation for the openSenseMap of this sensor.

## Settings for a senseBox connected through MQTT

Name	Type	Description
enabled	Boolean	enable or disable mqtt Default value: false
url	String	the url to the mqtt server.
topic	String	the topic to subscribe to.
messageFormat	String	the format the mqtt messages are in. Allowed values: "json", "csv"
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 ( <a href="https://github.com/mqttjs/MQTT.js#client">https://github.com/mqttjs/MQTT.js#client</a> )

## Error Response

Error-Response:

```
HTTP/1.1 415 Unsupported Media Type
{"code": "NotAuthorized", "message": "Unsupported content-type. Try application/json"}
```

## Update a senseBox: Image and sensor names method PUT

Modify the specified senseBox.

```
PUT /boxes/:senseBoxId
```

## Headers

Name	Type	Description
x-apikey	String	the secret API key which corresponds to the <code>senseBoxId</code> parameter.
content-type	String	Should be <code>application/json</code> or <code>application/json; charset=utf-8</code>

## JSON request body

Name	Type	Description
description	String	the updated description of this senseBox.
image	String	the updated image of this senseBox encoded as base64 data uri.
name	String	the name of this senseBox.
grouptag	String	the grouptag of this senseBox.
exposure	String	the exposure of this senseBox. Allowed values: "indoor", "outdoor"
boxType	String	the type of the senseBox. Currently only 'fixed' is supported. Allowed values: "fixed"
sensors	Sensor[]	an array containing the sensors of this senseBox.
loc	Location	the location of this senseBox. Must be a GeoJSON Point Feature. (RFC7946)

## A single sensor for the nested Sensor parameter

Name	Type	Description
title	String	the title of the phenomenon the sensor observes.
unit	String	the unit of the phenomenon the sensor observes.
sensorType	String	the type of the sensor.
icon	String	optional the visual representation for the openSenseMap of this sensor.

## Settings for a senseBox connected through MQTT

Name	Type	Description
enabled	Boolean	enable or disable mqtt Default value: false
url	String	the url to the mqtt server.
topic	String	the topic to subscribe to.
messageFormat	String	the format the mqtt messages are in. Allowed values: "json", "csv"
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 ( <a href="https://github.com/mqttjs/MQTT.js#client">https://github.com/mqttjs/MQTT.js#client</a> )
-------------------	--------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Parameter

Name	Type	Description
:senseBoxId	String	the ID of the senseBox you are referring to.

## Error Response

Error-Response:

```
HTTP/1.1 403 Forbidden
{"code":"NotAuthorized","message":"ApiKey is invalid or missing"}
```

Error-Response:

```
HTTP/1.1 415 Unsupported Media Type
{"code":"NotAuthorized","message":"Unsupported content-type. Try application/json"}
```

## Delete a senseBox and its measurements method DELETE

```
DELETE /boxes/:senseBoxId
```

## Headers

Name	Type	Description
x-apikey	String	the secret API key which corresponds to the <code>senseBoxId</code> parameter.

## Parameter

Name	Type	Description
:senseBoxId	String	the ID of the senseBox you are referring to.

## Error Response

Error-Response:

```
HTTP/1.1 403 Forbidden
{"code":"NotAuthorized","message":"ApiKey is invalid or missing"}
```

## Download the Arduino script for your senseBox method GET

```
GET /boxes/:senseBoxId/script
```

## Headers

Name	Type	Description
x-apikey	String	the secret API key which corresponds to the <code>senseBoxId</code> parameter.

## Parameter

Name	Type	Description
:senseBoxId	String	the ID of the senseBox you are referring to.

## Error Response

Error-Response:

```
HTTP/1.1 403 Forbidden
{"code": "NotAuthorized", "message": "ApiKey is invalid or missing"}
```

# Interpolation ---

## Get a Inverse Distance Weighting Interpolation as FeatureCollection

method GET

Retrieve a JSON object containing

- `breaks` : an array containing equal distance breaks. use `numClasses` parameter to control how many breaks to return
- `featureCollection` : a GeoJSON FeatureCollection with a computed Inverse Distance Interpolation for a certain region of interest and phenomenon.

The properties of each feature in the `featureCollection` is an object with ISO8601 timestamps which are the `timeSteps`. The number of the timesteps can be controlled using the `numTimeSteps` parameter. Values falling inside each timestep are first averaged. Please be aware that requests with  $(\text{areaSquareKilometers} / \text{cellWidth}) > 2500$  will be rejected.

```
GET /statistics/idw?bbox=7.6,51.2,7.8,51.4&phenomenon=Temperatur
```

## Parameter

Name	Type	Description
phenomenon	String	the name of the phenomenon you want to download the data for.
from-date	ISO8601Date	optional Beginning date of measurement data (default: 2 days ago from now)
to-date	ISO8601Date	optional End date of measurement data (default: now)
exposure	String	optional only return sensors of boxes with the specified exposure. Can be indoor or outdoor. Default undecided.  Allowed values: indoor,outdoor

gridType	String	optional The type of the grid for IDW calculation  Default value: hex Allowed values: hex,square,triangle
cellWidth	Number	optional The width of the grid cells in kilometers. Must be positive  Default value: 50
power	Number	optional The power of the IDW calculation  Default value: 1 Allowed values: 1-9
numTimeSteps	Number	optional Return this many timesteps between <code>from-date</code> and <code>to-date</code>  Default value: 6 Allowed values: 1-10
numClasses	Number	optional Number of classes in the breaks array. Must be positive  Default value: 6
bbox	String	A bounding box containing 4 WGS84 coordinates separated by comata (.). Order is longitude, latitude and southwest, northeast.

# Measurements ---

## Post new measurement method POST

Posts a new measurement to a specific sensor of a box.

```
POST /boxes/:senseBoxId/:sensorId
```

### Headers

Name	Type	Description
content-type	String	Should be <code>application/json</code> or <code>application/json; charset=utf-8</code>

### JSON request body

Name	Type	Description
value	String	the measured value of the sensor. Also accepts JSON float numbers.
createdAt	ISO8601Date	optional the timestamp of the measurement. Should be parseable by JavaScript.

### Parameter

Name	Type	Description
:senseBoxId	String	the ID of the senseBox you are referring to.

:sensorId	String	the ID of the sensor you are referring to.
-----------	--------	--------------------------------------------

Error Response

Error-Response:

```
HTTP/1.1 415 Unsupported Media Type
{"code": "NotAuthorized", "message": "Unsupported content-type. Try application/json"}
```

Get latest measurements of a senseBox method GET

Get the latest measurements of all sensors of the specified senseBox.

```
GET /boxes/:senseBoxId/sensors
```

Parameter

Name	Type	Description
:senseBoxId	String	the ID of the senseBox you are referring to.

Get the 10000 latest measurements for a sensor method GET

Get up to 10000 measurements from a sensor for a specific time frame, parameters `from-date` and `to-date` are optional. If not set, the last 48 hours are used. The maximum time frame is 1 month. If `download=true` `Content-disposition` headers will be set. Allows for JSON or CSV format.

```
GET /boxes/:senseBoxId/data/:sensorId?from-date=fromDate&to-date=toDate&download=true&format=json
```

Parameter

Name	Type	Description
from-date	ISO8601Date	optional Beginning date of measurement data (default: 48 hours ago from now)
to-date	ISO8601Date	optional End date of measurement data (default: now)
format	String	optional Can be 'json' (default) or 'csv' (default: json)  Default value: json Allowed values: "json","csv"
download	Boolean	optional if specified, the api will set the <code>content-disposition</code> header thus forcing browsers to download instead of displaying. Is always true for format csv.  Allowed values: "true","false"

:senseBoxId	String	the ID of the senseBox you are referring to.
:sensorId	String	the ID of the sensor you are referring to.
separator	String	optional Only for csv: the separator for csv. Possible values: <code>comma</code> for comma as separator, everything else: semicolon. Per default a semicolon is used. Alternatively you can use <code>delimiter</code> as parameter name.  Allowed values: "comma"

## Delete measurements of a sensor method DELETE

This method allows to delete measurements for the specified sensor. Use the request body to specify which measurements should be deleted.

```
DELETE /boxes/:senseBoxId/:sensorId/measurements
```

### Headers

Name	Type	Description
x-apikey	String	the secret API key which corresponds to the <code>senseBoxId</code> parameter.
content-type	String	Should be <code>application/json</code> or <code>application/json; charset=utf-8</code>

### JSON request body

Name	Type	Description
from-date	ISO8601Date	optional Beginning date of measurement data (no default)
to-date	ISO8601Date	optional End date of measurement data (no default)
timestamps	ISO8601Date[]	optional Allows to specify timestamps which should be deleted
deleteAllMeasurements	Boolean	optional Specify <code>deleteAllMeasurements</code> with a value of <code>true</code> to delete all measurements of this sensor  Default value: <code>false</code> Allowed values: <code>true</code> , <code>false</code>

### Parameter

Name	Type	Description
:senseBoxId	String	the ID of the senseBox you are referring to.
:sensorId	String	the ID of the sensor you are referring to.



## Error Response

Error-Response:

```
HTTP/1.1 403 Forbidden
{"code":"NotAuthorized","message":"ApiKey is invalid or missing"}
```

Error-Response:

```
HTTP/1.1 415 Unsupported Media Type
{"code":"NotAuthorized","message":"Unsupported content-type. Try application/json"}
```

## Get latest measurements for a phenomenon as CSV method GET,POST

Download data of a given phenomenon from multiple selected senseBoxes as CSV

```
GET,POST /boxes/data?boxid=:senseBoxIds&from-date=:fromDate&to-date:toDate&phenomenon=:phenomenon
```

### Parameter

Name	Type	Description
senseBoxIds	String	Comma separated list of senseBox IDs.
phenomenon	String	the name of the phenomenon you want to download the data for.
from-date	ISO8601Date	optional Beginning date of measurement data (default: 2 days ago from now)
to-date	ISO8601Date	optional End date of measurement data (default: now)
columns	String	optional (optional) Comma separated list of columns to export. If omitted, columns createdAt, value, lat, lng are returned. Possible allowed values are createdAt, value, lat, lng, unit, boxId, sensorId, phenomenon, sensorType, boxName, exposure. The columns in the csv are like the order supplied in this parameter  Default value: createdAt,value,lat,lng
exposure	String	optional (optional) only return sensors of boxes with the specified exposure. Can be indoor or outdoor  Allowed values: "indoor","outdoor"
separator	String	optional Only for csv: the separator for csv. Possible values: comma for comma as separator, everything else: semicolon. Per default a semicolon is used. Alternatively you can use delimiter as parameter name.  Allowed values: "comma"
bbox	String	A bounding box containing 4 WGS84 coordinates separated by comata (.). Order is longitude, latitude and southwest, northeast.

## Post multiple new measurements method POST

Post multiple new measurements in multiple formats to a box. Allows the use of csv, json array and json object notation.

CSV:

For data in csv format, first use `content-type: text/csv` as header, then submit multiple values as lines in `sensorId,value,[createdAt]` form. Timestamp is optional. Do not submit a header.

JSON Array:

You can submit your data as array. Your measurements should be objects with the keys `sensor` , `value` and optionally `createdAt` . Specify the header `content-type: application/json` .

JSON Object:

The third form is to encode your measurements in an object. Here, 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]`

For all encodings, the maximum count of values in one request is 2500.

```
POST /boxes/:boxId/data
```

### Parameter

Name	Type	Description
<code>:senseBoxId</code>	String	the ID of the senseBox you are referring to.

# Misc ---

## Get some statistics about the database method GET

returns an array with three numbers which denominates the count of senseBoxes, the count of measurements and the count of measurements in the last minute.

```
GET /stats
```

### Parameter

Name	Type	Description
<code>human</code>	Boolean	optional if true, make numbers easier human readable.  Default value: false Allowed values: true,false

### Success Response

[human=false]

```
[318,118241889,393]
```

[human=true]

```
["318","118M","393"]
```

