Table of Contents

Hilfe & Weiteres

The senseBox:edu Book

Here you will find all information you need to start with your own senseBox:edu.

About this Book

This book guides you through the construction of your own senseBox:edu. You will create your own environmental gaugin station and you will learn a lot about sensors, programming and electrical engineering. Read the introduction carefully. It helps to understand how the book works.

Introduction

This book is is for all people that like to puzzle and to do handicrafts. With this book you will understand technology, hardware and software of senseBox:edu. There are no limits to create your own applications and to write your own programs.

You get the senseBox:edu with a comprehensive set of sensors, LEDs, resistors, cables and more interesting components.

Sensebox:edu can be used in schools, to help pupils understand informatics, physics and technique. But also, it can be used at home to do some enviromental experiments.

This book will help you with your first steps with senseBox:edu. It declares the basis, shows you experiments which you can recreate and declares all components of senseBox:edu.

You can go through this book by using the left sidebar or by using the green arrow keys. Every „side“ has its own heading and below this heading it has a grey deposited box with green scripture, where you can find a short description about the given side.

This looks like this:

Heading

Here you will find a short description about what will happen on this side.

You will also get in contact with our four different info-boxes:

This Box shows you information for example links where you can find more information or background knowledge.

This box is a kind of checking list. All things you have to do are listed here. You can check if you did not forget anything.

> This box gives you warning messages. It shows you where sources of error could be. There you should work with caution.

> This Box shows mistakes. It tells you what to do if you get one of the error messages.

Furthermore there are paragraphs which are not shown directly. They include information that get more into detail. They are folded so that you get a better overview. If you want to see the information they have please click them. You will recognize these paragraphs by their grey box with green scripture. There is also an arrow that tells you if the paragraph is folded or not.

▼ A hidden paragraph

Well done, you got it right! Click the green heading of this paragraph once again and I will disappear!

As a last important element that you should know before you can start with this book, we have so-called "tabs". You probably know it from your internet browser and if you are currently on the internet, you probably have several tabs open. Similar to the browser, we have tabs, but they are built inside the page. They help us to reduce what you see to what you really need. For example, if we describe a tutorial for installing a program, we do it for different operating systems (Windows, OSX (Mac), and Linux). Since you normally only work on one computer, only the operating system installed is of interest to you. Therefore, we integrate the installation instructions in three tabs. Each of these three tabs displays the appropriate operating system instructions. You choose the appropriate tab for your system and get only the appropriate instructions displayed. Just try it out and see what happens when you click the different tabs here.

- First Tab
- Second Tab
- Third Tab

First Tab

I am the first tab. I am opened already, when you load this page.

Second Tab

I am the second tab. By clicking me, you changed the content to the second tab.

Third Tab

I am the third tab. By clicking me, you changed the content to the third tab. Now, have some fun with your senseBox:edu book!

Have you seen all of the tabs and other elements? If so, you are now readz to start off with this book and your senseBox:edu!

Please read the chapter „First steps“ before you start with your own project.

Enjoy your experience with senseBox[1]!

---

[1]. https://sensebox.de/ ↩

Overview

In this book, we will show you what you can do with your senseBox and explain everything you need to know to build a weather station. We also give you tips and suggestions for further experiments.

So that everything runs smoothly, we start from the beginning and guide you step by step through the installation of software and hardware up to the construction of your operational measuring station. First of all, you get an overview of how this book is structured.

1. First steps - Install required programs, test your sensors and start with your senseBox
    i. Installation of Required Software
    ii. Install the Board Support Packages and senseBox Libraries
    iii. Connecting and Cabling MCU and Sensors
    iv. Programming and Initial Tests with your Sensors
2. Basics
3. Projects
4. Components - Look at all parts of the senseBox and their functions]
    i. Content
        i. sensebox MCU
        ii. Bees
        iii. Sensors
        iv. Other Accessories
5. Help – Frequently Asked Questions, Answers and Troubleshooting

    i. FAQ
    ii. Add External Libraries
    iii. Firmware Update Wifi-Bee
    iv. Update Windows USB Bootloader Driver
    v. Updated the Board Support Package & senseBox Libraries

Step 1: Software Installation

Before you can start measuring phenomena, there are a few things to keep in mind. This includes the installation of various drivers and software. Do not worry, it's open source software, so you pay nothing for it. However, you should take a close look at steps 1 to 4 so that no problems occur later.

Installation of Arduino IDE

Before the senseBox can be activated, you need to install drivers and software on your computer. It is also advisable to carry out a test run before commissioning the senseBox, to check whether the sensors are working correctly and that communication with the Internet is running smoothly.

Review the instructions for your operating system and follow the steps given.

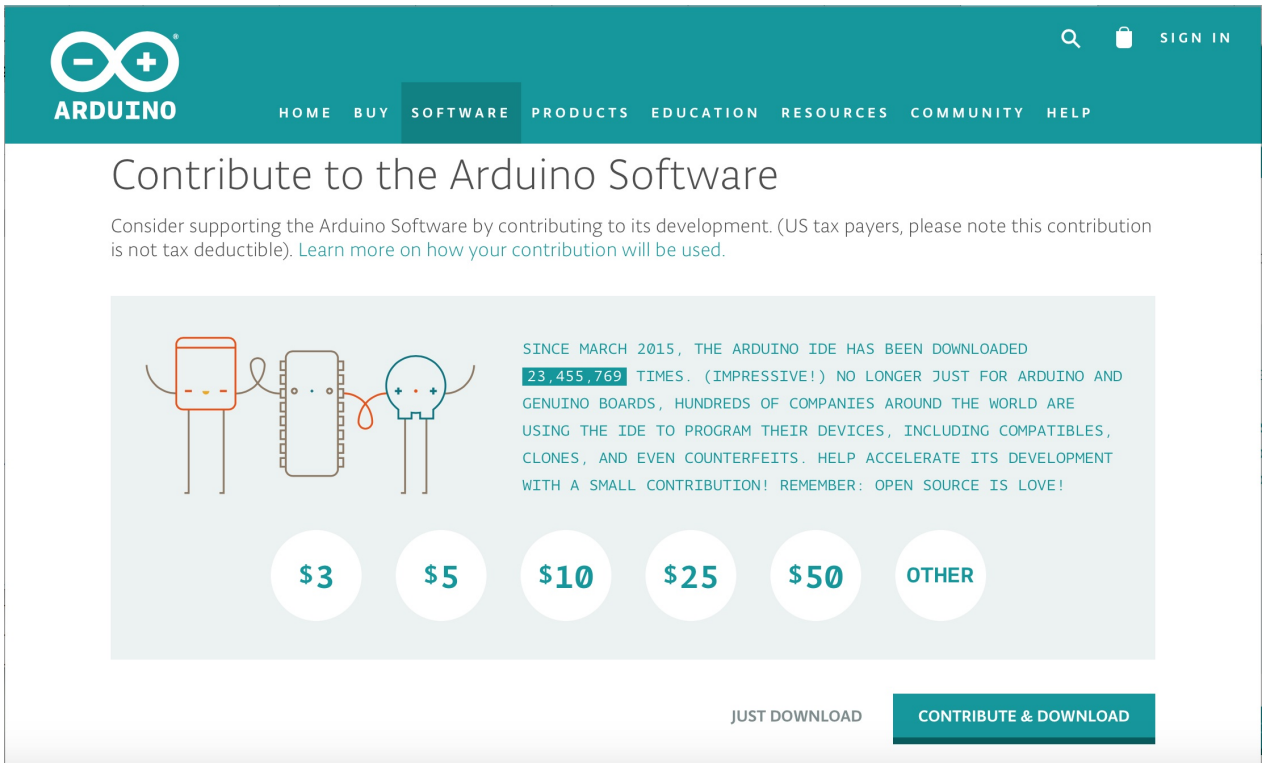
- Windows
- Mac(OSX)
- Linux

Download Arduino Software for Windows

> For a smooth process please use Arduino 1.8.5 or higher

The senseBox is a micro controller with various components and sensors. It is programmed via the development environment Arduino IDE. Download the latest version as a zip file from the Arduino Homepage[321]:

Arduino is an open source project funded by donations. Therefore you will be asked for a donation before downloading; You can skip that by clicking `JUST DOWNLOAD`.



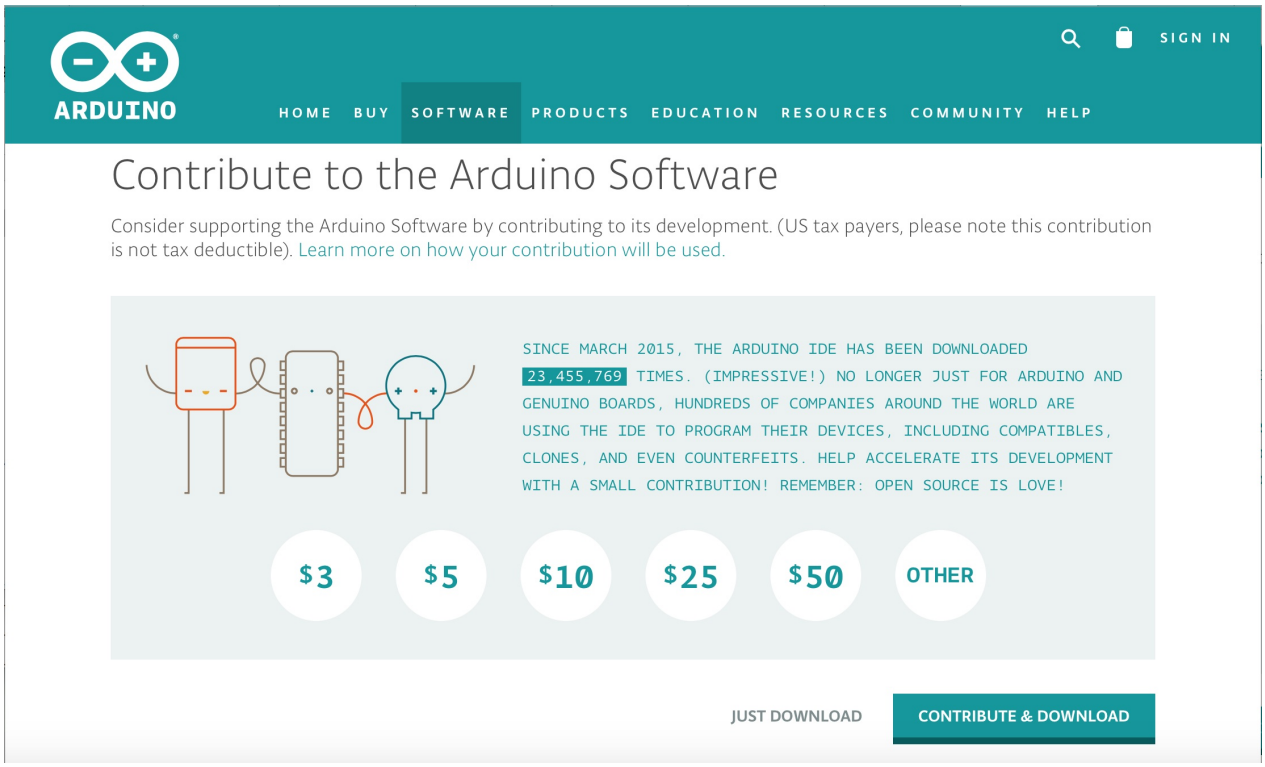
Put a new folder on your hard drive and unzip the zip-file. By starting the file `arduino.exe` the IDE can be started.

Download Arduino Software for Mac(OSX)

> For a smooth process please use Arduino 1.8.5 or higher.

The senseBox is a micro controller with various components and sensors. It is programmed via the development environment Arduino IDE. Download the latest version from the Arduino Homepage:

Arduino is an open source project funded by donations. Therefore you will be asked for a donation before downloading; You can skip that by clicking `JUST DOWNLOAD` .



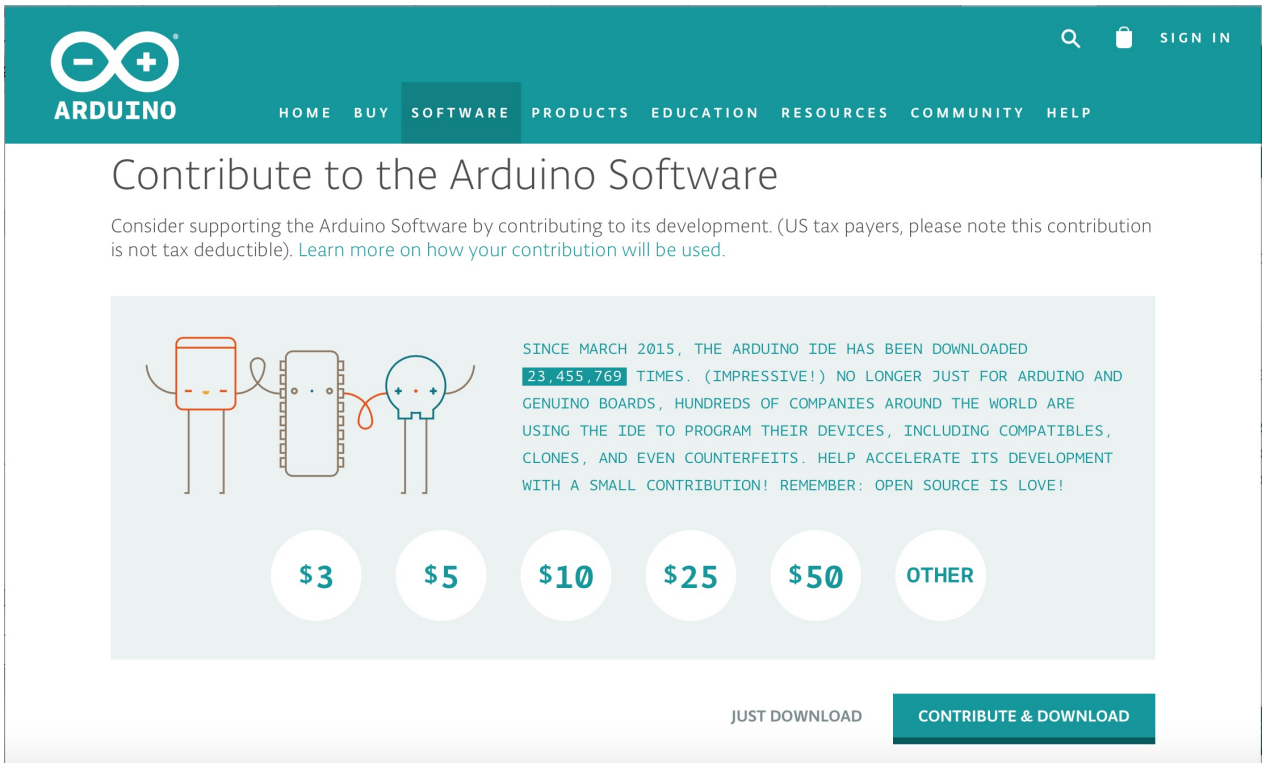
An Arduino.app file should appear in your Downloads folder. Move this file to your "Applications" folder. By starting the file `Arduino.app` the IDE can be started.

Download Arduino Software for Linux

> For a smooth process please use Arduino 1.8.5 or higher.

The senseBox is a micro controller with various components and sensors. It is programmed via the development environment Arduino IDE. Download the latest version from the Arduino Homepage:

Arduino is an open source project funded by donations. Therefore you will be asked for a donation before downloading; You can skip that by clicking `JUST DOWNLOAD`.



Installation of the IDE under Linux

Linux users can download and unpack the Linux version. The included `install.sh` script automatically creates a desktop shortcut. The fastest way to do this is via terminal. Open the terminal by pressing the keys `Ctrl + Alt + T` and enter the following commands:

```
# If the downloaded file is not saved in the Downloads folder, replace "Downloads" with the path to the appropriate f
older
cd downloads
```

```
# Unpack the file with the following command and install Arduino
tar -xvf arduino-1.8.5-linux64.tar.xz
cd arduino-1.8.5
./install.sh
```

> Make sure the command matches the downloaded Arduino version! Load, for example Arduino 1.8.6 also has to stand everywhere where arduino-1.8.5 stands arduino-1.8.6. To check which version you have downloaded, look at the name of the download file.

To program the Arduino, additional rights are required under Ubuntu 14 & 16. These can be set up for the current user with the following commands (benötigt Admin-Rechte):

Run `udevadm monitor --udev` and connect the Arduino via USB to determine the Device ID. The specified name at the end of the output (eg `ttyUSB0`) is the device ID. Exit `udevadm` by `ctrl+C` and execute the following commands, using the device id found:

```
sudo usermod -a -G dialout $(whoami)
sudo chmod a+rw /dev/<device-id>
```

After a logout and login again, the Arduino from the Arduino IDE should be programmable!

---

1. https://www.arduino.cc/en/main/software ↩
2. https://www.arduino.cc/en/main/software ↩
3. https://www.arduino.cc/en/main/software ↩

Step 2: Install Board Support Package

In order for the Arduino IDE to support your senseBox MCU uand allow you to transfer programs to it, you will need to install two board support packages before you begin. These contain the necessary drivers and the necessary software to communicate with your processor. The board support package of the senseBox also already contains our senseBox libraries. This will give you all the basic methods for programming the enclosed sensors.

Libraries

For the programming of the senseBox, the senseBox libraries should be included at the beginning. We have integrated these libraries into the board support package of the senseBox to make the installation as easy as possible.

▼ 'Library' - What is a library and why do I need it?

As the name suggests, a library is a collection of something - a collection of methods to be more specific. Methods are programming smaller sections of code that can be applied to an object. For example, with the senseBox, a method can be invoked to turn the LEDs on and off on the MCU. There are a lot of such standard methods that are used by a variety of programs. In order not to have to transfer these methods individually into the program code, they can be stored in libraries. So a library is a file that stores many methods. You can include libraries in your code. For this it is enough if they are stored in the Arduino folder for libraries and then they are integrated with a single line at the beginning of the program code. This looks like this in Arduino for the library named „senseBoxIO“:

```
#include <senseBoxMCU.h>;
```

If the library is included, all methods contained in it can be used in the code.

Include Board Support Package

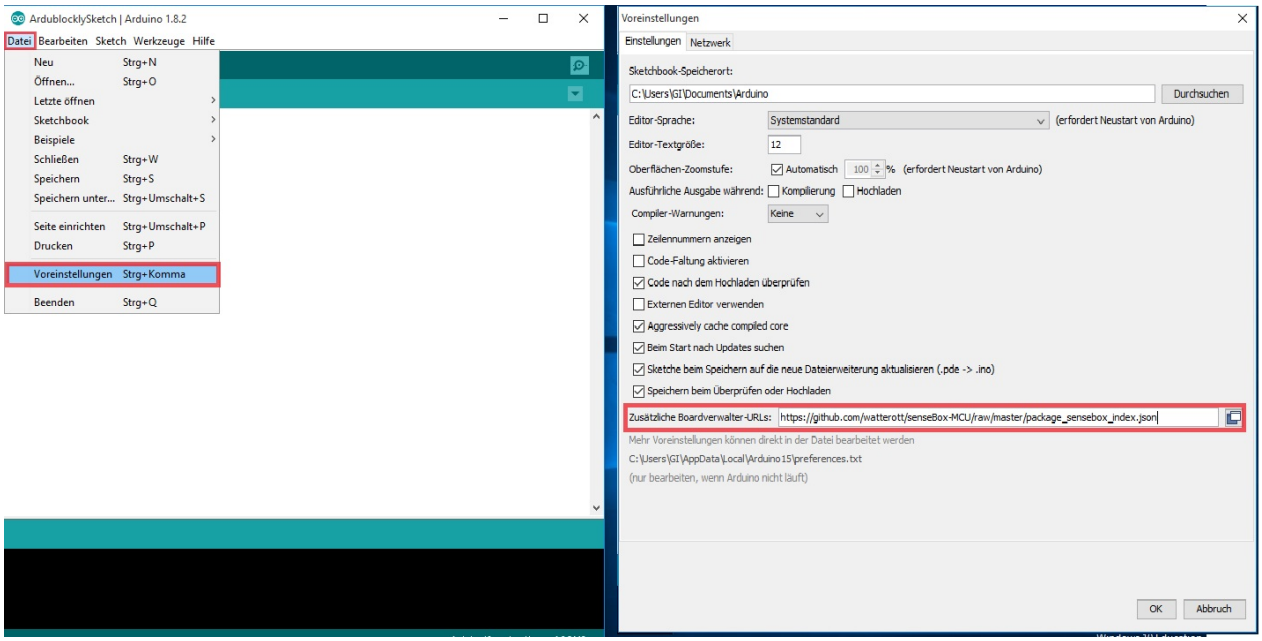
Choose your operating system to see the appropriate instructions:

- Windows
- Mac(OSX)
- Linux

Instructions for Windows

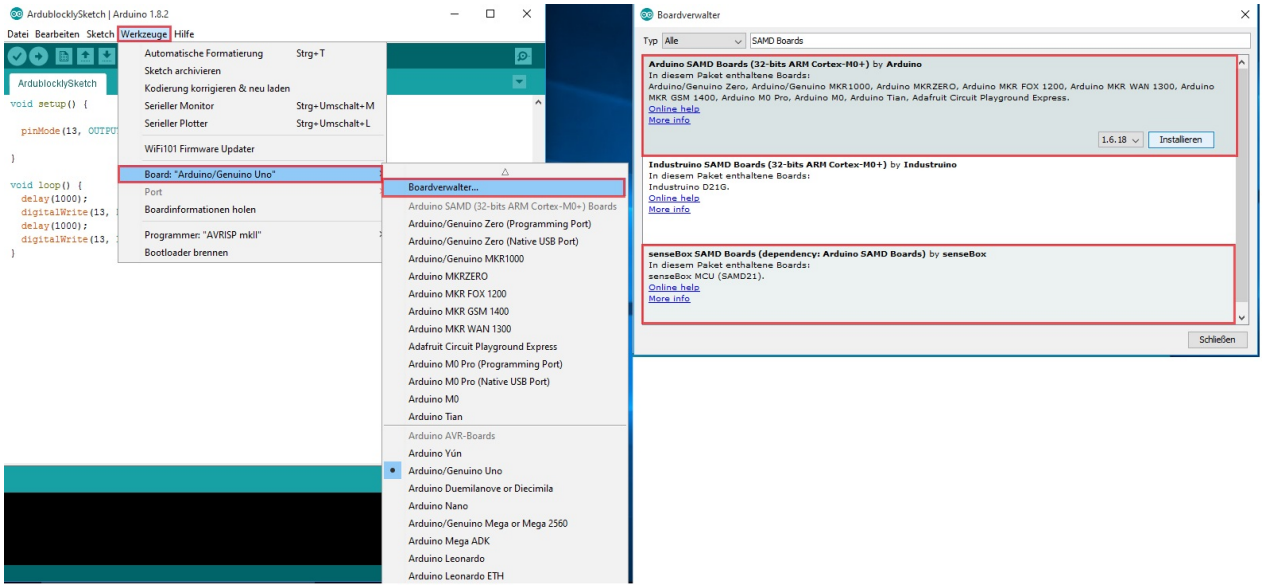
Add the following URL in your Ardunino IDE under File -> Preferences to the Additional Board Administrator URLs box

```
https://github.com/sensebox/senseBoxMCU-core/raw/master/package_sensebox_index.json
```



Open the preferences and paste the URL

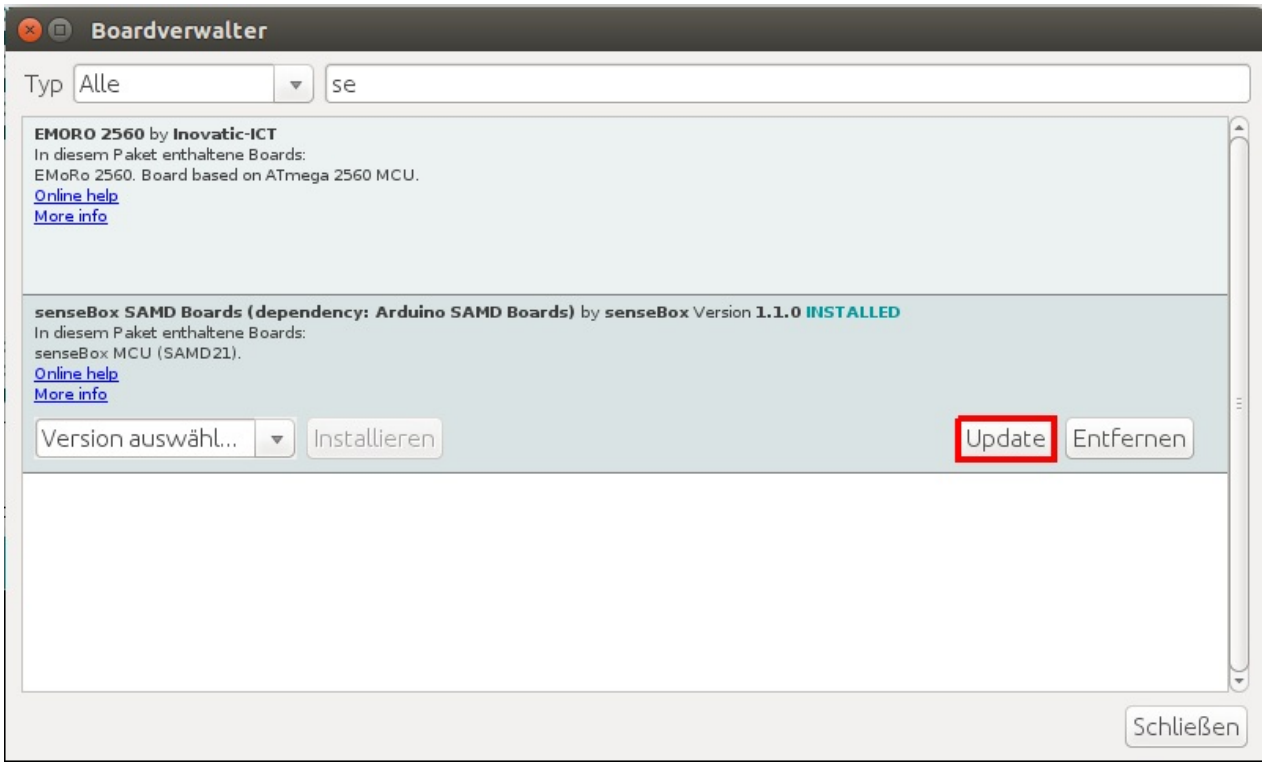
Then open the board administrator under Tools -> Board:"..." -> board administrator and install there the two board support packages named Arduino SAMD Boards by Arduino and senseBox SAMD Boards by senseBox.



Open the board administrator and install the two packages

> Enter "SAMD" at the top of the search bar to find the packages faster

Since we regularly update the senseBox SAMD Boards-Package for you, you should always go back to the board administrator and check if the senseBox SAMD Boards-Package is still up-to-date. As described above, open the board administrator and search for senseBox SAMD Boards. If you click on the entry in the list, there appears, in the case of a new version, an update button. Click this to install the latest version.



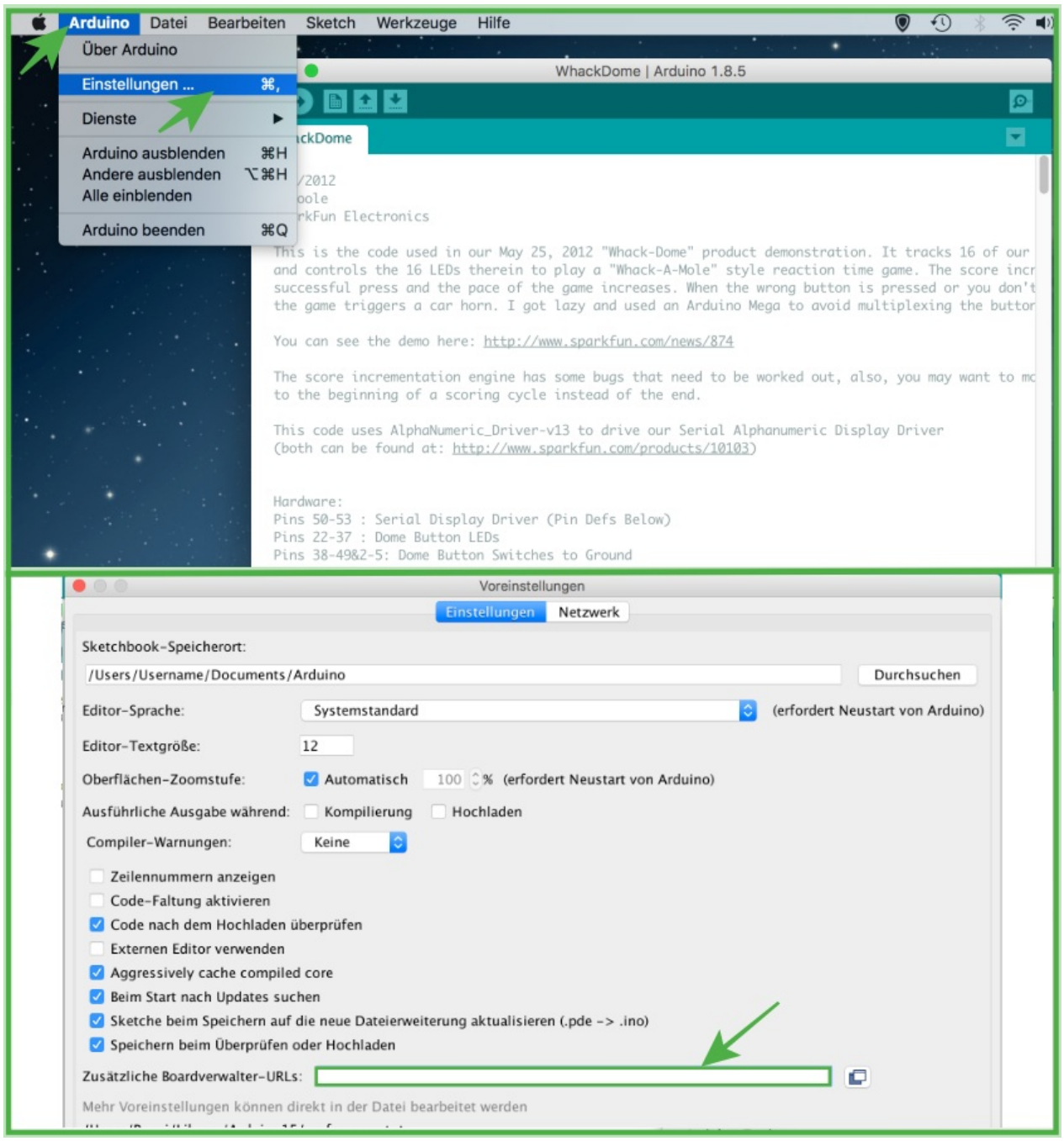
Click 'Update' to update the board support package

> It is important to click on the entry first. Otherwise, the update button is not displayed, even if there is already a new version.

Instructions for Mac

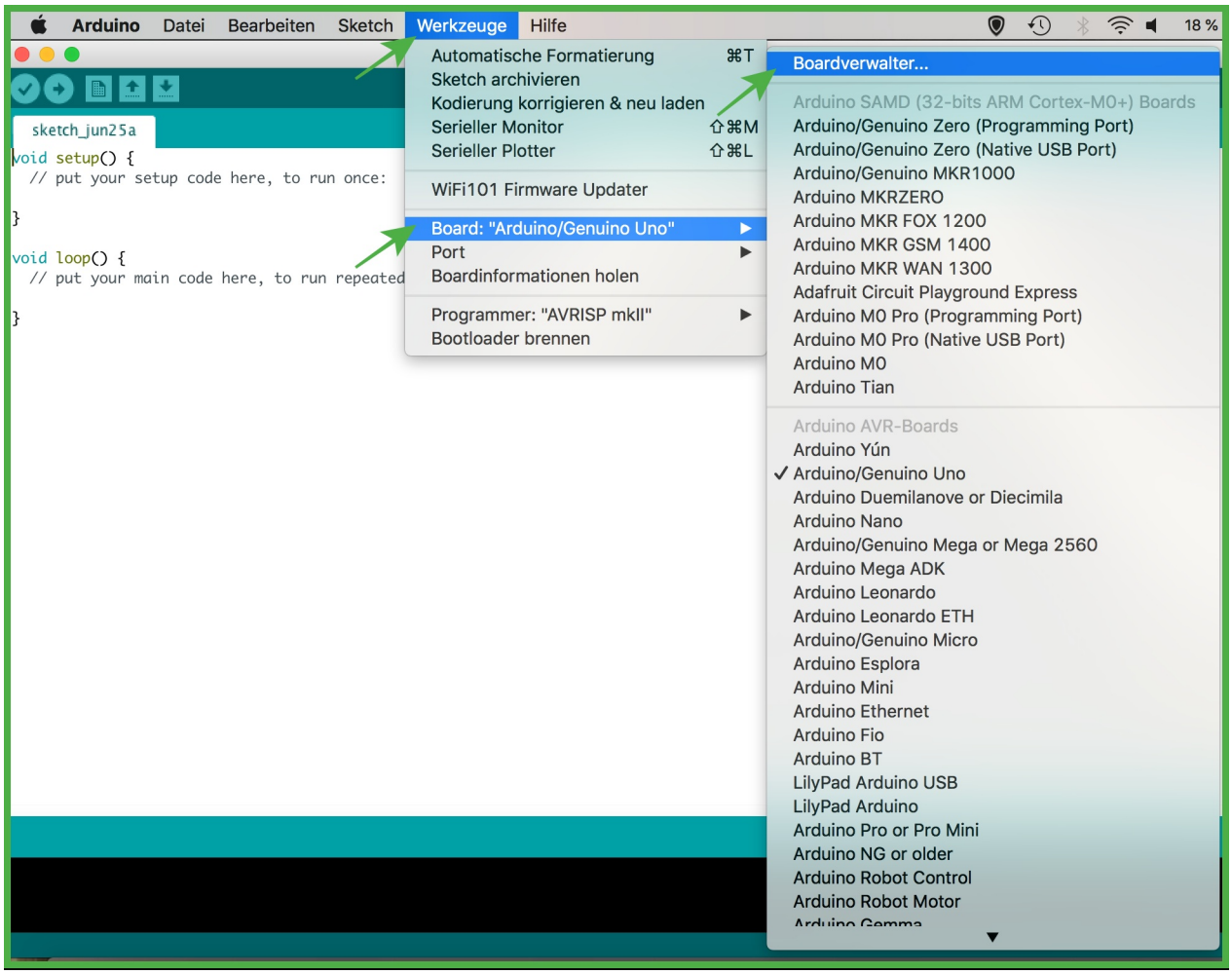
Add the following URL in your Arduino IDE under `Arduino -> Preferences...` to the to the Additional Board Administrator URLs box:

```
https://github.com/sensebox/senseBoxMCU-core/raw/master/package_sensebox_index.json
```



Open the preferences and paste the URL

Then open the board administrator under Tools -> Board:"..." -> Board Administrator and install there the two board support packages named Arduino SAMD Boards by Arduino and senseBox SAMD Boards by senseBox.

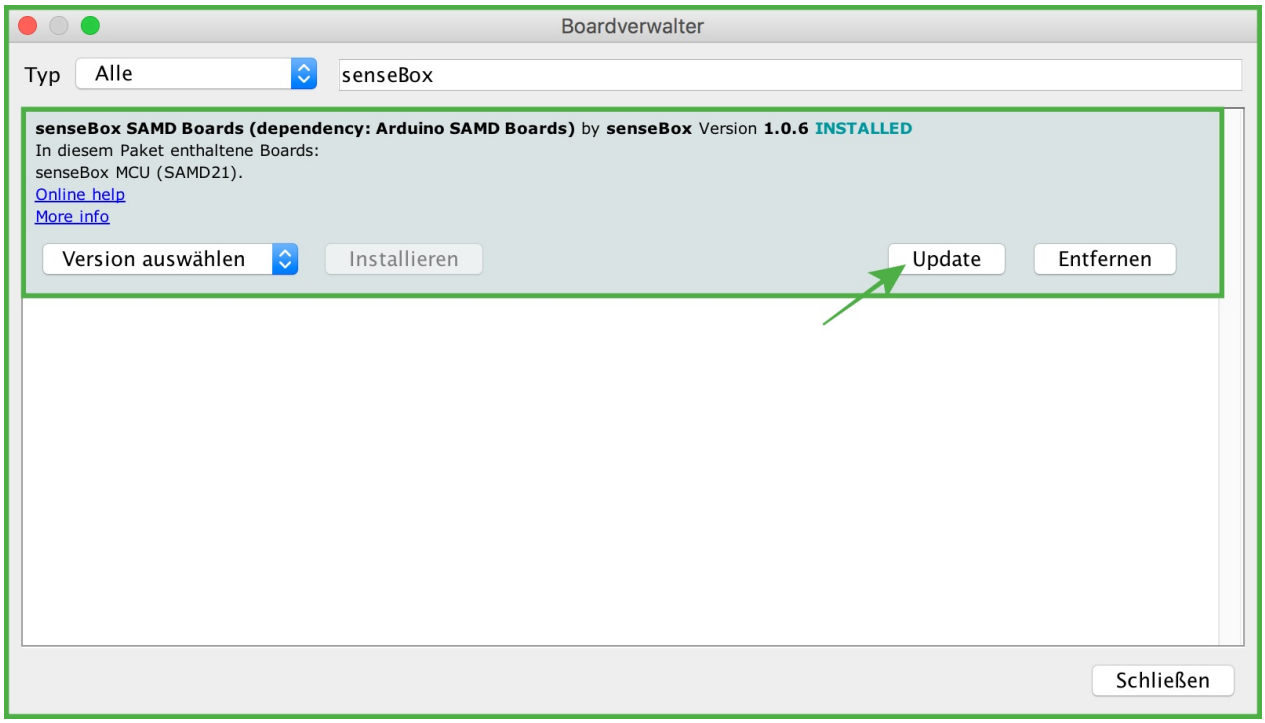


Open the board administrator

Install the two marked packages

> Enter "SAMD" at the top of the search bar to find the packages faster

Since we regularly update the senseBox SAMD Boards-Package for you, you should always go back to the board administrator and check if the senseBox SAMD Boards-Package is still up-to-date. As described above, open the board administrator and search for senseBox SAMD Boards. If you click on the entry in the list, there appears, in the case of a new version, an update button. Click this to install the latest version.



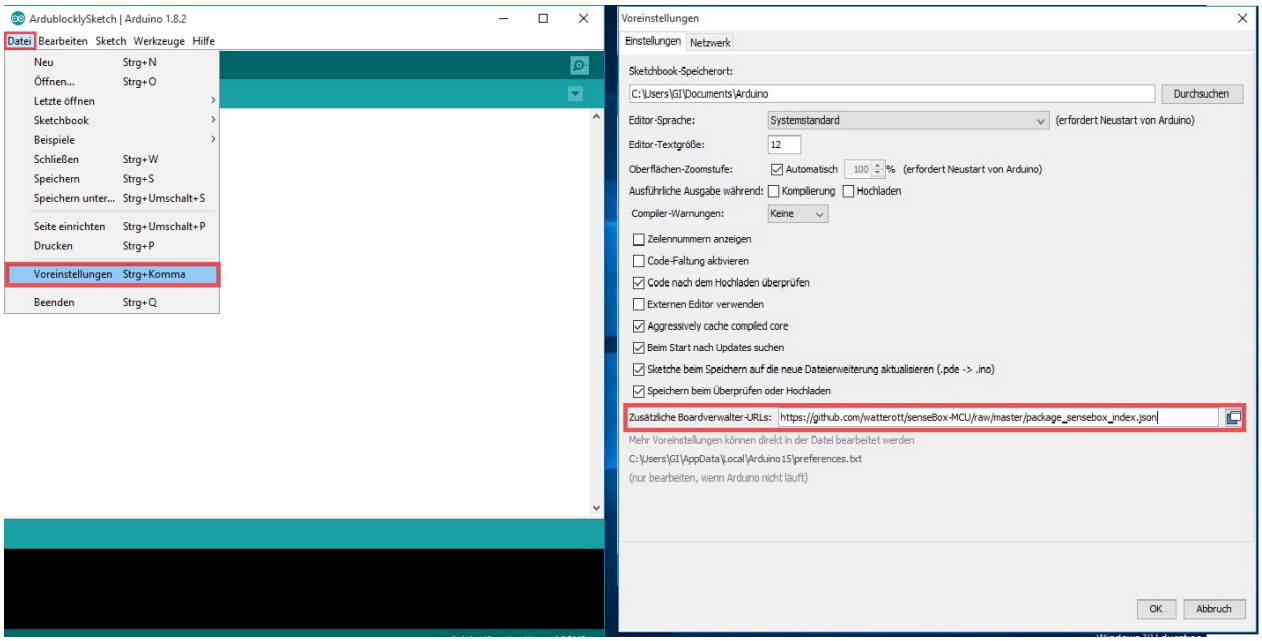
Click 'Update' to update the board support package

> It is important to click on the entry first. Otherwise, the update button is not displayed, even if there is already a new version.

Instructions for Linux

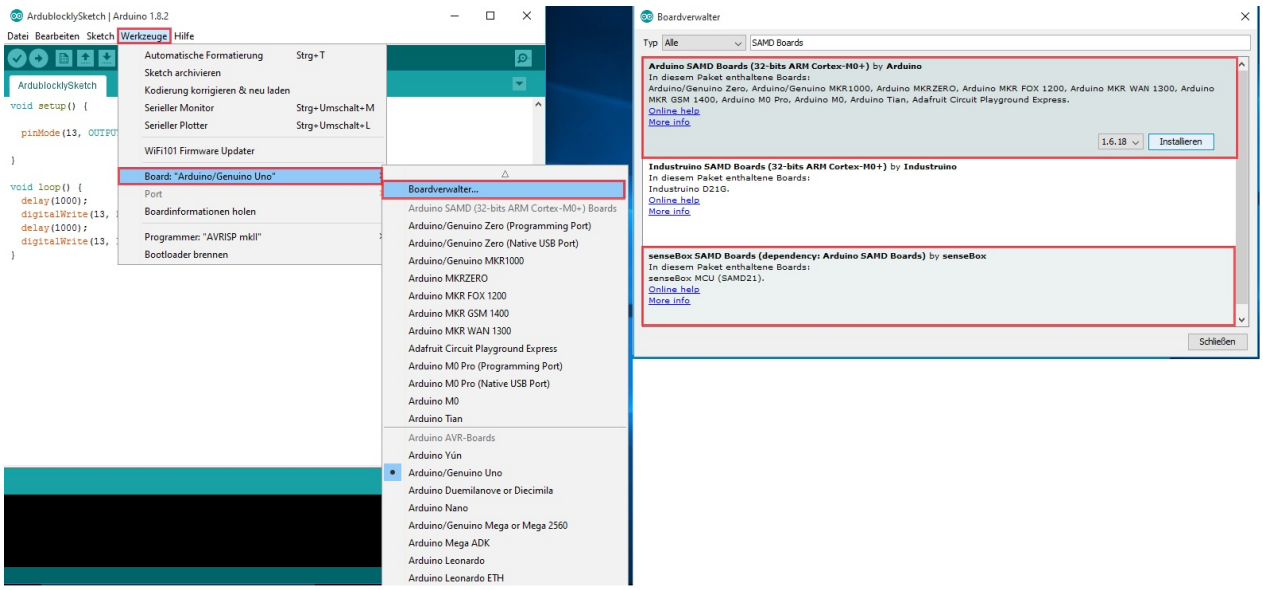
Add the following URL in your Arduino IDE under File -> Preferences to the Additional Board Administrator URLs box:

```
https://github.com/sensebox/senseBoxMCU-core/raw/master/package_sensebox_index.json
```



Open the preferences and paste the URL

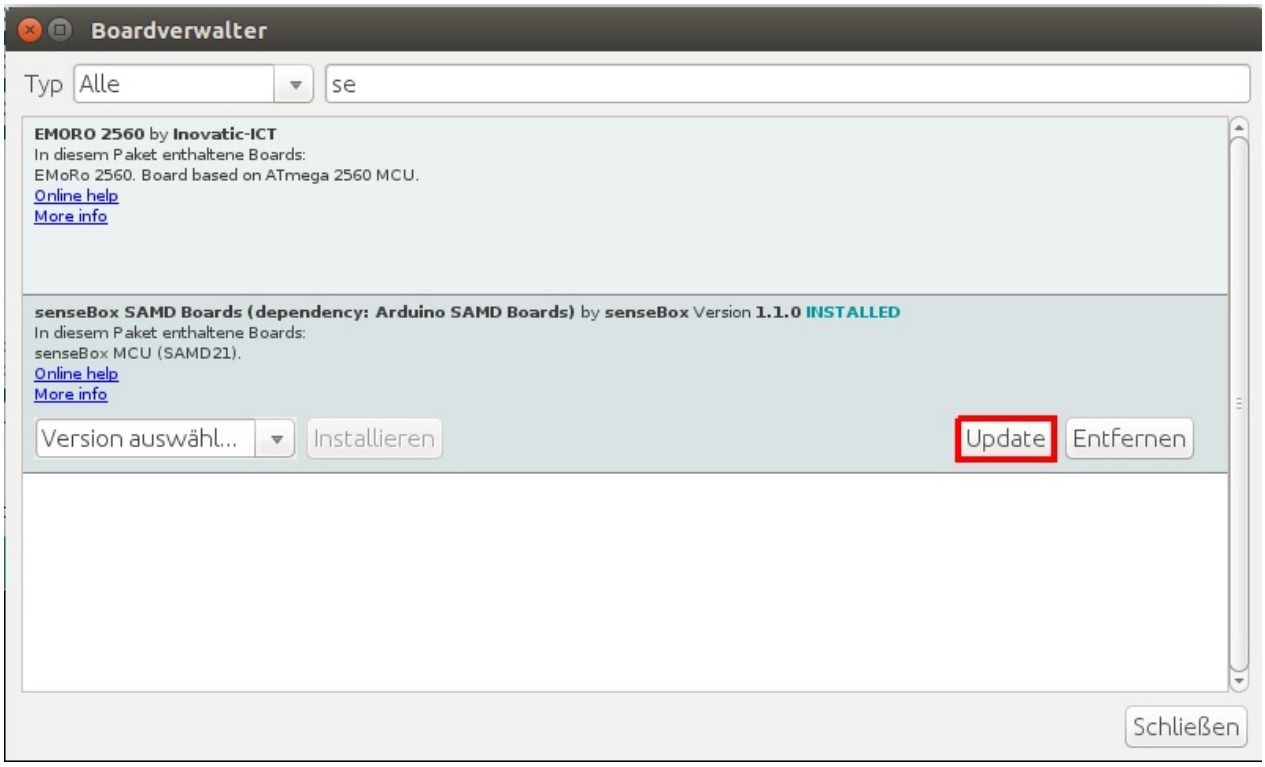
Then open the Boardadministrator under Tools -> Board:"..." -> Boardadministrator und installiere dort die zwei and install there the two board support packages named Arduino SAMD Boards by Arduino and senseBox SAMD Boards by senseBox.



Open the board administrator and install the two packages

> Enter "SAMD" at the top of the search bar to find the packages faster

Since we regularly update the senseBox SAMD Boards-Package for you, you should always go back to the board administrator and check if the senseBox SAMD Boards-Package is still up-to-date. As described above, open the board administrator and search for senseBox SAMD Boards. If you click on the entry in the list, there appears, in the case of a new version, an update button. Click this to install the latest version.



Click 'Update' to update the board support package

> It is important to click on the entry first. Otherwise, the update button is not displayed, even if there is already a new version.

Step 3: Connecting and Wiring

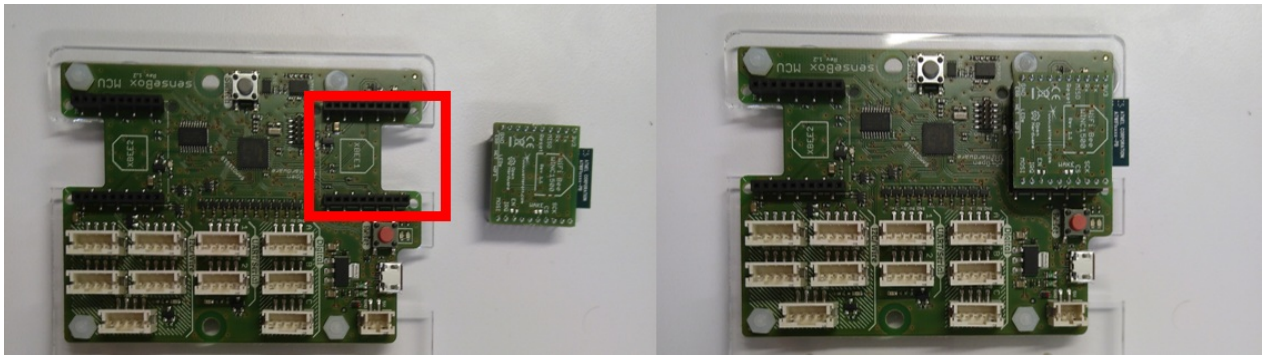
The I2C connector system makes it very easy to connect the sensors and components to the senseBox MCU. Here you can see again a rough overview of the individual components

Connection of Bees

The connection of the Bees is very simple. Through the plug-in system, it is sufficient to put the Bee directly on the microcontroller. There are only two things to consider: 1. The orientation on the board and 2. The correct port connection on the microcontroller.

WiFi Bee, Ethernet Bee and LoRa Bee

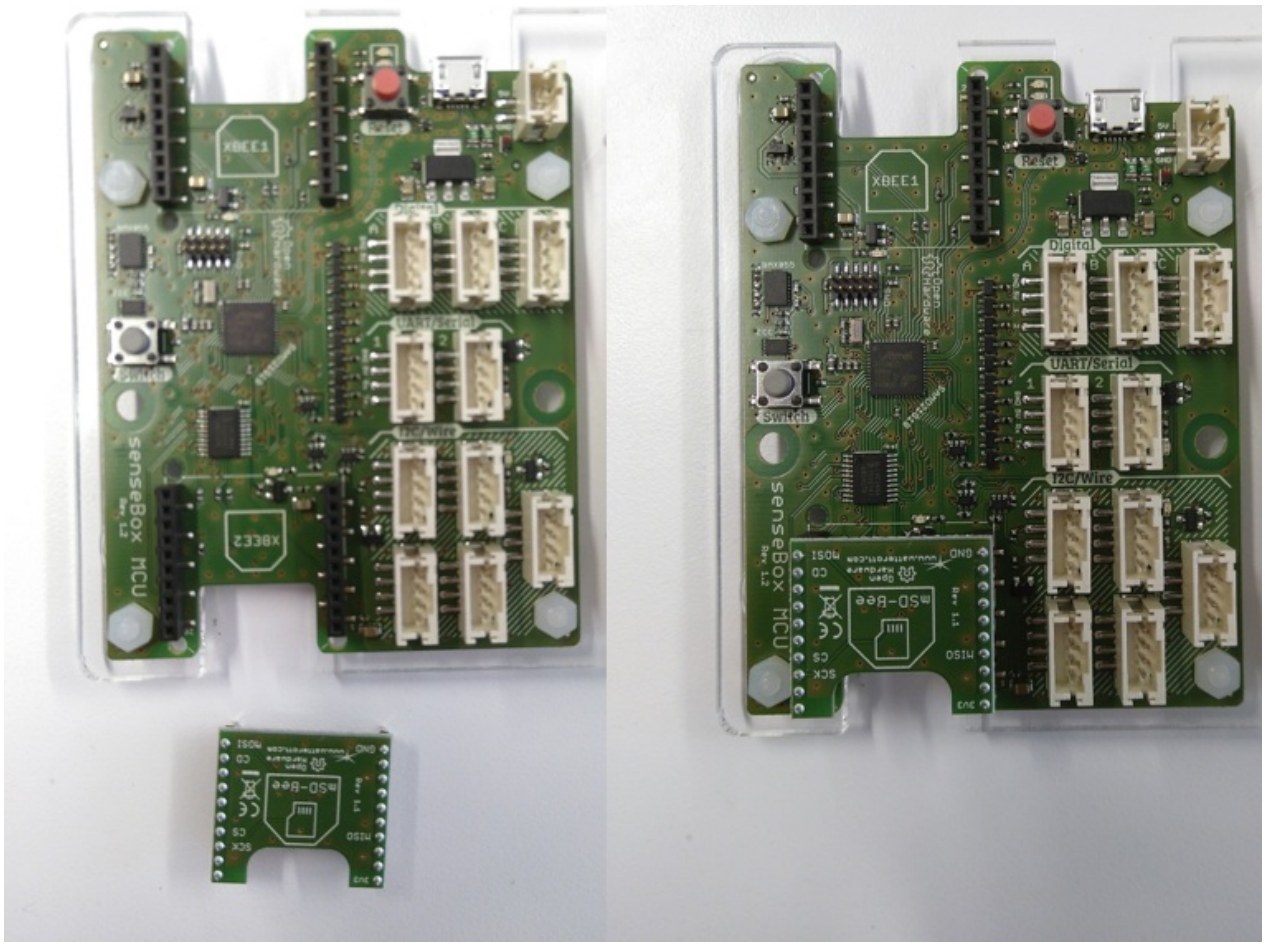
These Bees are all attached to port 1. The correct port can be recognized by the caption: `XBEE1` . The correct direction when plugging you can tell by the 7-sided marking on the board and the Bee.



Exemplary connection of the WiFi Bee to the MCU (XBEE1)

mSD-Bee

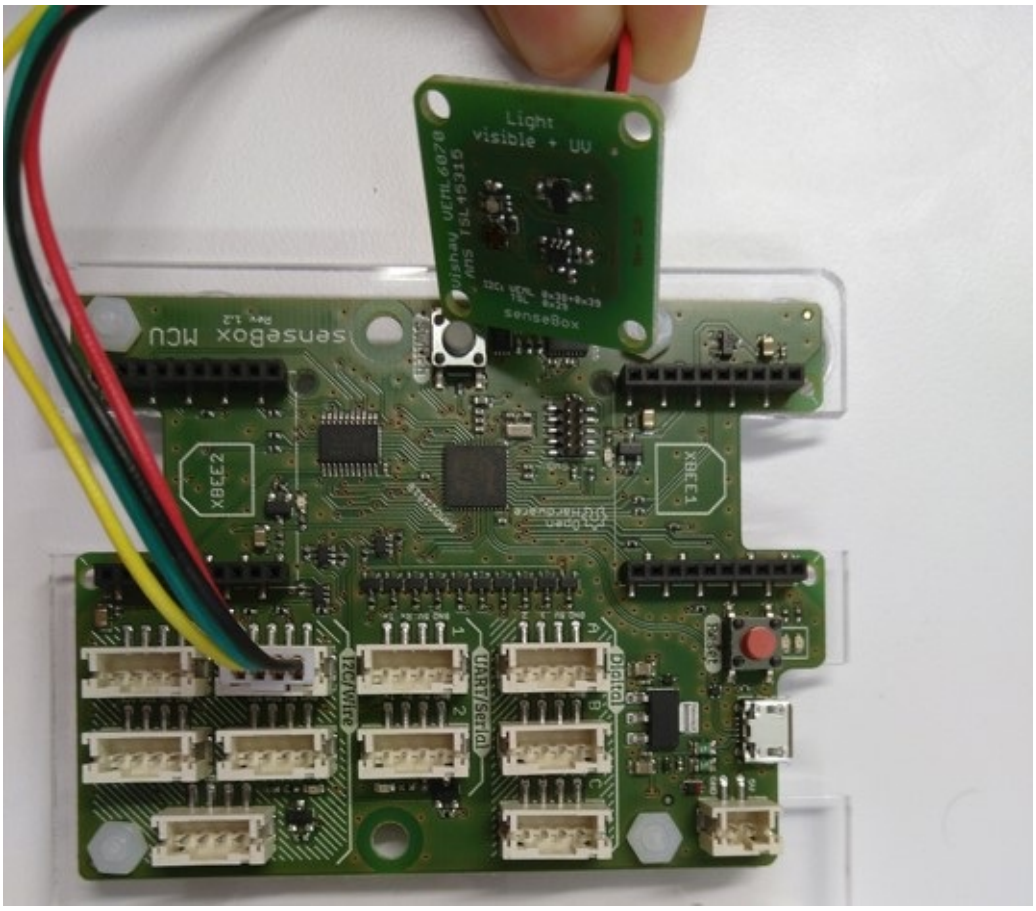
The SD-Bee is attached to port 2, which is enabled by default. The correct port can be recognized by the caption: `XBEE2` . The correct direction when plugging you can tell by the 7-sided marking on the board and the Bee.



Connection of mSD-Bee to the MCU (XBEE1)

Connection of Simple Sensors

The sensors that can be purchased with the senseBox are very easy to connect with the enclosed "I2C to I2C" cables. For this you need to use the slots on the board marked with `I2C/Wire` .



Connection of simple sensors)

Connection of the Fine Dust Sensor

The fine dust sensor that can be purchased with the senseBox has an enclosed matching cable which can connect the sensor to the board. For this you need to use the slots on the board marked `UART/Serial`. You can find further information about the connection of the particulate matter sensor here[1].

Connection of the fine dust sensor

[1]. See *5.1.3.4 Fine Dust* ↩

Step 4: Programming and Component Testing

This chapter describes how the programming of the senseBox works and how you can test the included sensors and components

---

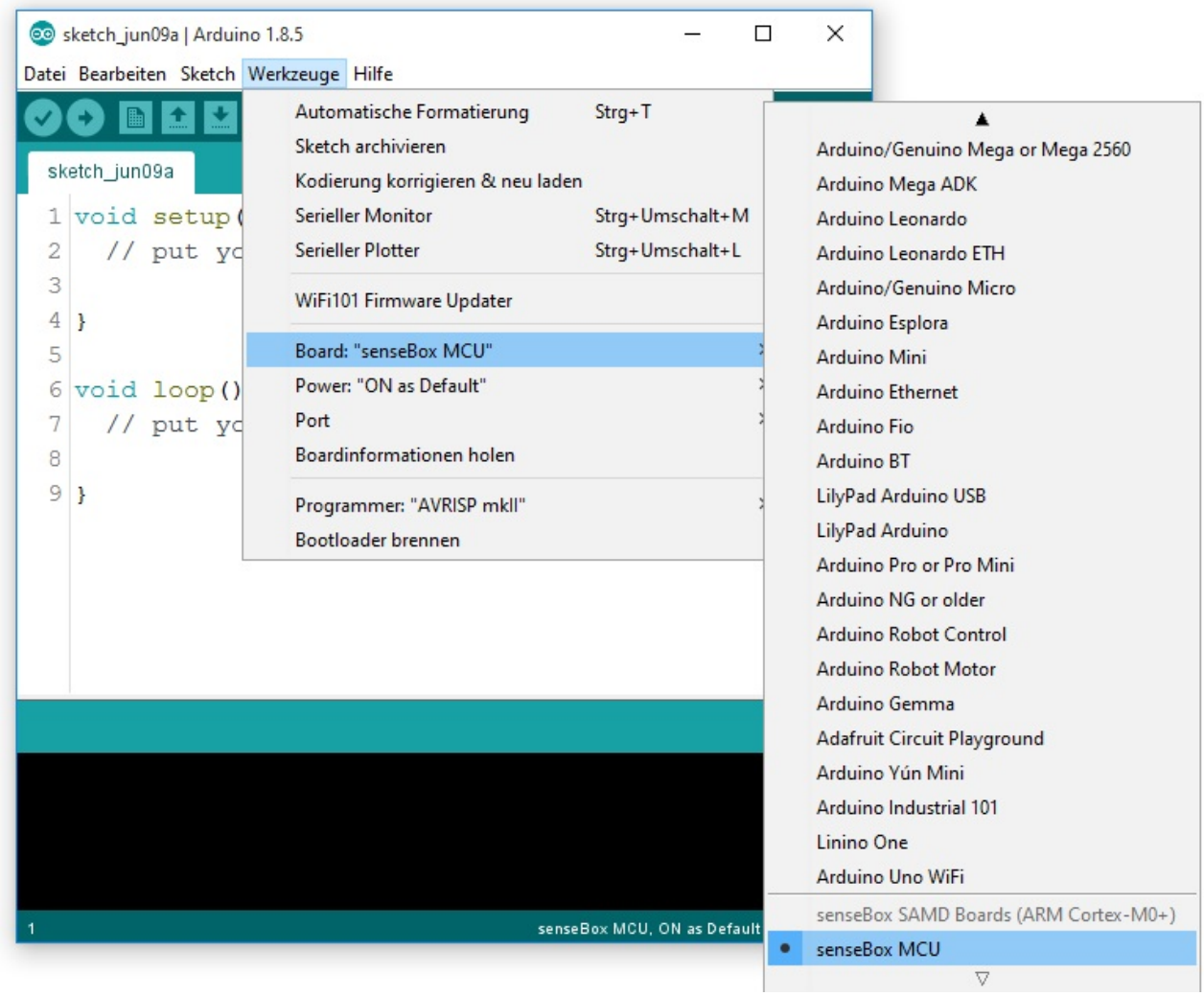
A software program for the senseBox is also called Sketch in the following

Programming with the Arduino IDE

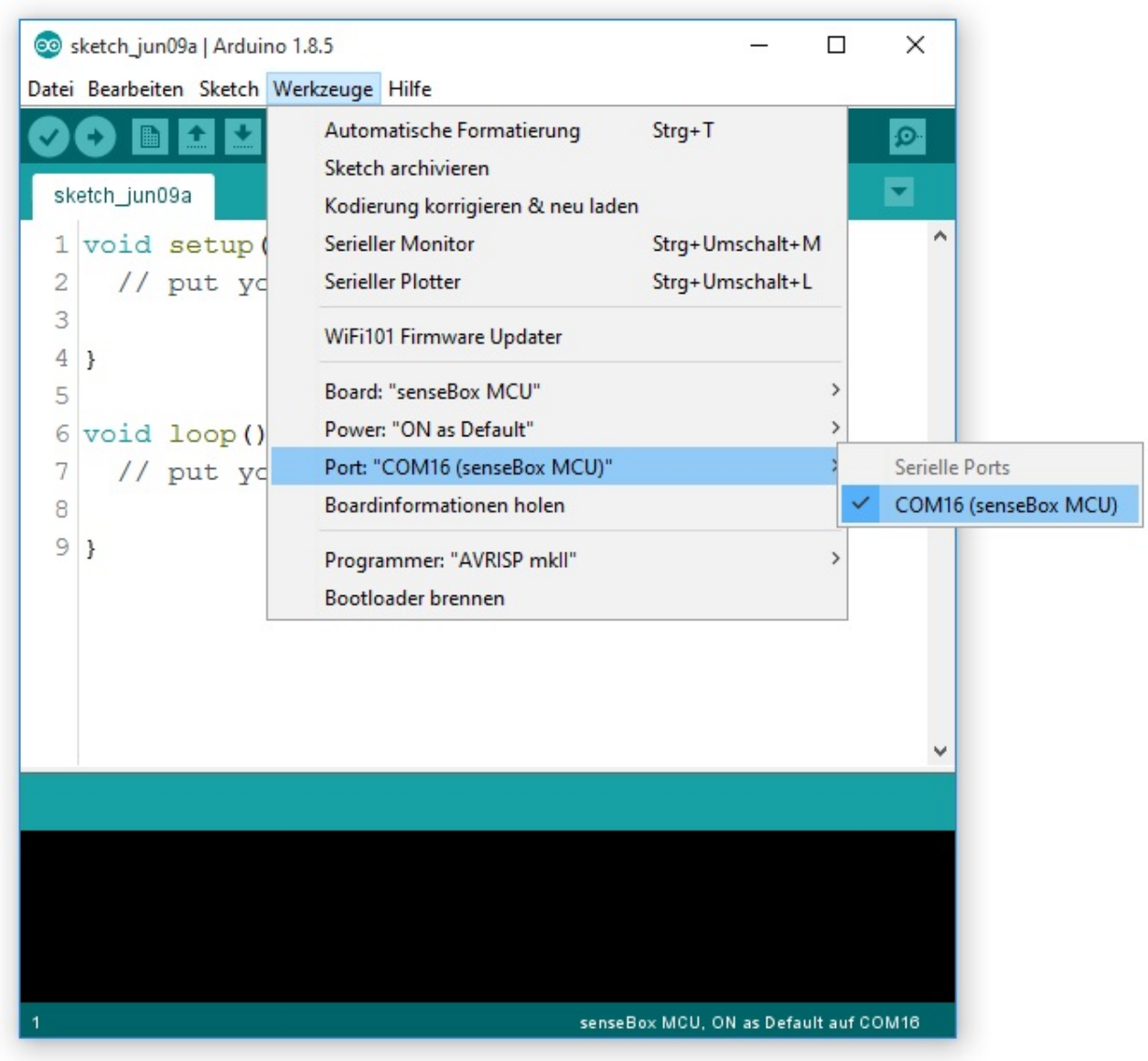
With the Arduino IDE a sketch can be compiled and uploaded to the senseBox MCU. Now connect the senseBox via USB cable to your computer and follow the next steps.

Configuration in the Arduino IDE

Before the senseBox can program, you have to make a few settings in the Arduino IDE. In the `Tools` tab you have to select the senseBox MCU at the bottom of the list under the `Boards` option.



Next, under `Tools` -> `Port` select the port number of the USB port where the senseBox MCU is connected to the computer.



port selection

> The port can only be selected if the senseBox has been connected to the computer with the USB cable.

Hello World Example

Copy the example below into your Arduino environment and click on the arrow symbol in the toolbar. In the lower part of the Arduino interface you get feedback on the upload process. If everything worked, the message `Upload completed` appears there.

```
int ledPin = LED_BUILTIN;

void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  digitalWrite(ledPin, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);                // wait for a second
  digitalWrite(ledPin, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);                // wait for a second
}
```

The text behind the // is a comment that is not evaluated by the compiler. This makes sense so that you can handle the code better and it makes it easier for other programmers to understand their and your own code.

> Unlike a laptop or smartphone runs on your senseBox no operating system such as Windows, Linux or MacOS. The senseBox MCU is a microcontroller running only the last program that has been uploaded.

Testing Sensors and Internet Connection

Before you connect your senseBox to the openSenseMap, all sensors and the network module should be checked to prevent later errors. With our test program, the measurement process and the network connection can be tested after the station has been set up.

> The prerequisite for this is the latest version of the board support package from step 2 Schritt 2. At the end of step 2 is explained how you can bring the board support package up to date.

Open the Test Sketch

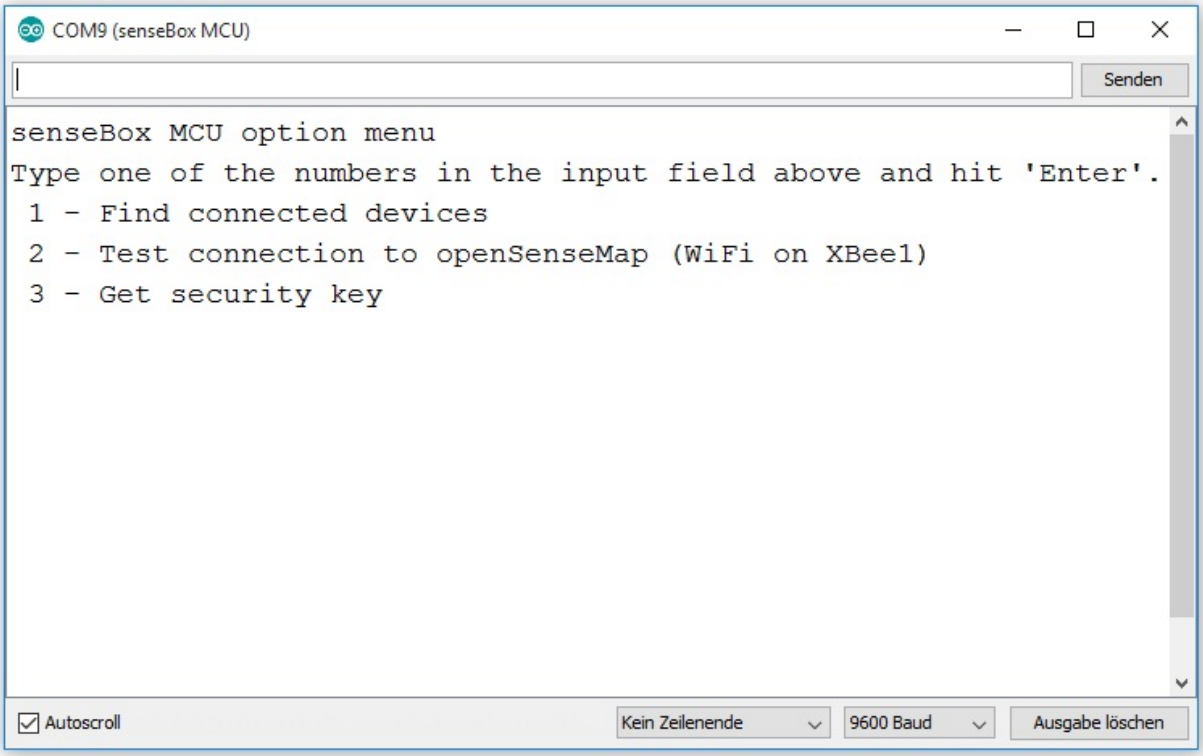
Öffnet aus den Beispielen die Datei `mcu_component_test` (`Datei` -> `Beispiele` -> `senseBoxMCU`). Nachdem ihr diesen Sketch auf die MCU hochgeladen habt, startet ihr den seriellen Monitor indem ihr auf das Lupen-Symbol rechts oben in der Werkzeugleiste klickt.

Open the file `mcu_component_test` (`File` -> `Examples` -> `senseBoxMCU`) from the examples. After uploading this sketch to the MCU, start the serial monitor by clicking on the magnifying glass icon in the upper right corner of the toolbar.

> In the event that the monitor does not open, you should check whether the board is in program mode (press Reset once) and whether the correct port has been selected. Then trie again with a click on the magnifying glass.

Options Menu

After you have opened the serial monitor, a menu appears on its functions you can access via the input field:

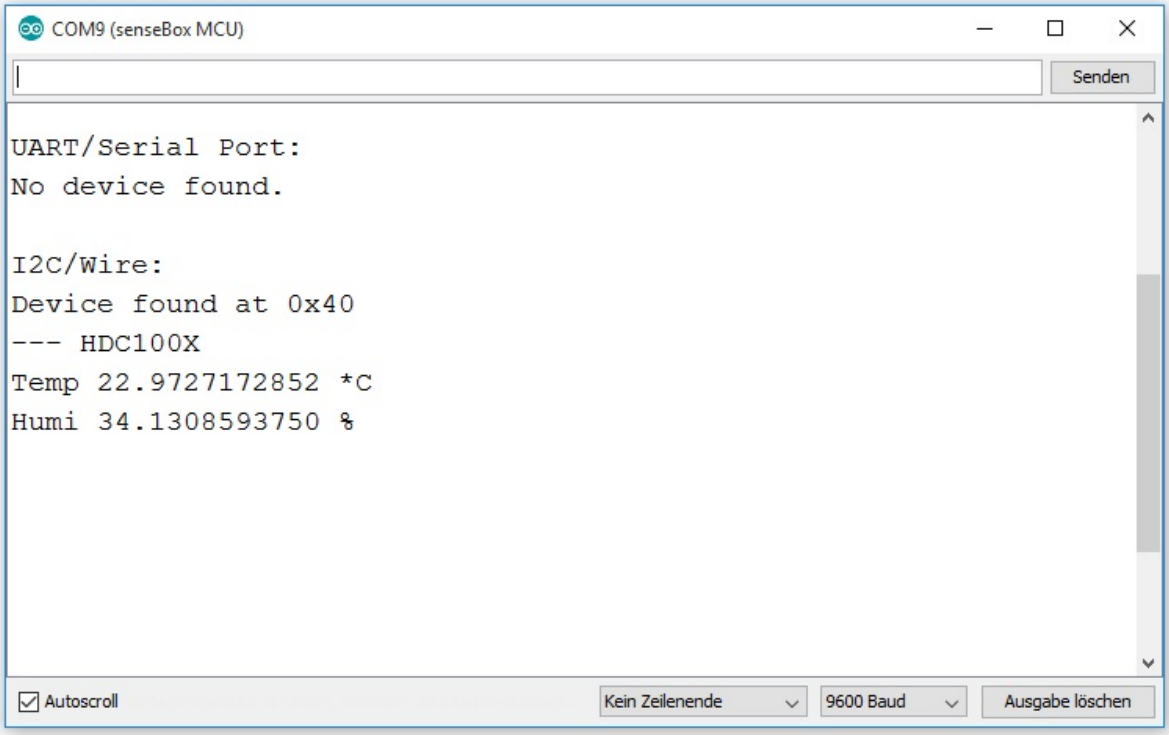


options menu

To do this, write the number of the corresponding option in the input field and click on "Send". Below is a list of options with brief descriptions.

1. Find connected sensors

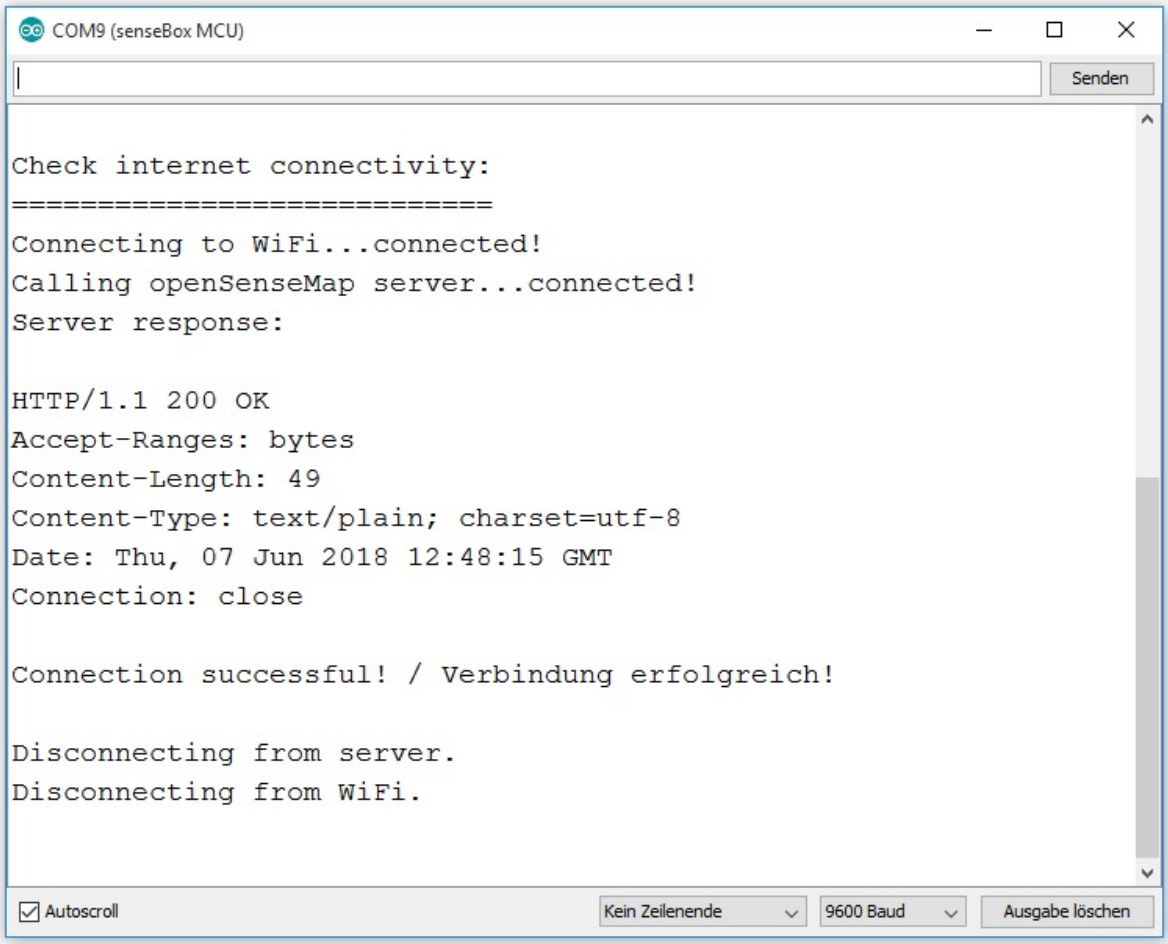
   Here you can check if all connected sensors have been properly initialized and recognized. For each connected sensor there should be a feedback and a test measurement. In the example below, an HDC1080 temperature and humidity sensor was connected to an `I2C/Wire` port.



> If one of the connected sensors is missing during the output, you should check the cable connection and repeat the test.

2. Test connection to openSenseMap

   This option tests the Internet connection. If the connection is successful, a response with HTTP status 200 should be issued by the server:



> If you use a WiFi module, it also checks if the latest version of the firmware is installed on the module. If the version is outdated, you should update it:

3. Get security key

   Each senseBox board has its own unique security key that you can read with this option. It is used to encrypt the connection between the openSenseMap and your senseBox so nobody can manipulate your measurements from the outside.

> You need this key in the next step when registering your senseBox on openSenseMap.

Arduino IDE and Sketches

Before you can start writing your first program, you have to look at the Arduino program – the so-called development environment (IDE).

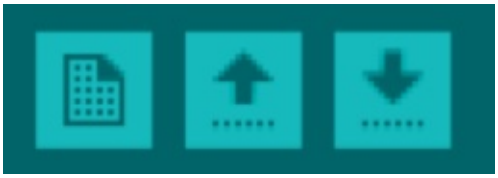
Basics

When you open the IDE, you will see a large white area directly in which you will write your program. The black area below shows status and error messages. It is always worthwhile to take a look at the news.

Lastly, you should look at the small buttons above the white area.

check for spelling errors - transfer program to senseBox MCU

The tick and the arrow are the two most important symbols for you: With the check mark you can have your program checked for spelling errors, and with the arrow you transfer your program to the senseBox MCU.



create new program - open saved program - save program

The other three icons – starting with the small leftmost leaf – are for creating a new program, opening a saved one and saving your written program.

The Arduino Sketch

An Arduino program (also called "Sketch") has a very simple structure that consists of two main components. These two required functions contain blocks of instructions that describe the program flow:

```
void setup(){
    // instruction
}
void loop(){
    // instruction
}
```

The `setup` -function is only executed once when the program is started. In the `loop` -function, however, all instructions are repeated in an endless loop. Both functions are mandatory to successfully compile and run the program. "Compile" means the translation of the program into machine code which can be understood by the Arduino processor; this is what the Arduino IDE does for us.

With a double slash (`//`) you can add comments to the program code. It is always important to comment on your program code so that others can understand what is happening in a certain place.

Comments in the Source Code

The commenting of source code is unfortunately a topic that is neglected. The benefit of good comments is often only recognized when one tries to muddle through foreign source code or when one 'digs out' one's own program after a long time. Comments are not evaluated by the compiler and therefore do not affect the program. Text and program parts that are commented out can be recognized by the fact that they are colored gray.

One-line Comments

One-line comments are often found in the source code. They serve to explain certain commands or constructs. A one-line comment is indicated by two `//`.

```
// I am a comment
int led = 1;  // variable 'led' gets the value 1
```

Multi-line Comments

Multi-line comments are often at the beginning of a program or a method. They begin with `/*` and end with `*/`. You can also use them to comment out parts of a program. For example, if you have an error and want to check in which part of the program it is.

```
/*
 *
 * I am a multi-line comment.
 * For instance I can declare the use of the whole program or
 * of a single method.
 *
 * By the way:
 * <- these stars are automatically created when writing a multi-line comment,
 * but they are no necessesity
 *
 */
```

How Many Comments Does a Source Code Need?

That is a question for which there is no clear answer. There are programmers who expect every line of code to be commented on. This is not necessary with our simple programs. Basically, at least the following program parts should be commented on:

- A comment at the beginning, which describes the purpose of the program.
- Each method must be commented on, and in particular the input parameters and possible returns.
- Mathematically or logically demanding commands or special 'gimmicks' that the programmer has considered.

> If you want to continue to deal with the topic, then look at Wikipedia

if / else-condition

With if and else it is possible to make decisions in a program and the senseBox MCU, depending on how the decision fails, to execute different code.

Use of if

To understand the meaning of `if`, let's take a look at the program code for the following example:

If you want to light an LED depending on a switch, the code would look like this:

```
if (digitalRead(BUTTON_PIN) == HIGH) {
  digitalWrite(LED_PIN, HIGH);
}
```

The first line of code begins with an `if`. Within the following brackets, the condition to be tested is specified, in this case, if the button is pressed. If this condition is `true` (returns true), the code entered in the curly braces is executed.

As you may have noticed, the condition uses a comparison operator, namely a double equal sign (`==`). A common mistake is that only a single match is used. For the Arduino, however, a single equal sign is not "check if equal" but for "set left equal value right".

Using else

With `else`, you can add an additional action to your if statement, which is alternatively executed if the condition is not true. So if you add an `else` to the above code, the whole sketch would look like this:

```
#define LED_PIN 1
#define BUTTON_PIN 3

void setup() {
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT);
}
void loop() {
  if (digitalRead(BUTTON_PIN)==HIGH){
    digitalWrite(LED_PIN, HIGH);
  } else {
    digitalWrite(LED_PIN, LOW);
  }
}
```

Loops

If you want to flash an LED 50 times, that's a lot of paperwork.
Since computer scientists are lazy, they have come up with a simple solution: loops .
A loop executes a statement several times until a certain condition is met.

Construction of loops

Loops consist of two parts, a loop head and a loop body. Instructions to be repeated are written in curly braces in the loop body. But now we have to decide how often these instructions should be repeated. This happens in the loop head.

There are different types of loops that can be used as needed. Here are the two most important types of loops to be presented.

The for-loop

`for` loops are used if you know exactly how many instructions should be repeated. In our example, we know that the LED should flash 50 times. The head of a `for` loop consists of three parts separated by a semicolon (`;`):

1. A count variable is generated, which indicates how often the loop has already been executed
2. A condition indicates when to count. What a condition looks like you have already seen in if-Anweisungen[1]!
3. A definition of how to count. Usually, the count variable is increased by `1` .

```
for (int counter = 1; counter < 50; counter = counter + 1) {
    // let the LED blink
}
```

In this example, our counter variable is called `counter` . The condition is: "As long as `counter` is less than 50". The `counter` is incremented by one after each loop. Therefore, the loop body is executed 50 times.

- For the command counter = counter + 1 you will often find the shorthand counter number ++. This one does the same.
- Of course, you can give any name to the count variable. Often the name `i` is used for ‚index'.

Task 1

Now write a statement in the loop body that will give you the value of the count variable via the serial monitor.

> Tipp: The Serial Monitor[2] explains how to do it!

- **a)** Examine what happens if you replace `counter = counter + 1` durch `counter = counter*2` or `counter--` .
- **b)** Examine what happens if you replace `int counter = 1` with `int counter = 25` .

The while-loop

In many cases, you do not know at the beginning how many times a statement should be repeated. Then you can use the `while` loop. The `while` loop has a less strict structure: The loop header consists of the identifier `while` followed by parentheses. In these brackets, an operator is written, which is checked before each loop pass. As long as this condition returns `true`, the loop body will be executed

```
while (condition) {
    // let the LED blink

}
```

You can, for example, connect a button to the Arduino and loop through it only when the button is pressed.

> Attention: A common mistake is that a condition is always true (For example, if you write as condition: 1 > 0). In this case, your loop will go through again and again. One speaks of an endless loop. In this case, your Arduino stops responding and it's relatively hard to figure out why.

exercise 2

- a) Program a `while` -loop that says "The statement is true!" via the serial monitor if a variable `a` is greater than 0.
- b) Program a `while` loop that will make an LED blink when a button is pressed.
- **c)** Each `for` loop can also be described by a `while` loop. Put the following `for` loop in a `while` loop:

```
for (int i = 10; i > 0; i--) {
    Serial.print("Countdown: ");
    Serial.println(i);
}
```

---

[1]. See *3.1.2 if/else-conditions* ↩

[2]. See *3.1.5 The Serial Monitor* ↩

Variables and Data Types

To hold data in programs, you use variables. A variable is a storage container that can be addressed by its name and in which data can be stored. You can access both read and write variables, so the value is variable..

Data Types

A variable always has an associated data type, and the following types are important to Arduino programming:

Datatype	Meaning	Description
boolean	right or wrong	Can only accept two values, 1 or 0.
char	character	Alphanumeric characters (letters, numbers, special characters)
byte	whole number	integers from 0 to 255
int	whole number	integers from -32758 to 32767
long	whole number	whole numbers from - 2 billion to 2 billion
float	floating point number	fractions
String	string	Text consisting of ASCII characters
array	variable field	A list of variables with identical datatypes

There are some conventions in programming, some rules that have been agreed upon to improve the readability of code. One of these is that variable names always start with a lowercase.

Use of Data Types

boolean

A boolean can only take two values, true or false. (`true` or `false`).

```
boolean testValue = false;
```

The assignment `= false` stands for the start value of the variable in this case.

char

For example, to save a letter, you need the data type `char` . The value is passed in single quotes(`'`).

```
char testValue = 'a';
```

byte

One byte stores an 8-bit, unsigned number from 0 to 255.

```
byte testValue1  = 18;
byte testValue2 = B10010;
```

The `B` indicates that the following sequence of numbers is written in binary code. `B10010` corresponds to 18 in the decimal system, so both variables contain the same value with different spelling. .

int

The `int` data type stores integers in a range of -32768 to 32767.

```
int testValue = 99;
```

long

The `long` data type is required if the value range of an integer is no longer sufficient. It can store integers from -2 billion to 2 billion.

```
long testValue = 9999999;
```

float

To save broken numbers you need the data type `float` .

```
float testValue = 2.4476;
```

String

A String is defined as follows:

```
String testValue = "Hello World";
```

Unlike the data types you've known before, the identifier `String` is capitalized. You have to pay attention to this, otherwise the program will not recognize the data type. Most programming languages have primitive data types and higher data types. You can tell if your identifiers are small (primitive data type) or large (higher data type). For our applications in the senseBox: edu it is not necessary to differentiate between primitive and higher data types. If you later program more complex applications, you will learn more about it. If you would like to know more about it now, then look here[1].

array

An array is not an actual data type, but rather a collection of multiple variables of the same type.

```
int testArray[5] = {5, 10, 15, 20, 15};
```

In the example, an array of type `int` is created, since integers should be stored. The 5 in square brackets after the name of the variable determines the number of memory locations. Arrays on the Arduino have a fixed size, and can not be subsequently enlarged.

The memory locations of an array are numbered beginning at 0. In a program, you can access the various memory locations of the array by placing the index of the memory space in square brackets after the variable name:

```
Serial.print(testarray[0]); // is 5
Serial.print(testarray[4]); // is 5
Serial.print(testarray[5]); // enerates an error!
```

Lifetime of Variables

A variable is always visible in the block (within the curly brackets) for the program in which the variable was declared. One distinguishes between global and local variables. Local variables are all those that have been declared within curly braces (usually within a function). Global variables are usually defined before the `setup` function and are visible to the entire program.

> Since global variables are always visible, they also consume memory for the entire program runtime. Want to save space, define variables only where you need them. If you want to know more about the lifetime of variables, look at Wikipedia

---

[1]. https://en.wikipedia.org/wiki/Data_type#Composite_types ↩

The Serial Monitor

The serial monitor is a tool to display data via the USB connection of the senseBox MCU directly in the IDE and to transfer data from the computer keyboard to the senseBox MCU.

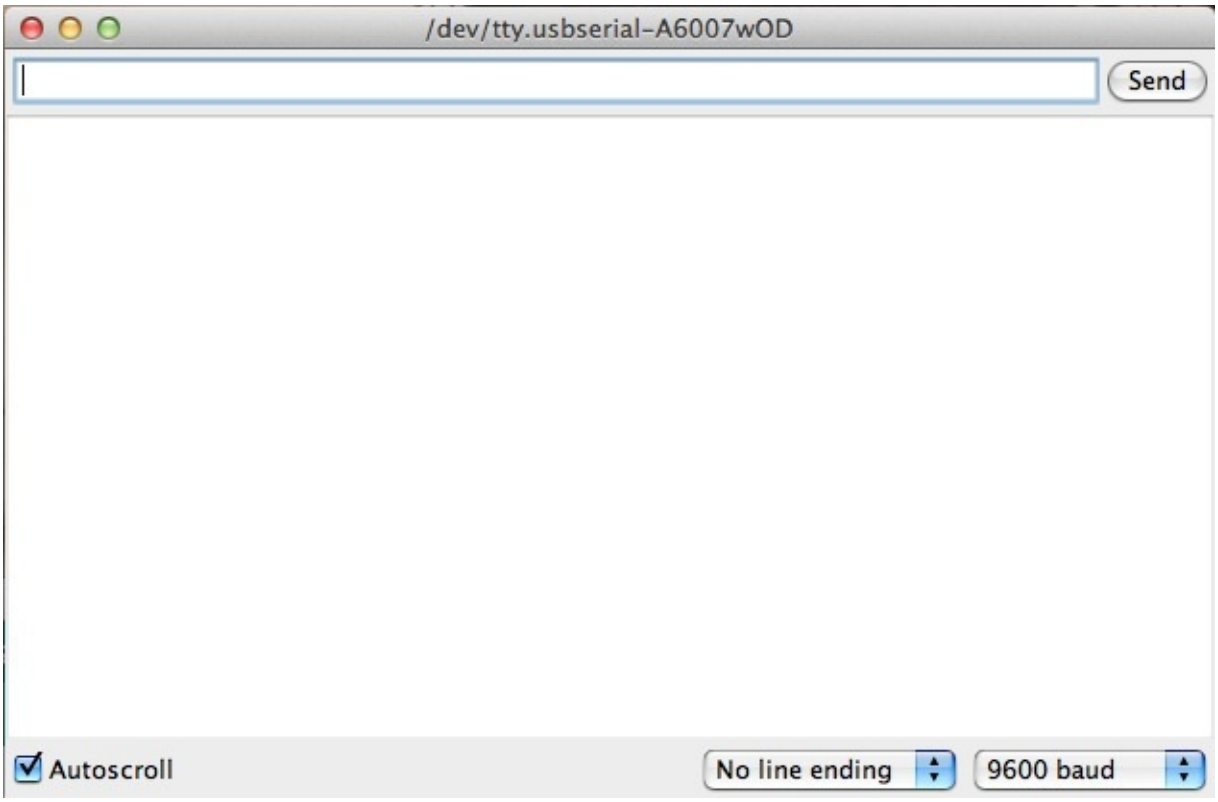
This serial monitor can be used to display data on the PC that the microcontroller sends to the PC (numbers or texts). This is very useful, because you have not always connected an LCD display on the microcontroller, on which you could read certain values.

Start the Serial Monitor

To start the serial monitor, first open the IDE and then click on the icon with the little magnifying glass in the toolbar.

magnifer-symbol

The now open window has an input line with a "Send" button at the top and an output window below it. The output window will continuously display the latest issues. If the check mark is set to Autoscroll, only the most recent editions are displayed. That is, when the output window is full, older data is pushed up out of the visible area of the screen to make room for current output. If you deactivate the autoscroll function, you have to scroll manually via the scroll bar on the right edge.



serial monitor

Output Values to the Serial Monitor

To be able to display data in the serial monitor, it must first be initialized. This happens via the function `Serial.begin(9600)` in the `setup()` Funktion. The value `9600` defines the baud rate, ie the speed with which data is transferred between the computer and Arduino. The entered value must always correspond to the speed selected in the serial monitor at the bottom right.

To send data to the serial monitor, use the `Serial.print()` and `Serial.println()` . The first variant of the function simply outputs the data, while the second variant inserts a line break at the end.

As a first try you should now show text in the output window. To display text, it must appear in quotation marks in the parentheses of the function:

```
Serial.println("senseBox rocks!");
Serial.print("senseBox ");
Serial.println("rocks!");
```

The example should in each line the text "senseBox rocks!" output. Note the use of `print` and `println` !

In addition to text, you can also display the contents of variables in the serial monitor. For this purpose, the name of the respective variable must be entered instead of the desired text:

```
String exampleVariable = "hello world!";
Serial.println(ExampleVariable);
```

Digital Signals

Digital signals can only accept the values 1 or 0 or high or low. So you only use countable elements such as fingers. Hence the term digital, which goes back to the Latin digitus, the finger. To understand how digital signals work and where we work with them, we need to understand building a sketch in Arduino..

Drive Digital Actuators

> The digital interfaces of the senseBox MCU have 4 connections: A Ground(GND), a power supply (5V) and two digital pins with which you can control the actors!

To control a digital actuator - for example, an LED - you need two commands: The first is in the `setup()`, the second in the `loop()`. In the `setup` -function the command `pinMode(1, OUTPUT);` determines that pin 1 has something connected to it, which should be used as output (or OUTPUT). The 1 can be replaced by any other pin number, depending on which Arduino pin you have connected the actuator. The second function in `loop()` is `digitalWrite (1, HIGH);`. Thus, the actuator connected to pin 1 is supplied with power, ie switched on. The counterpart to this command would b `digital´Write(1, LOW);` to stop the power supply again. Again, the 1 is replaceable by any other pin number. The sketch should look like this:

```
void setup(){
    pinMode(1,OUTPUT); //Declare the pin on which the LED
                       // is connected as output
}
void loop(){
    digitalWrite(1,HIGH); // Turn on the LED
}
```

Reading Digital Sensors

The same pins we used to drive digital actuators can also be used to register input signals. Digital inputs can assume two states just like digital outputs; `HIGH` or `LOW`. For incoming signals to be processed, they must be stored in Variablen[1].

To store digital signals, a Boolean variable (also called `boolean`),which can take only two values, is particularly suitable. In order to read out a digital sensor, two commands are needed, similar to the control of digital sensors. In the `loop()` the command pinMode `pinMode(1, INPUT);` Pin 1 of the Arduino set as input.In `setup()` the command `testVariable = digitalRead(1);` a sensor connected to pin 1 is read out and the value stored in the previously created test variable. Just like driving digital actuators, 1 stands for the pin used and can be replaced by any other digital pin. The sketch should look like this:

```
boolean testVariable = 0;          // declare a new variable of thype boolean

void setup() {
  pinMode(1,INPUT);
}
void loop() {
  testVariable = digitalRead (1); // write the read value into the variable
}
```

The content of the created variable can be displayed in the serial monitor[2].

---

[1]. See *3.1.4 Variables* ↩

[2]. See *3.1.5 The Serial Monitor* ↩

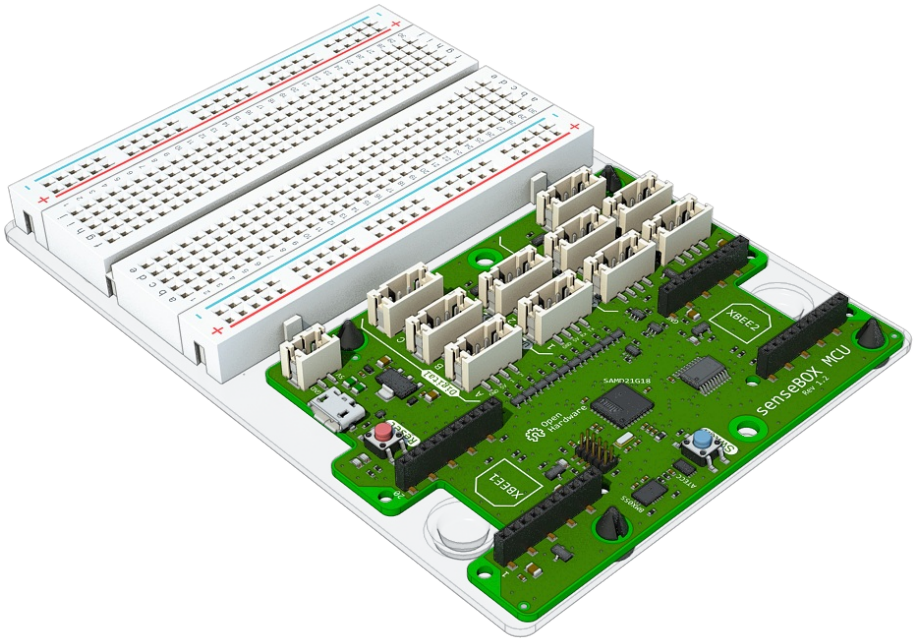
The Serial Data Bus

The Arduino can communicate with other devices via a data bus. A data bus describes a system that allows two or more devices to exchange data in an orderly fashion. With our Arduino, the second device would almost always be a sensor or an actuator.

Der I²C-Bus

The I²C bus is an easy to use data bus to transfer data. Here, the data is transferred between the Arduino and the other device through two cables called `SDA` und `SCL`. The `SDA` (serial data) designated line is the data line over which the actual data is transmitted. The `SCL` (serial clock) line is also called the clock line and specifies the clock frequency. On the Arduino you will find the two ports as `A4` (SDA) and `A5` (SDC).

If several I²C devices are to be connected to the Arduino, this is implemented via a series connection. The SDA cable on the first sensor would therefore be extended to the next sensor on the same row of the breadboard:



senseBox MCU with Breadboard

If you use the I²C bus on the Arduino, the Arduino is always considered a master device and all other devices on the bus as a slave. Each slave has its own address in the form of a hexadecimal number, with which it can be clearly addressed. Usually, each device brings with it a range of bus addresses that can be used. The respective addresses can be found in the data sheet of the

manufacturer.

Analog Signals

Sorry, this section is not finished yet. We are just about to be work on it.

Bees

A bee denotes a pluggable component with which the senseBoxMCU can transmit or store data. Here you have the choice between the Wifi-Bee or the mSD-Bee.

WiFi Bee

The WiFi Bee is the connector to connect the senseBox to the Internet. Your readings will be transferred to the existing network via WLAN (WiFi). The WiFi Bee is based on the ATWINC15000 microchip from Atmel, which has a very low power consumption and a long range.

Configure the WiFi Bee & Upload on the openSenseMap

> Make sure you have the latest board support package installed because you need the correct software libraries. How to do that was explained to you in Step 2!

▼ Declaration of the objects

First, create an instance of Bee and openSenseMap.

```
Bee* b = new Bee(); // instance of Bee
OpenSenseMap osem("senseBox ID",b); // instance of openSenseMap
float temp = 24.3; // Test value that we later upload to openSenseMap
```

> Make sure that you have to replace the parameter "senseBox ID" with your Box ID!

Once we have done this, the bee can be called in the program code consecutively with the abbreviation `b`. In the `setup ()` function, we now connect to our desired WiFi network and upload a first test value on the openSenseMap.

▼ setup()

```
void setup(){
    b->connectToWifi("SSID","PW"); // Connect to WiFi
    delay(1000);
    osem.uploadMeasurement(temp,"sensor ID") // Test value is uploaded
                                             //"sensor ID" still needs to be replaced
    };
```

> Make sure that you have to replace the parameters "SSID" with the network name of your WiFi network, "PW" with the

> corresponding password and "sensor ID" with the sensor ID of the corresponding sensor!

Now, your WiFi Bee has connected to the Internet and should upload a first value to openSenseMap.

Congratulations, you have just uploaded your first data on the map, now you are ready for your first Environmental Measuring Station[1].

---

[1]. See */../../projekte/Umweltstation/README.md* ↩

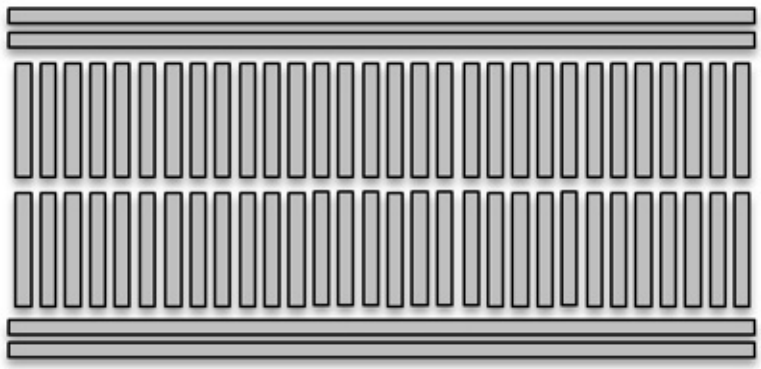
Breadboard

In the senseBox: edu you will find the senseBox MCU together with a "breadboard" on a holder. What this "breadboard" can do and how it is structured can be found here.

Use of a Breadboard

The breadboard helps you to safely connect circuits without soldering. The electronic components are simply plugged into the spring contacts, so a circuit can be changed quickly by relocating.

The breadboard consists of two mirrored, non-conductive connected sides. These each consist of two long (horizontal in the following figure) rows for the plus and minus terminals and two times 30 (vertical rows in the following figure) rows of five spring contacts labeled a to e and f to j, respectively. The plus and minus terminals and the five horizontal spring contacts of a row are conductively connected as shown below.



Breadboard

Registration of the senseBox:edu as :home on openSenseMap

Register your senseBox now at openSenseMap to collect environmental data and make it available to a community of researchers and tinkerers, and help to measure the environment. Become part of one of the biggest citizenScience movements worldwide!

Attention!
The openSenseMap is a portal where people worldwide upload measurement data of their senseBoxes or other measuring stations. These data are used by individuals, but also by researchers, for scientific purposes. You should therefore only register the senseBox:edu on the oSeM if you want to upload data in the longer term and present these real environmental measurements and are not falsified for experimental purposes. If you still want to register your senseBox: edu, you simply have to enter it as senseBox:home-v2.

You do not even know what the openSenseMap is? Then check it out, go to www.opensensemap.org[1] and discover a huge pool of open-data with nearly a billion measurements worldwide!

You have already seen the openSenseMap? - What are you waiting for? Now go to the openSenseMap[2] and register your senseBox to upload, view and process measurements.

1. https://www.opensensemap.org ↩

2. https://opensensemap.org ↩

DIY Environmental Measuring Station

In this project you will learn how to set up a senseBox environmental station. In the end, the measurement of various environmental phenomena such as temperature, humidity, brightness and air pressure, as well as the publication of the data on the openSenseMap be possible!

Introduction

This project is the most extensive. It has been split into several subchapters. In each new chapter, an additional module is introduced until a complete - the functions similar to the senseBox: home - weather station was built!

Within this project you will find the following 5 chapters:

- Temperature and Humidity[1]
- Experiments with Light[2]
- UV-Sensor[3]
- Air Pressure[4]
- Data Upload to the OSeM[5]

You can edit the stations as individual projects, or expand your program code with each station to measure all five phenomena simultaneously.

Good luck building your DIY Environmental Station!

---

[1]. See *4.1.1 Temperature and Humidity* ↩

[2]. See *4.1.2 Experiments with Light* ↩

[3]. See *4.1.3 UV-Sensor* ↩

[4]. See *4.1.4 Air Pressure* ↩

[5]. See *4.1.5 Data Upload to OSeM* ↩

DIY - Temperature and Humidity

So that we can see the weather report daily on the Internet, on television, in the newspaper or in apps, not only satellite data are evaluated. Also data from weather stations on the ground play an important role in the prediction. But how does the measurement and display of temperature and humidity values work?

Requirements

- The Use of Software Libraries[1]
- The Serial Data Bus I²C[2]
- The Serial Monitor[3]

Destinations of the Station

In this station we are dealing with the temperature and humidity sensor of the senseBox, the HDC1080.

Materials

- combined temperature and humidity sensor `HDC1080`

Basics

▼ HDC1080 Sensor

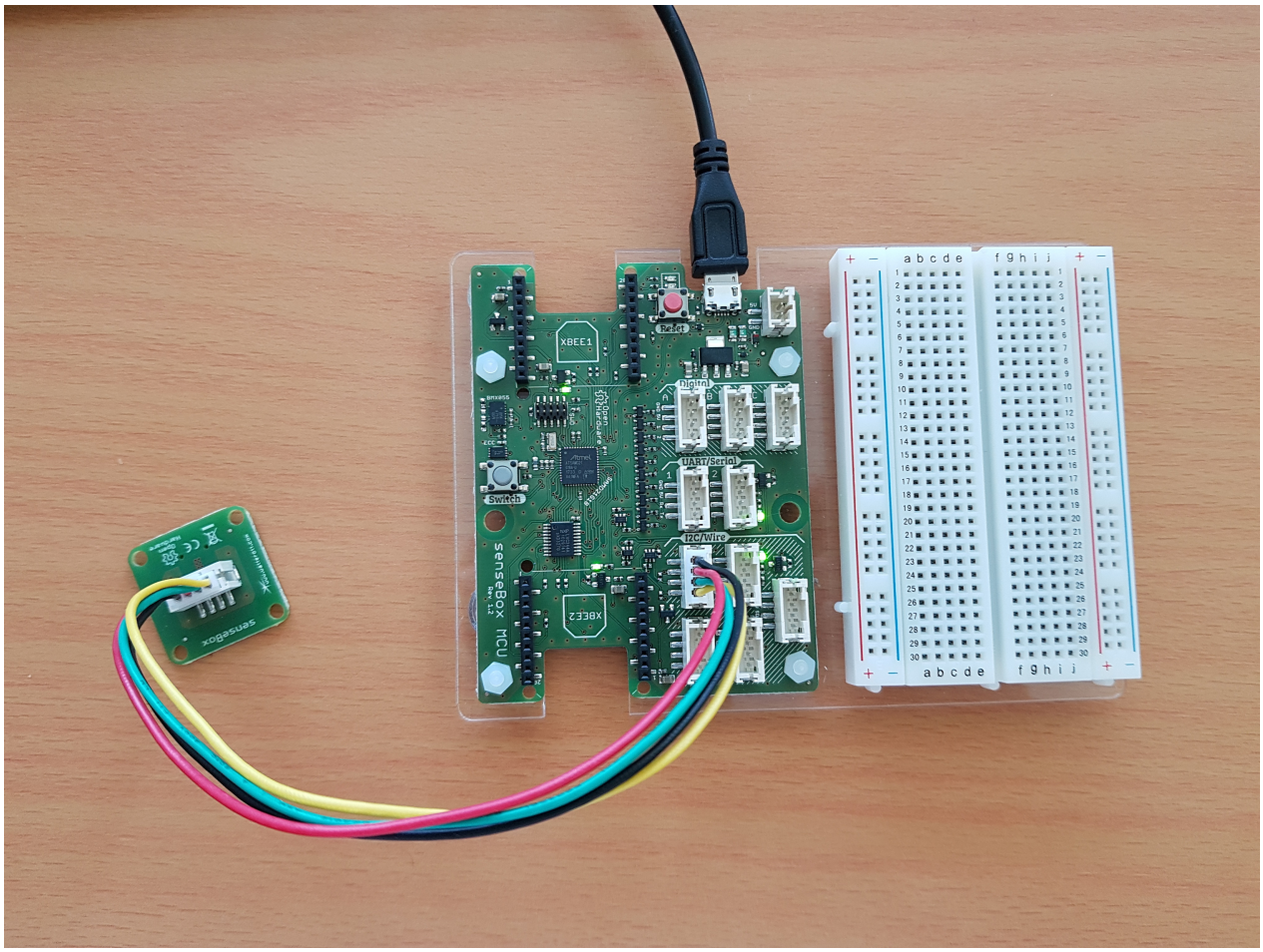
The `HDC1080`, from the Texas Instruments HDX10XX series, is a combined temperature and humidity sensor. The sensor can measure the humidity from 0% to 100%, as well as the temperature from -40 ° C to 125 ° C with an accuracy of ± 2% or ± 0.2 ° C.

▼ I²C Bus

The communication of the sensor with the microcontroller runs over the seriellen Datenbus I²C[4]. Unlike simple digital or analog inputs, multiple I²C devices (such as sensors or displays) can be connected in parallel to the data bus. Each device has a unique identifier so that the data bus can assign each one of them and address them separately.

Construction

Plug in the circuit as you see it in the graphic below.



Temperature and humidity sensor connected via I2C port

Programming

> Make sure you have the latest board support package installed because you need the correct software libraries. How to do that was explained to you in step 2 !

First, an instance of the sensor must be created.

```
#include "SenseBoxMCU.h"
HDC1080 hdc;
```

▼ setup() function

In the `setup()` -function the sensor should now be started:

```
void setup(){
    hdc.begin();
}
```

▼ loop() function

After initializing the sensor as described above, you can use two commands in the `loop ()` -function to output a temperature or humidity

```
void loop(){
    hdc.getHumidity();
    hdc.getTemperature();
}
```

> When storing the measured values, the variables should have the same data type as the return values of the measuring functions. In our case these are both float values.!

Tasks

▼ Task 1

Build the circuit which is described above and try to read the HDC1008 out. Print the measured data in the serial monitor.

> For this look at the examples from First steps !

---

1. See *2.2 Step 2: Board-Support-Packages installieren* ↩
2. See *3.3 The serial Data Bus(I²C)* ↩
3. See *3.1.5 The Serial Monitor* ↩
4. See *3.3 The serial Data Bus(I²C)* ↩

DIY - Experiments with Light

If you watch television, turn on the radio, write a message on your smartphone, or warm up food in the microwave, you use electromagnetic energy. Nowadays, all people are constantly reliant on this energy. Without them, life in modern cities would be completely different from what you know.

Requirements

- The Use of Software Libraries[1]
- The Serial Data Bus I²C[2]
- The Serial Monitor[3]

Destinations of the Station

In this station, you use a light sensor to measure the illuminance of visible light in lux.

Materials

- Light sensor `TSL 45315`

Basics

▼ Light intensity

Electromagnetic energy moves in waves through space. Their spectrum ranges from very long radio waves to very short-wave gamma radiation. The human eye can only perceive a very small part of this spectrum: the visible light. Our sun is the source of energy across the spectrum. The Earth's atmosphere protects us from being exposed to excessive levels of radiation that could be life-threatening for us.

For us, the intensity of visible light is particularly interesting. In order to measure the so-called illuminance of the incident light in the visible part of the spectrum, the unit lux is used. It gives the ratio of brightness in lumens per square meter. On a bright sunny day, it is over 100,000 lux, but only about 1 lux in a full moon night.

▼ TSL45315 Sensor

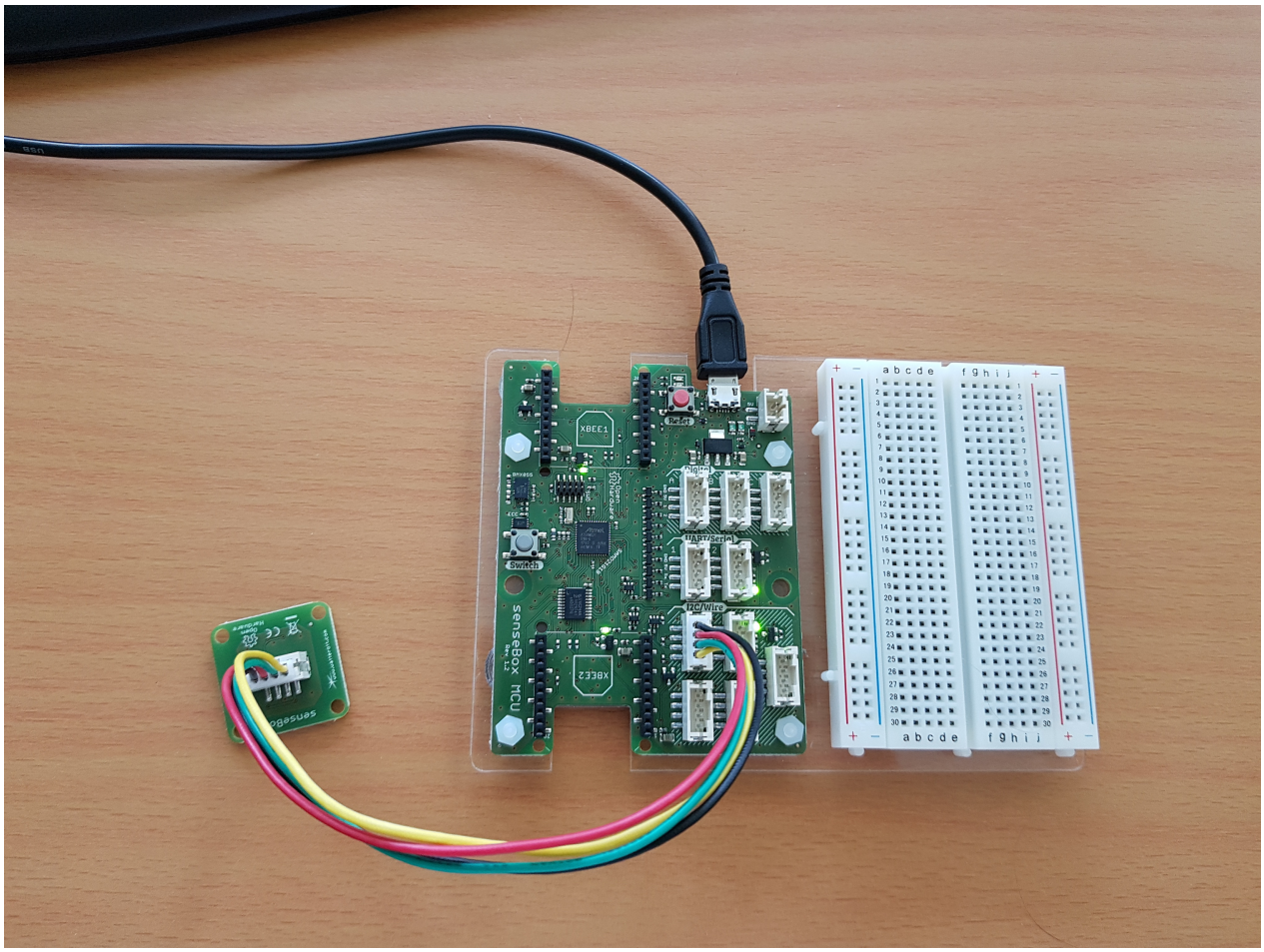
For this measurement, we will use the sensor TSL45315 from AMS-TAOS below. The sensor datasheet shows that its sensitivity is matched to the visible part of the light spectrum, which is approximately between 400 and 700 nm. According to the data sheet, this sensor has a range of 2 to 200,000 lux, with a resolution of 3 lux. Furthermore, the sensor must be operated at 3.3V.

The sensor is addressed via the I²C protocol. We address it directly with the following commands taken from the data sheet: For this measurement, we will use the sensor TSL45315 from AMS-TAOS below. The sensor datasheet shows that its sensitivity is matched to the visible part of the light spectrum, which is approximately between 400 and 700 nm. According to the data sheet, this sensor has a range of 2 to 200,000 lux, with a resolution of 3 lux. Furthermore, the sensor must be operated at 3.3V.

The sensor is addressed via the I²C protocol. We address him directly with the following commands taken from the data sheet:

ADDRESS	RESISTER NAME	R/W	REGISTER FUNCTION	RESET VALUE
--	COMMAND	W	Specifies register address	0x00
0x00	CONTROL	R/W	Power on/off and single cycle	0x00
0x01	CONFIG	R/W	Powersave Enable / Integration Time	0x00
0x04	DATALOW	R	ALS Data LOW Register	0x00
0x05	DATAHIGH	R	ALS Data HIGH Register	0x00
0x0A	ID	R	Device ID	ID

Construction



Exposure and UV sensor connected via I2C port

Programming

> Make sure you have the latest board support package installed because you need the correct software libraries. How to do that is explained in step 2!

First, an instance of the sensor must be created.

```
#include "SenseBoxMCU.h"
TSL45315 lux_sensor;
```

▼ **setup() function**

In the `setup ()` -function the sensor should now be started:

```
void setup(){
    lux_sensor.begin();
}
```

▼ **loop() function**

In the `loop ()` -function, we can use the getIlluminance () command to get the current measured light intensity:

```
void loop(){
    lux_sensor.getIlluminance();
}
```

Exercises

▼ **Exercise 1**

Combine the code from this lesson and add a function to output the data in the serial monitor.

▼ **Exercise 2**

Try to turn an LED on and off depending on the lighting. For this the chapter if/else - Bedingung[4] can be helpful.

---

[1]. See *2.2 Step 2: Board-Support-Packages installieren* ↩

[2]. See *3.3 The serial Data Bus(I²C)* ↩

[3]. See *3.1.5 The Serial Monitor* ↩

[4]. See */../../grundlagen/if_else_bedingung.md* ↩

DIY - UV Light Sensor

When temperatures rise in the summer and we spend more time outdoors, we increasingly try to protect ourselves from the UV rays of the sun, for example with sunscreen. But is there only this UV radiation in summer? What does it look like when the sun is obscured and how much do the values fluctuate? Find out how much UV radiation reaches the earth and measure the UV intensity with your senseBox!

Requirements

- The Use of Software Libraries[1]
- The Serial Data Bus I²C[2]
- The Serial Monitor[3]

Aims of the station

In this station, we use a UV light sensor to measure the intensity of UV light in microwatts per square centimeter (µW / cm²). Then we want to convert the value into the UV index.

Materials

- UV-light sensor `VEML6070`

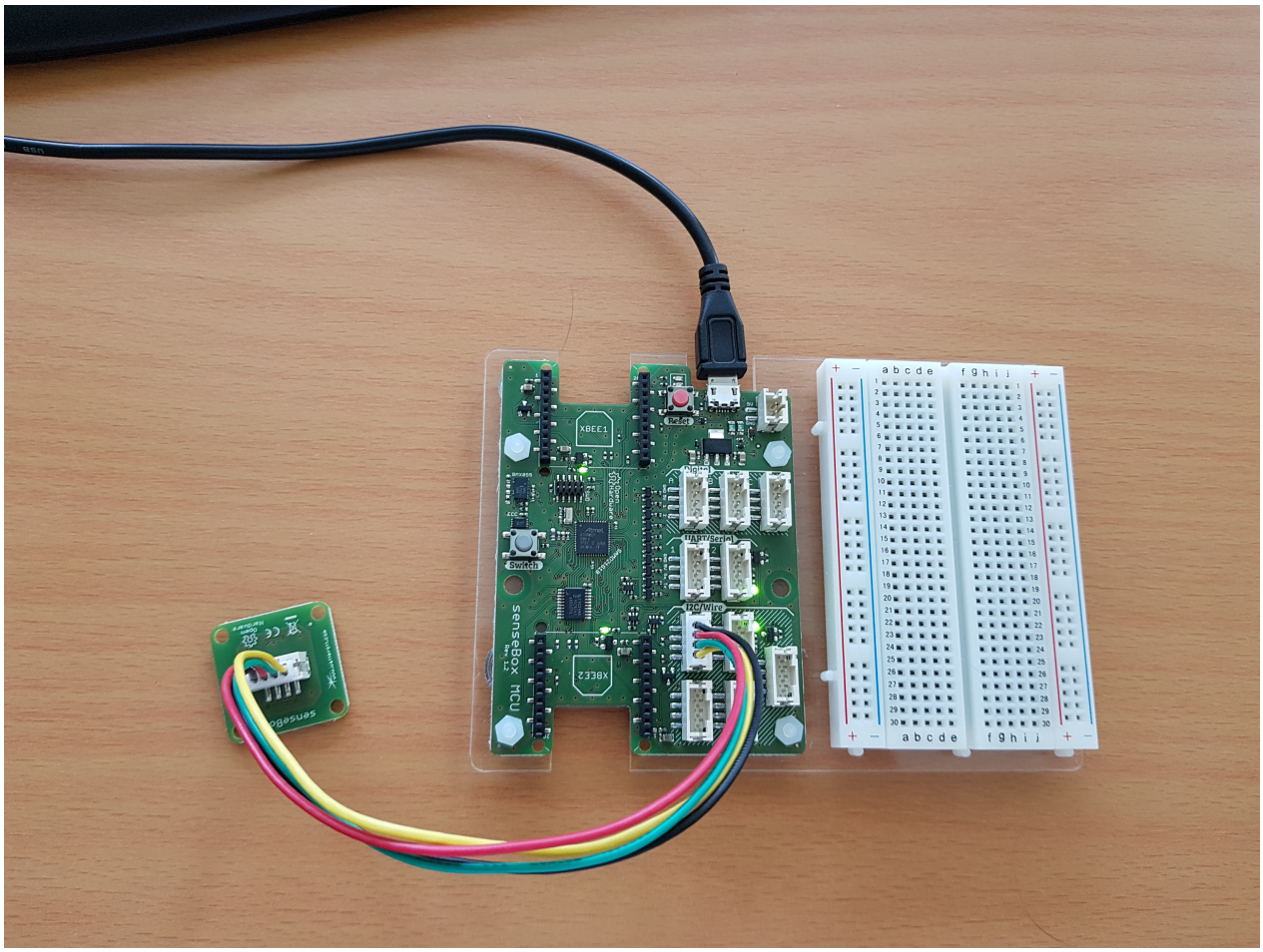
Basics

▼ **Ultraviolet light**

Ultraviolet (UV) light is invisible electromagnetic radiation to humans with a wavelength shorter than visible light but longer than that of X-rays. UV light covers the wavelengths from 100 nm to 380 nm. Because of the absorption in the earth's atmosphere - especially in the ozone layer - only little UV-B radiation (100 - 300 nm) penetrates to the earth's surface. UV-A radiation (300 - 380 nm), which is less harmful to human skin, is less absorbed by the atmosphere.t.

UV light intensity is measured in microwatts per square centimeter (µW / cm²). Our sensor measures in the range of approx. 300 - 400 nm, so it absorbs only UV-A radiation (for more details see the Datasheet[1]).

Construction



Exposure and UV sensor connected via I2C port

Connect the sensor to the senseBoxMCU as shown in the graphic.

Programming

> Make sure you have the latest board support package installed because you need the correct software libraries. How to do that was explained to you in step 2 !

First, an instance of the sensor must be created.

```
#include "SenseBoxMCU.h"
VEML6070 vml;
```

▼ setup() Funktion

In the `setup()` -function the sensor now should be started:

```
void setup(){
    vml.begin();
}
```

▼ loop() Funktion

In the `loop()` -function, we can use the `getIlluminance()` command to get the current measured light intensity:

```
void loop(){
  vml.getUvIntensity();
}
```

> If you want to see the UV index you have to declare a function before that will do it for you. How to do that you will learn in the next step!

▼ Transformation to UV-Index

Since the UV-Index[2] is often used in everyday life, we now want to write a method that converts the measured value into a UV index:

```
/*
 * getUVI()
 * expects the knife of the UV sensor as an input parameter
  * and returns the corresponding value on the UV index
 */
float getUVI(int uv) {

  float refVal = 0.4; // Reference value: 0.01 W / m 2 is equivalent to 0.4 as UV index
  float uvi = refVal * (uv * 5.625) / 1000;
  return uvi;
}
```

Exercise

▼ Exercise 1

In the serial monitor, try using the `getUVI ()` -function to print out the UV index.

---

1. See *2.2 Step 2: Board-Support-Packages installieren* ↩
2. See *3.3 The serial Data Bus(I²C)* ↩
3. See *3.1.5 The Serial Monitor* ↩
1. https://github.com/sensebox/resources/raw/master/datasheets/datasheet_veml6070-UV-A-Light-Sensor.pdf ↩
2. https://en.wikipedia.org/wiki/Ultraviolet_index ↩

Air Pressure

The measurement of the air pressure allows not only weather forecasts but also indirectly the determination of the height of the sensor.

Requirements

- The Use of Software Libraries[1]
- The Serial Data Bus I²C[2]
- The Serial Monitor[3]

Materials

- Air pressure sensor `BMP280`

Basics

▼ BMP280 sensor

The BMP280 sensor measures both air pressure (hPa) and temperature (° C). This sensor is controlled via the I²C Protokoll[4], and requires an operating voltage of 3.3 to 5 volts.

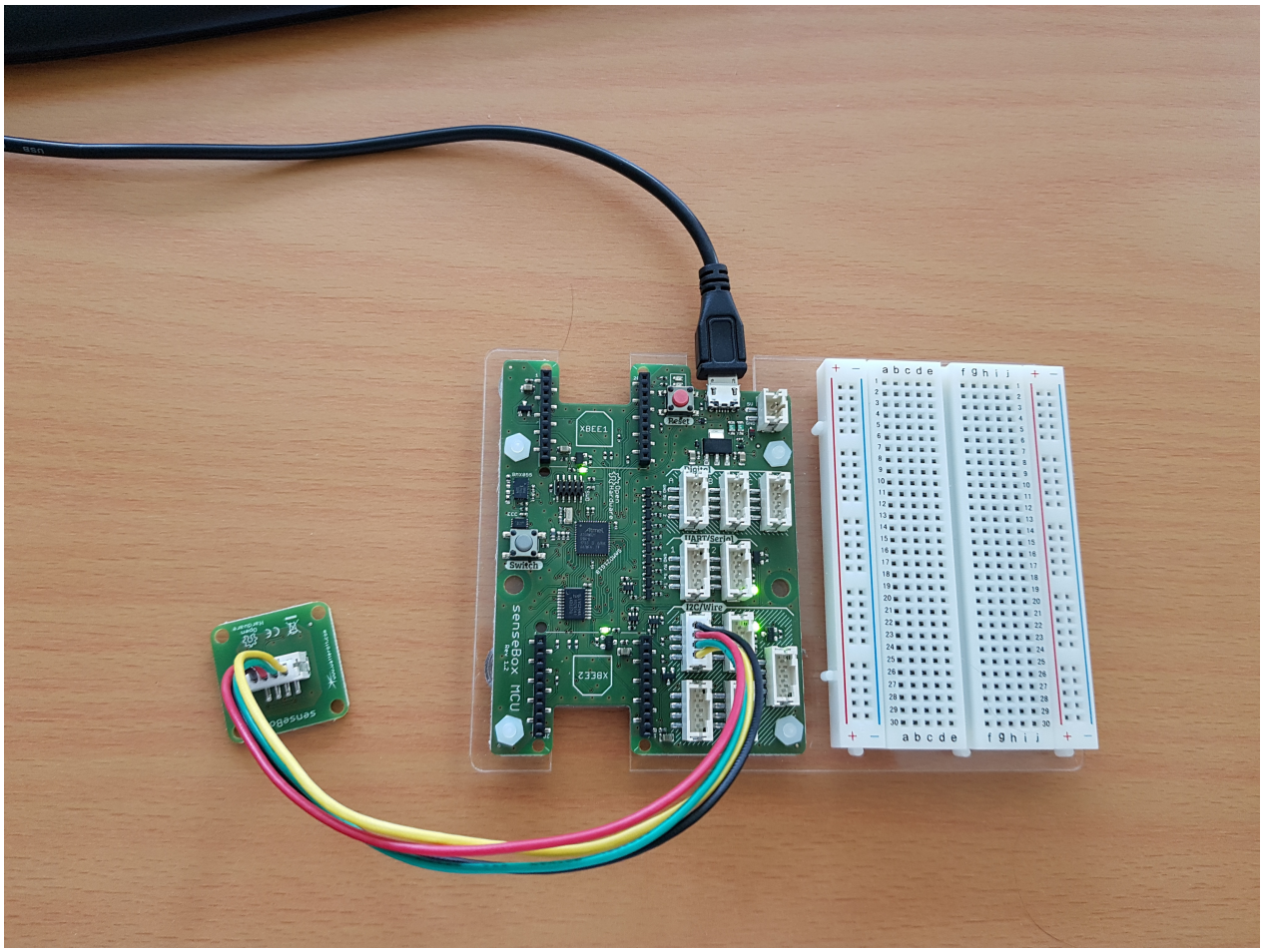
I²C devices are connected to the senseBoxMCU via the I²C / Wire port and thus read out digitally (see also The serial databus[5]). The I²C address of the BMP280 can be switched via the `SDO` pin: If `SDO` is at ground (`GND`) the address is `0x76` , otherwise `0x77` . This communication is handled by the senseBox library for us.

▼ Height determination via the air pressure

Since the air pressure depends on the altitude above sea level, the body height of the senseBox can also be determined via the `BMP280` . For this purpose, a reference pressure `P0` is needed whose height is known. Usually, the current air pressure is used at sea level. Since the air pressure can fluctuate greatly depending on the current weather, this "altitude measurement" but is not very accurate, and must be recalibrated again and again.

Construction

To get the sensor up and running just plug it in to the I²C / wire port!



Temperature and air pressure sensor connected via I2C port

Programming - Reading the sensor

The sensor can be controlled via the `SenseBoxMCU.h` library. After this has been integrated, an instance `bmp` of it must be created. On this object all functions of the library are called:

```
#include <SenseBoxMCU.h>
BMP280 bmp_sensor;
```

▼ setup() function

In the `setup ()` -function, the sensor must be initialized. Use the following lines

```
void setup(){
    bmp_sensor.begin();
}
```

▼ loop() function

Now the sensor has to be read in the `loop ()` -function. The variables `temp` and `pressure` then each contain the current measured values.

```
void loop(){
    double temp, pressure;
    pressure = bmp_sensor.getPressure();
    temp = bmp_sensor.getTemperature();
}
```

Exercises

▼ Exercise 1

Connect the `BMP280` sensor to the Arduino, and create an Arduino sketch, which regularly outputs air pressure and temperature on the serial monitor!

▼ Exercise 2

You have learned that the construction height of the senseBox can be determined from the measured air pressure. Use the function `bmp.altitude (...)` to calculate the height and output it on the serial monitor as well.

> Take a look at the example enclosed with the BMP280 library. The reference pressure P0 must be adapted to the current weather conditions: [here](https://www.meteoblue.com/en/weather/webmap/index/?variable=mslp_pressure&level=surface&lines=none)[^1] you will find the current air pressure.*

---

1. See *2.2 Step 2: Board-Support-Packages installieren* ↩
2. See *3.3 The serial Data Bus(I²C)* ↩
3. See *3.1.5 The Serial Monitor* ↩
4. See *3.3 The serial Data Bus(I²C)* ↩
5. See *3.3 The serial Data Bus(I²C)* ↩

1. https://www.meteoblue.com/en/weather/webmap/index/?variable=mslp_pressure&level=surface&lines=none ↩

Data Upload

If we set up our weather station, it would be nice to be able to retrieve the data from any location. There is the [openSenseMap](https://openSenseMap.org/)[^1] ((OSeM), which collects various sensor data online and displays it on a map. Via the Ethernet or WiFi-Bee or we can connect our senseBox to the Internet and upload the data to the OSeM.

Requirements

- Using Software Libraries[1] : You should have looked at the first steps for this purpose.
- **Bees**[2]: Read this chapter to learn how the senseBox MCU can use Bees to establish a network connection to transfer data to openSenseMap.

Destinations of the station

In this station, the integration of a sensor into the openSenseMap is shown by way of example so that the data obtained is available online.

Materials

- WiFi-Bee
- At least one (arbitrary) sensor

Programming

In the chapter Bees[3] you have already learned how to connect to the Internet, now let's see how we can continuously upload our metrics to openSenseMap. As already described there, we must first create the instances for the openSenseMap and provide our Wi-Fi network + access data.

▼ Declaration of the objects

```
#include "SenseBoxMCU.h"
Bee* b = new Bee(); // Instance of the Bee
OpenSenseMap osem("senseBox ID",b); // Instance of the openSenseMap
HDC1080 hdc; // Instance of the temperature and humidity sensor
void setup(){
    b->connectToWifi("SSID","PW"); // Vconnect to the Wifi
    hdc.begin();
    };
```

In the `loop ()` -function we upload our measurements.

▼ loop()

```
void loop(){
    osem.uploadMeasurement(hdc.getTemperature(),"Sensor ID")
    delay(5000);
    };
```

Exercises

▼ Exercise 1

Familiarize yourself with openSenseMap (see Requirements) and register your senseBox with the sensors you have previously connected.

▼ Exercise 2

In the Arduino sketch, which you received when registering in Exercise 1, the reading of sensors is still missing. Extend the sketch from the OSeM registry so that your connected sensors are read out.

> Follow the instructions in the basic chapter.You can reuse most of your previous code!<

---

[1]. https://openSenseMap.org/ ↩

[1]. See *2.2 Step 2: Board-Support-Packages installieren* ↩

[2]. See *3.5 Bees* ↩

[3]. See */../../grundlagen/Bees.md* ↩

Traffic Light

It should be simulated a traffic light. With a button you can switch the traffic light.

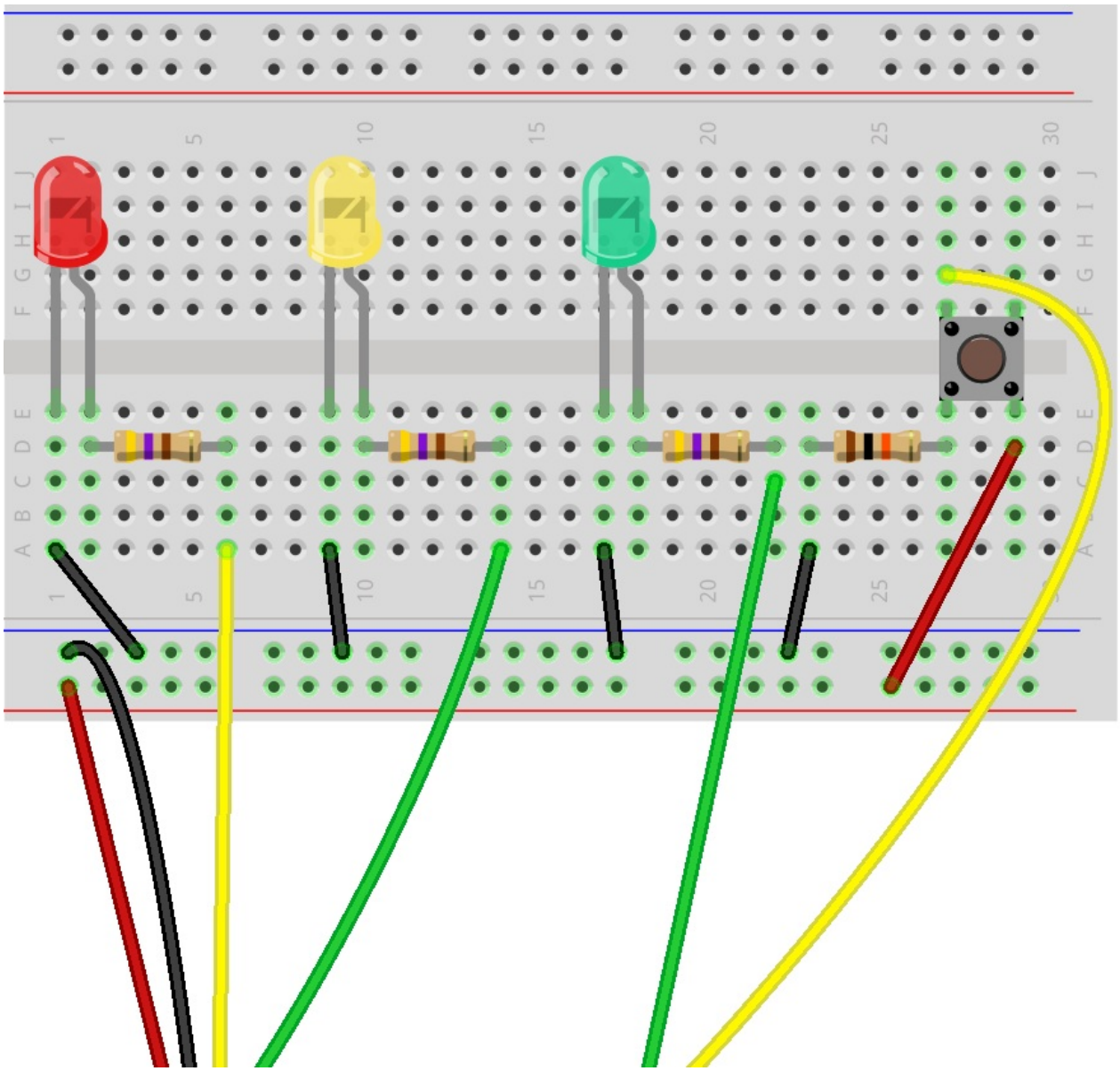
Materials

- senseBox MCU
- red LED
- yellow LED
- green LED
- 3x 470Ω resistance
- Button
- 10Ω resistance
- 2x senseBox JST adapter cable

Construction

Hardware Configuration

To connect all components you need two JST adapter cables. The first is connected to Digital A (digital pins 1 and 2), the second to Digital B (digital pins 3 and 4). On the cable in Digital A the red and the yellow LED are connected, on the cable in Digital B the green LED and the button.



Wiring the traffic light circuit

Sketch

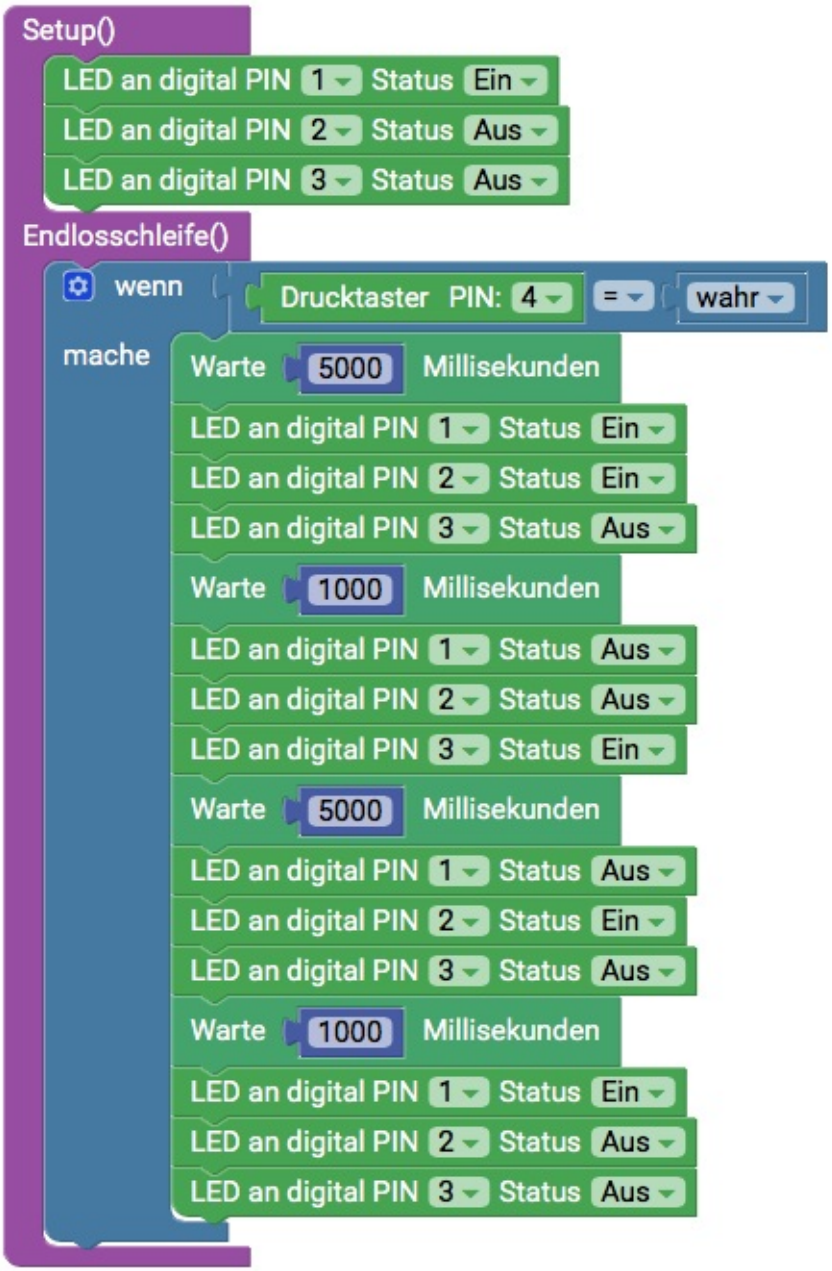
- Arduino Quellcode
- Blockly

Arduino Quellcode

```
int rot = 1;
int gelb = 2;
int gruen = 3;

int button = 4;

void setup() {
 pinMode(rot, OUTPUT);
 pinMode(gelb, OUTPUT);
 pinMode(gruen, OUTPUT);

 // The button should measure inputs
 pinMode(button, INPUT);

 // First set the traffic light to RED
 digitalWrite(rot, HIGH);
 digitalWrite(gelb, LOW);
 digitalWrite(gruen, LOW);
}

void loop() {

 // Here it is checked if the button is pressed
 if(digitalRead(button) == HIGH) {

   delay(5000);

   // ROT zu GRUEN
   digitalWrite(rot, HIGH);
   digitalWrite(gelb, HIGH);
   digitalWrite(gruen, LOW);

   delay(1000);

   digitalWrite(rot, LOW);
   digitalWrite(gelb, LOW);
   digitalWrite(gruen, HIGH);

   delay(5000);


   // green to red
   digitalWrite(rot, LOW);
   digitalWrite(gelb, HIGH);
   digitalWrite(gruen, LOW);

   delay(1000);


   digitalWrite(rot, HIGH);
   digitalWrite(gelb, LOW);
   digitalWrite(gruen, LOW);
 }
}
```

- At the beginning of the `loop()` -function it is queried each time whether the button is pressed.
- `digitalRead(button)` reads the current state of the button. If pressed, the function returns `HIGH`, otherwise `LOW`.
- To check if the button was pressed `digitalRead(button)` has to be compared with `HIGH`. The comparison is done with **two** equals `==` (comparison operator). **One** equals `=` is an assignment, such as `int red = 13`.

Blockly


Setup()
LED an digital PIN 1 Status Ein
LED an digital PIN 2 Status Aus
LED an digital PIN 3 Status Aus
Endlosschleife()
wenn
Drucktaster PIN: 4
=
wahr
mache
Warte 5000 Millisekunden
LED an digital PIN 1 Status Ein
LED an digital PIN 2 Status Ein
LED an digital PIN 3 Status Aus
Warte 1000 Millisekunden
LED an digital PIN 1 Status Aus
LED an digital PIN 2 Status Aus
LED an digital PIN 3 Status Ein
Warte 5000 Millisekunden
LED an digital PIN 1 Status Aus
LED an digital PIN 2 Status Ein
LED an digital PIN 3 Status Aus
Warte 1000 Millisekunden
LED an digital PIN 1 Status Ein
LED an digital PIN 2 Status Aus
LED an digital PIN 3 Status Aus

Blockly

Traffic Counters

The goal is to develop a traffic or passenger counter.
For this we use an ultrasonic distance sensor. The values recorded in this way should be output in the serial monitor

Materials

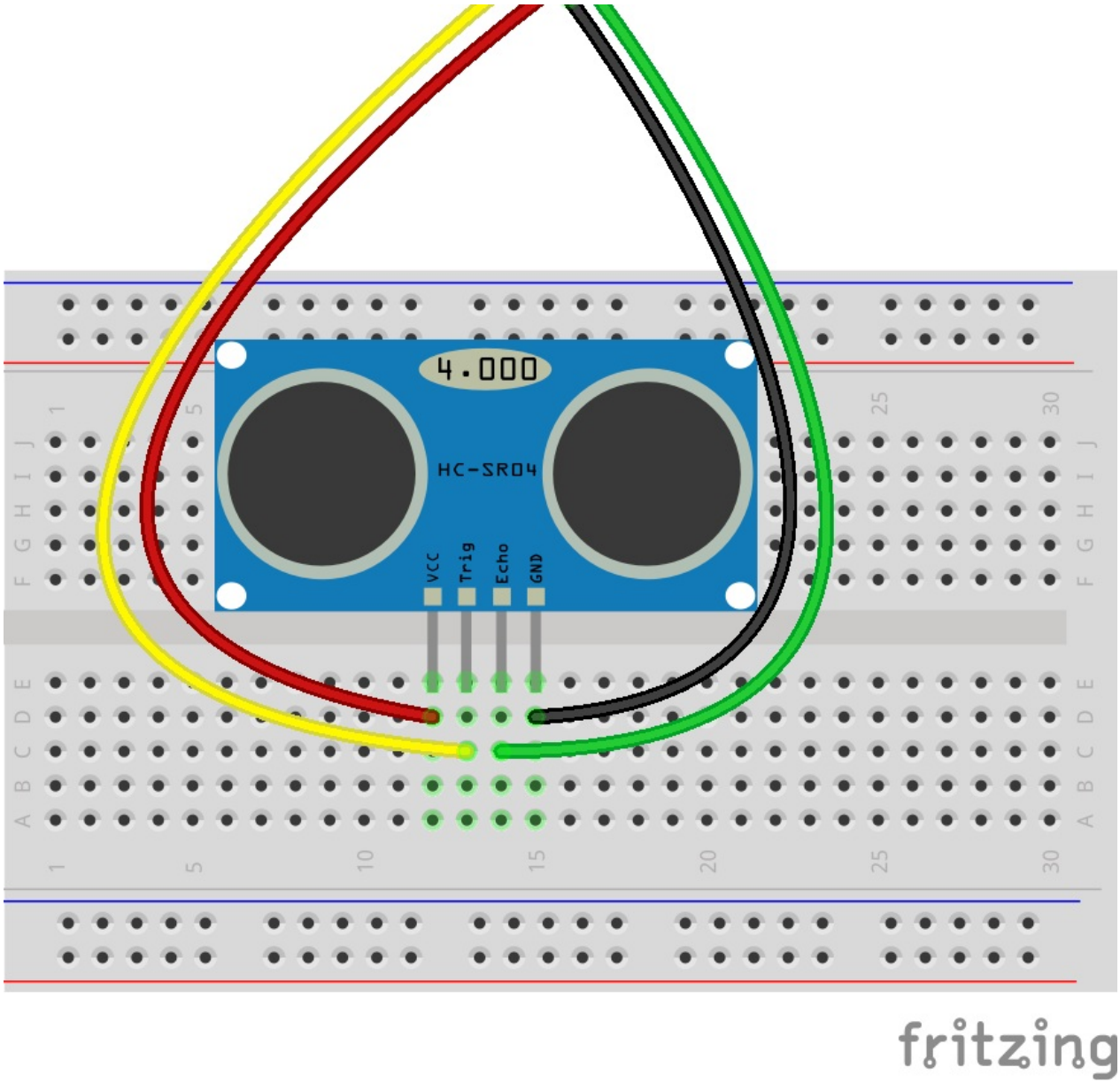
- Ultrasonic distance sensor
- senseBox MCU
- senseBox adapter cable

Basics

The ultrasonic distance sensor uses the sound to determine the distance of objects. The sensor sends out a pulse and measures the time until it receives the echo of the pulse again. From this time one calculates with the help of the speed of sound the distance of the object.

Construction

The ultrasonic sensor is connected to the senseBox MCU using a JST adapter cable. To do this, connect the JST adapter cable to the Digital A slot. For power supply the VCC pin of the sensor is connected to the red cable (5V) and the GND pin of the sensor to the black cable (GND). For data transmission then the green cable (1) with the echo and the yellow cable (2) with the Trig pin of the sensor is connected.



Wiring of the ultrasonic sensor

Note: You can of course use any slot labeled "Digital", but remember to change the code.

Programming

Defines the pins where you connected the sensor as usual. In addition, two variables are created in which the measured time and the calculated distance are stored.

```
int trig = 2;  // Trig pin of the sensor is on pin 2
int echo = 1;  // Echo pin of the sensor is connected to pin 1.
unsigned int time = 0;
unsigned int distance = 0;
```

In the `setup()` -function you have to start the Serial Monitor and define the pins to which the sensor is connected as input or output. The trigger pin of the sensor must be defined as output and the echo pin as input.

```
Serial.begin(9600);
pinMode(trig, OUTPUT);
pinMode(echo, INPUT);
```

The `loop()` -function will start with the commands:

```
digitalWrite(trig, HIGH);
delayMicroseconds(10);
digitalWrite(trig, LOW);
```

sent out a 10 microsecond ultrasonic pulse The following command `time = pulseIn(echo, HIGH);` stores the time until the echo is received in the variable `time` . Finally, the distance from the time must be calculated, and the values are displayed on the serial monitor.

```
distance = time / 58;
Serial.println(distance);
```

Note We assume that the sound propagates at 348 meters per second. This number is not fixed but depends on the ambient temperature.[1].

▼ Exercise 1

Attempts to develop a personal or traffic counter with the help of known commands and the above sketch to the ultrasonic sensor.

Note the following notes:

- Try to evaluate only a certain distance range, so that it does not interfere with movements in the background. Effectively the sensor measures approx. 3 meters.
- To avoid multiple counts of a stationary vehicle, you should program a condition that stops counting until the track is clear, so the sensor measures a preset maximum distance for the lane. For this purpose, a `while` loop offers. First, it must be checked whether something is in the measuring range. As long as the sensor does not measure that the roadway is free again, it should measure again. Only when the lane is clear again increase your counter variable by one.
- To prevent the measured values from fluctuating too much when entering the measuring range, it may help to program a delay of 200ms between the individual measurements.

---

[1]. https://en.wikipedia.org/wiki/Speed_of_sound ↩

Mobile Station

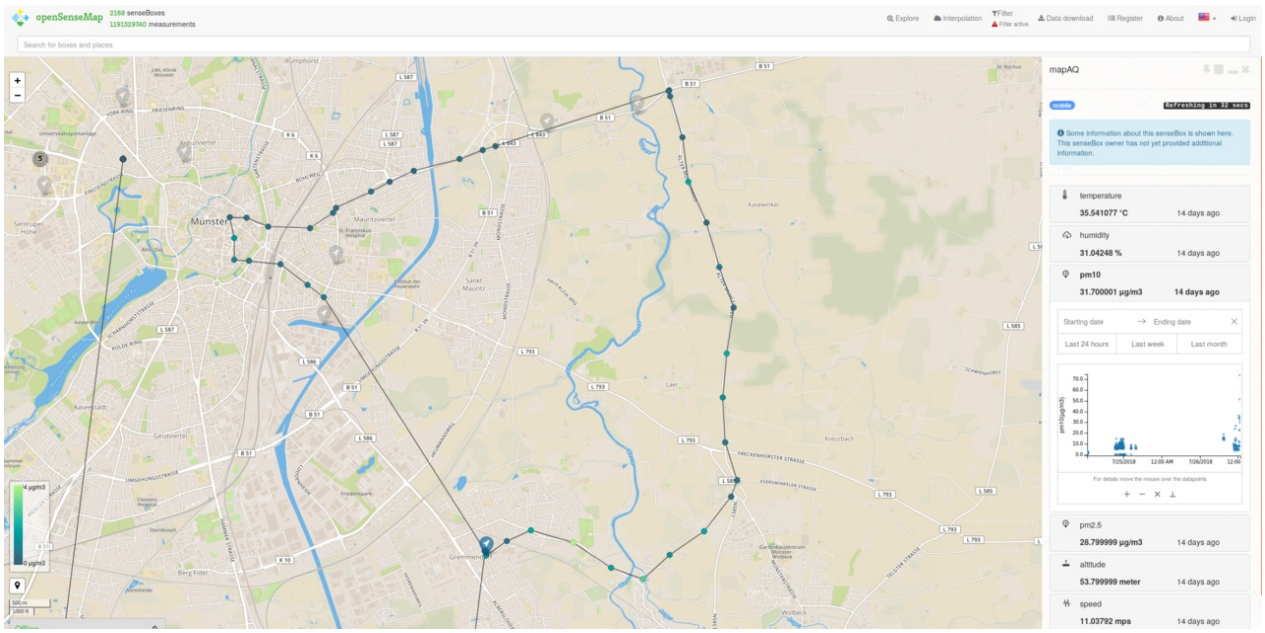
A mobile senseBox can upload data to openSenseMap from anywhere you have internet!
How this works is explained in this chapter with an example.

Material

- senseBoxMCU
- GPS module
- senseBox JST adapter cable

Basics

A mobile senseBox has many applications. For example, would you like to measure the concentration of particulate matter, your daily route to work or school? The GPS module receives the position (longitude / latitude / high) of the senseBox. This sensor is compatible with the popular GNS systems (GPS, QZSS, GLONASS, BeiDou, Galileo) and is based on the u-blox CAM-M8Q Multi GNSS module.



A mobile senseBox on openSenseMap

Construction

In order for the GPS module to have a signal throughout the measurement period, and thus be able to request the location, you must make sure that nothing obscures the module. Ideally, you attach it outside the case for the measurement period.

For the power supply during the measurement period you can, for example, use a power bank. Some power banks turn off, too little power is drawn (for example, when a cell phone is fully charged), so that it may happen that the power bank does not permanently power the senseBoxMCU. Before measuring, make sure that this will not be the case.

> It may take a while for the GPS sensor to receive correct signals. Especially when used for the first time this process can take up to 5 minutes. After a network has been detected, the information is internally stored in the sensor so it can go faster with the next use!

Programming

> Make sure you have the latest board support package installed because you need the correct software libraries. How to do that was explained to you in step 2!

How the GPS module is programmed is exemplified by a temperature measurement. Each metric of a mobile station is uploaded along with its latitude and longitude values.

First, an instance of the sensors must be created. In addition, we define 2 more variables for latitude and longitude:

```
#include "SenseBoxMCU.h"
HDC1080 hdc;

GPS gps;
float lat; // Geographic latitude
float lon; // Geographic longitude
```

In the `setup()` -function we now start the two sensors.

▼ setup() function

```
void setup(){
    hdc.begin();
    gps.begin();
}
```

The `loop()` -funktion now queries the location of the station and uploads it to the openSenseMap together with the value for temperature..

▼ loop() function

```
void loop(){
    lat = gps.getLatitude();
    lon = gps.getLongitude();
    temp = hdc.getTemperature();

    osem.uploadMobileMeasurement(temp,"SensorID",lat,lon)
}
```

Overview of Available Components

Here you will find a list with all sensors, bees and other components of the senseBox. We created a page for each part to explain it and provide you the information necessary to work with the part.

If you want to know how and for what you can use the part, click on it.

1. senseBox MCU
2. Bees
   - Wifi-Bee[1]
   - LAN-Bee[2]
   - SD-Bee[3]
   - LoRa-Bee[4]
3. Sensors
   - Temperatur & Humidity (HDC1080)[5]
   - Air pressure & Temperatur[6]
   - Illumination and UV[7]
   - Fine dust[8]
4. Additional Components
   - Sensor Protection[9]
   - Casing[10]
   - Power Adabter und USB-cable[11]
   - LED-Display[12]
   - HUB[13]
   - Micro-SD Karte[14]
   - GPS[15]

There are many more sensors which you can adapt with some own initiation and do-it-yourself skills. However, currently we can only provide information about the listet components. If you are interested in other sensor technology you can visit our Forum[1]. There you reveice help from the senseBox community.

1. See *5.1.2.1 Wifi-Bee* ↩
2. See *5.1.2.2 LAN-Bee* ↩
3. See *5.1.2.3 mSD-Bee* ↩
4. See *5.1.2.4 LoRa-Bee* ↩
5. See *5.1.3.1 Temperature & Humidity (HDC1080)* ↩
6. See *5.1.3.2 Air Pressure & Temperature* ↩
7. See *5.1.3.3 Light and UV Intensity* ↩
8. See *5.1.3.4 Fine Dust* ↩

[9]. See *5.1.4.1 Radiation Protection* ↩

[10]. See *5.1.4.2 Housing* ↩

[11]. See *5.1.4.3 Power Supply and USB Cable* ↩

[12]. See *5.1.4.4 LED-Display* ↩

[13]. See *5.1.4.5 Expander* ↩

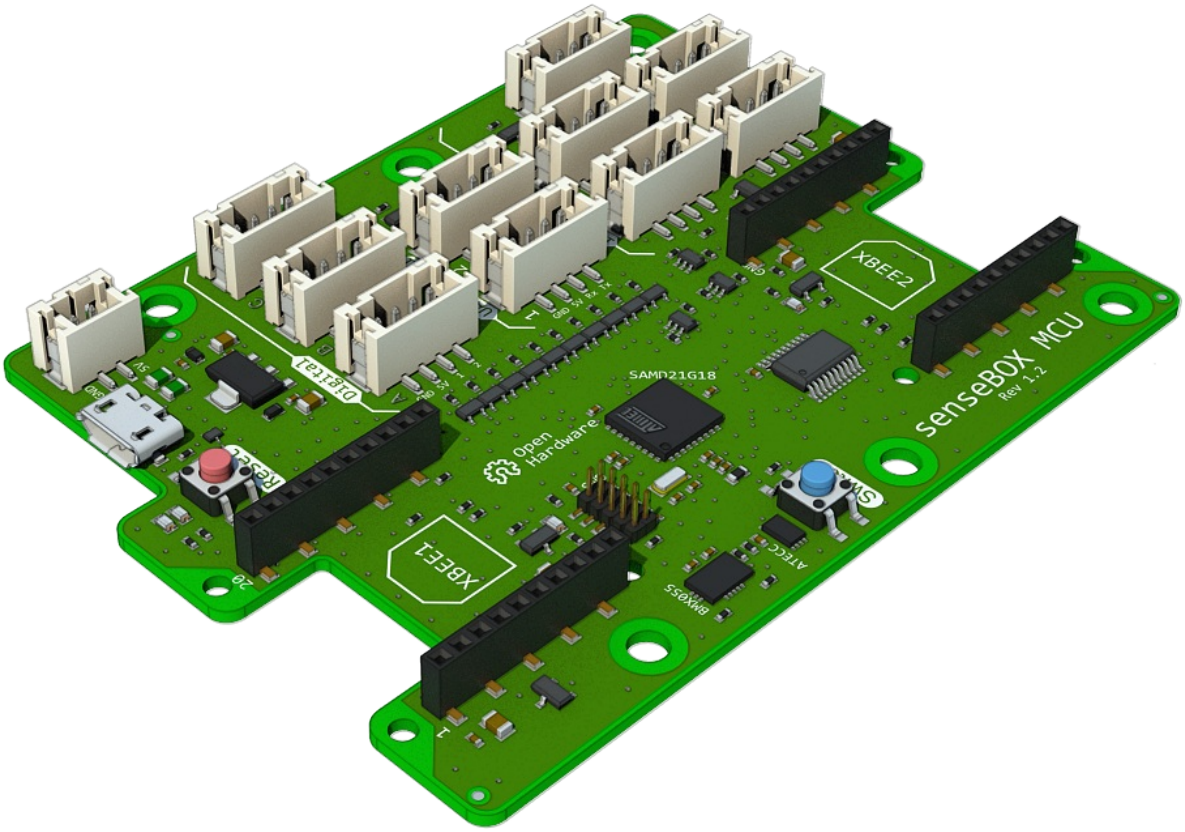
[14]. See *5.1.4.6 Micro-SD Card* ↩

[15]. See *5.1.4.7 GPS* ↩

[1]. https://forum.sensebox.de/ ↩

senseBox MCU

The senseBox-Microcontroller (MCU) is designed and developped for the special needs of a senseBox. Therefore the microcontroller is focussed on especially the following three features: speed, low in energy usage and a big program storage.
However, if you want to participate in openSenseMap the senseBox team offers you a guideline to program it without prior knowledge in coding. For advanced programmer the Arduino IDE can be used to access the board and create individual projects.



The senseBox MCU

Technical Specifications

Processor

The processor is based on a ARM Cortex-M0+ processor form the SAM D21 family by Microchip.

Interface

Sensors and actors can be activated using standard interfaces like I2C, UART and digitale I/Os with a robust JST-Connecting system (5V tolerant).

Data transmission

Using the both Bee compatible sockets, UART or SPI models can be offered. Therefore, real time data transmission via Wifi, LAN, or LoRa is offered as well as saving data on a Mikro-SD card.

Features

1. Crypto Authentication for OTA (Over the Air)
2. Firmware-Upgrades using the ATECC608A by Microchip
3. Integrierted BMX055 sensor by Bosch, for measuring accelaration, affinity and orientation
4. USB CDC+MSC Bootloader (Arduino compatible)
5. Interfaces: I2C = 5 (more using a I2C Hub) | 2 UART | 6 analoge digital digitale IOs

Bees

There are four different Bees which you can add to your senseBox to save data or upload it to the openSenseMap.

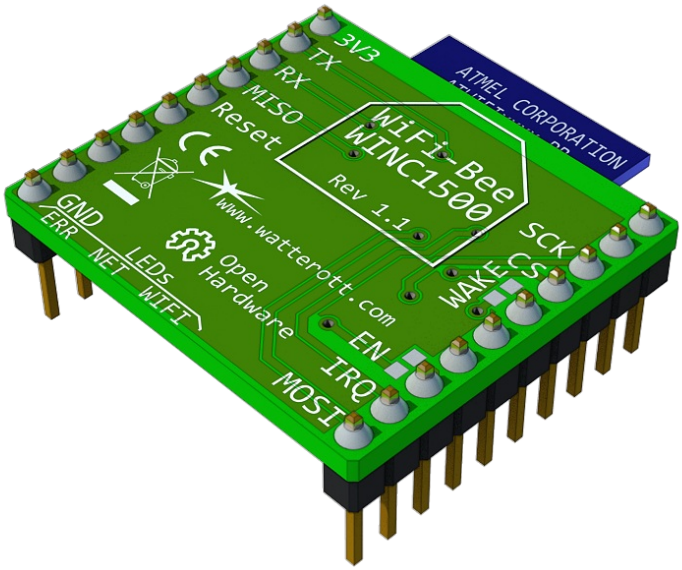
- Wifi-Bee[1]
- Ethernet-Bee[2]
- SD-Bee[3]
- LoRa-Bee[4]

1. See *5.1.2.1 Wifi-Bee* ↩
2. See *5.1.2.2 LAN-Bee* ↩
3. See *5.1.2.3 mSD-Bee* ↩
4. See *5.1.2.4 LoRa-Bee* ↩

Wifi-Bee

The Wifi-Bee is the connector between the senseBox and the internet. The data of the senseBox is transmitted via Wifi to the existing network. The Wifi-bee is based on the ATWINC1500 microchip by Atmel which has a very low energy consumption and a long range.

Some of our WINC1500 WiFi Bees may have outdated firmware (version 19.4.4) installed. This can lead to transmission problems. If these problems occur with you, please visit this website to refresh the firmware.



Wifi-Bee

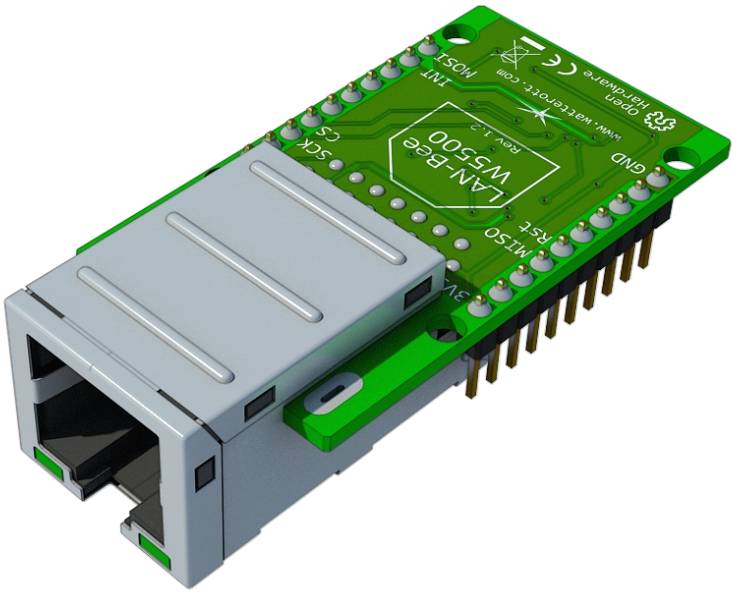
Technical Details

- "Plug-in-and-Go" senseBox compatibel
- Single-band 2.4GHz b/g/n
- Operating voltage: 3.0V to 4.2V

- Serial host interface: SPI
- Security protocols supported: WPA/WPA2 Personal, TLS, SSL
- Network services: DHCP, DNS, TCP/IP (IPv4), UDP, HTTP, HTTPS
- Name: WINC1500
- Measurements: 24mm x 25mm x 9mm
- Weight: 3,5 g

Ethernet-Bee

This Bee connects the senseBox with your router. The data of your senseBox will be transmitted using a ethernet cable to your router. The bee is based on the W5500 Mikrochip by Wiznet which allows a high ethernet data transmission rate.



Ethernet Bee

Technical Information

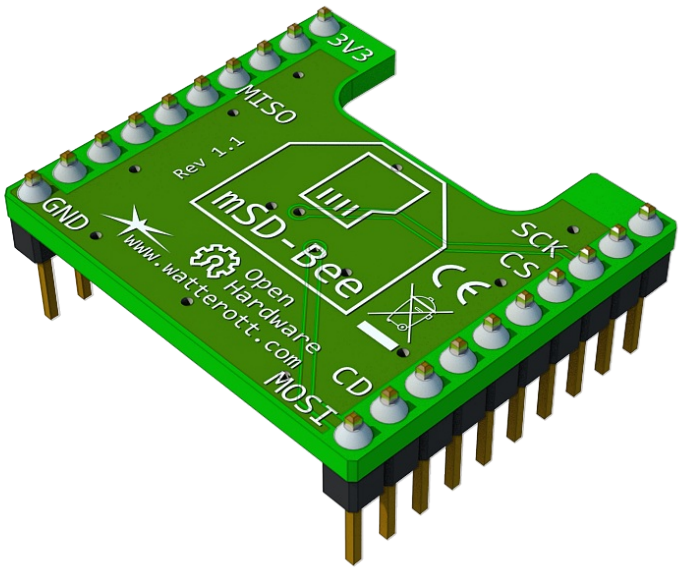
- "plug-in-and-go" senseBox compatible
- 3.3V operating voltage with 5V I/O signal tolerance
- Indication: W5500
- Measurements: 46mm x 25mm x 12mm
- Weigt: 9.2 g

Tipps and Hints

The Ethernet Bee will be delivered without Ethernet cable. We prefer to use flat cables, which can be also pulled below windows and doors.
Especially in combination with Power over Ethernet (PoE) the Ethernet bee is an interesting option.

mSD-Bee

With the SD-Bee, the data of the senseBox can be stored on an SD card. So you can measure, even if there is no Internet connection in the vicinity of the senseBox.



microSD-Bee

Technical Details

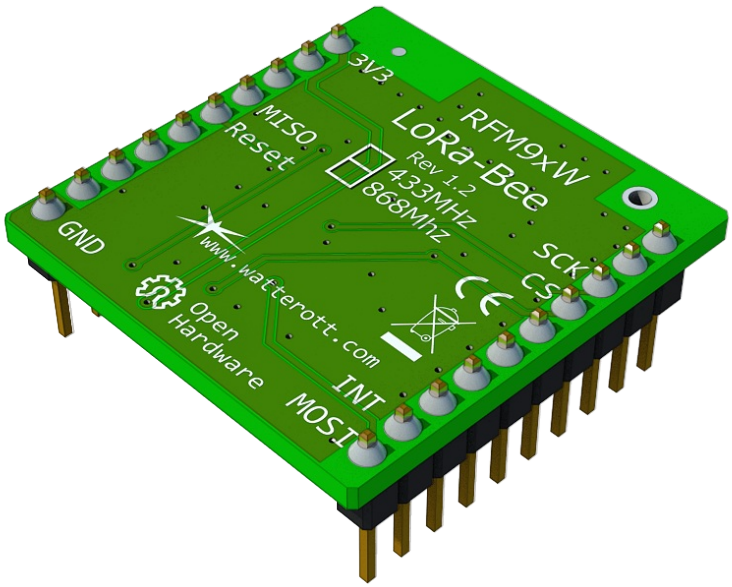
- "Plug-in-and-Go" senseBox compatibel
- Port for miniSD-card
- Name: mSD-Bee
- Measurements:24mm x 21mm x 9mm
- Weight:2,4 g

Information

Attention: The SD-bee is deliverd without a SD-card if you order in the senseBox-Shop.

LoRa-Bee

Use the LoRa-Inferface to upload your data on the openSenseMap. The LoRa WAN-Bee-Modul is a low energy and cost free option to upload your data unsing the LoRa-Radio-Standard. Therefore existing LoRa-Networks such as TheThingsNetwork are used for data transmission. The necessary infrstructure is provided by the community of TheThingsNetwork and available in more and more regions.



Lora Bee

Technical Information

- HopeRF RFM95W/RFM96W LoRa Transceiver
- LoRa-Bee 868 / 915 MHz uses RFM95W (SX1276 compatible)
- LoRa-Bee 433 / 470 MHz uses RFM96W (SX1276 compatoble)
- SPI interface
- Indication: RFN9xW
- Measurements: 46mm x 25mm x 12mm
- Weight: 1,1 g

Information

Please check if your area is already covered by LoRa before you get your senseBox with LoRa Bee: https://www.thethingsnetwork.org/community#list-communities-map

Attention: Due to the increased complexity of the installation, we recommend the LoRa module except advanced users of open hardware

Upload via LoRaWAN

It is possible to load sensor data via LoRaWAN ™ by the [TheThingsNetwork] (https://thethingsnetwork.org) (TTN) to the openSenseMap. LoRa is an increasingly popular radio standard, which is similar to WiFi. It allows digital data transmission in an IP network, but provides notable different features including:

- Data throughput: 300 - 3000 Bit/s
- Range: up to 15km

TTN is one of several projects that are related to the radio hardware Infrastructure implemented for the IP network. Whereby registered devices can be connected to the internet

Users can add Gateways as well as Nodes to the network.

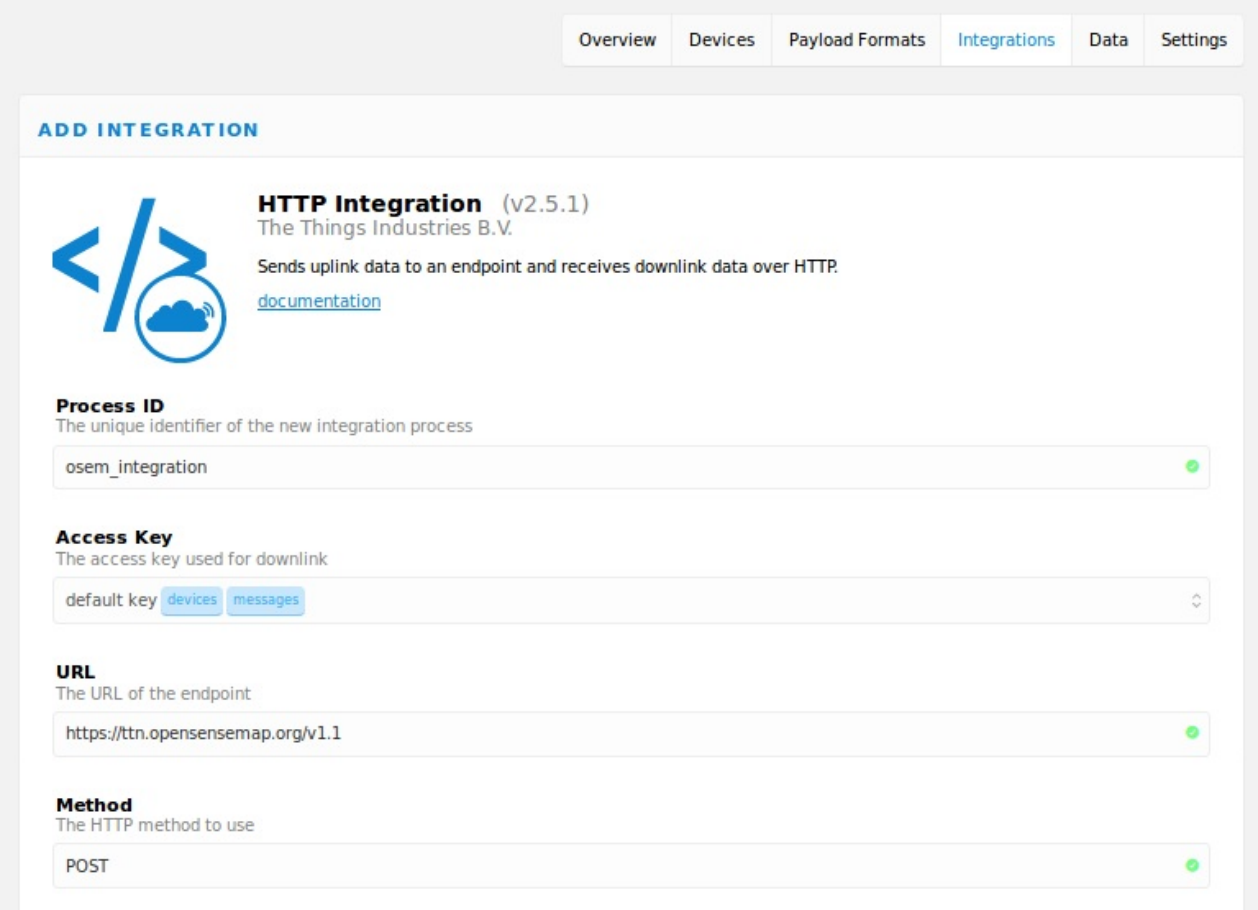
TTN openSenseMap Integration

The openSenseMap provides a direct integration into the TTN network, which simplifies the configuration. You therefor need to create an account on [TheThingsNetwork] (https://thethingsnetwork.org).

Registration in TTN Console

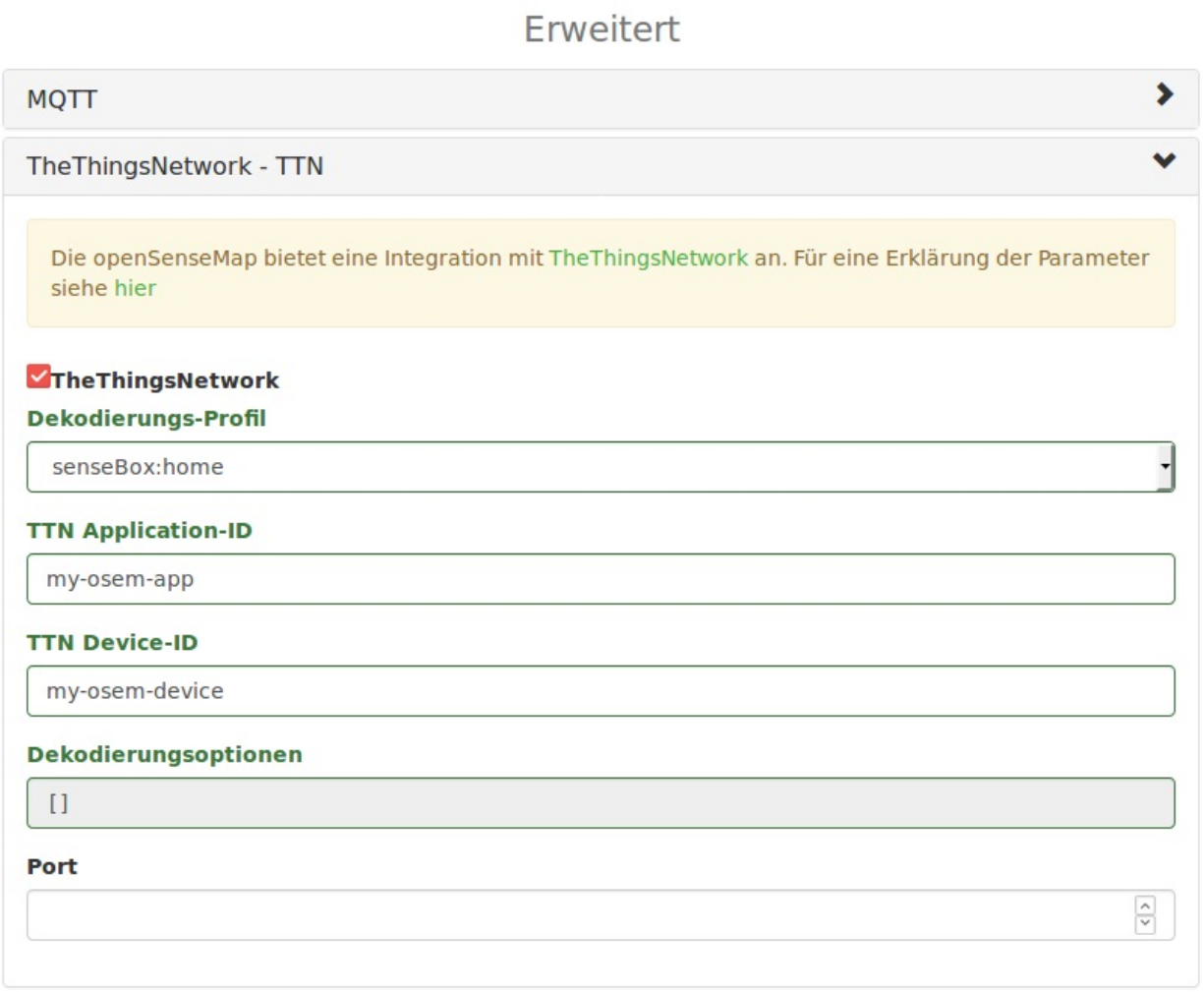
To integrate a device in to the TTN you have to first register an Application and a Device on the thethingsnetwork.org[1] Here you receive a `app_id` and a `dev_id`.

For the registered application, the HTTP integration must be activated under https://console.thethingsnetwork.org/applications/DEINE_APPID/integrations/create/http-ttn. To transmit messages from devices via `POST` to `https://ttn.opensensemap.org/v1.1`, you have to configurate this The authorization-field can stay empty.



ttnconsole

For the data transfer to openSenseMap, the `app_id` and `dev_id` must be included for the registration on openSenseMap in the TTN configuration. In addition, a suitable decoding profile must be configured. Which determines how the - because of the low bandwidth as raw bytes transmitted - data should be interpreted as measurements.



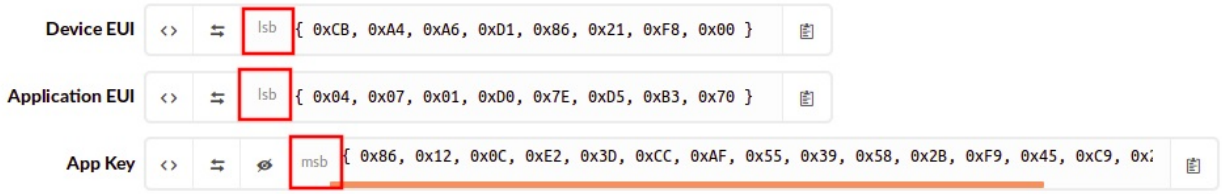
osemregister

Optionally you can indicate a port in the field `port`, on which the transmitter can send his data to the TTN. So you can use the same `app_id` and `dev_id` for multiple sensor stations.

Arduino Sketch

This could be an Arduino sketch that lets you send data to the openSenseMap over the TTN network.

> Important: You have to paste your recently crated Application-EUI, Device-EUI and the App-Key in the sketch. Please do this in the first line of the programmecode where 'INSERT YOUR ID HERE' is indicated.
>
> Mind that you have chosen the Device-EUI the Application-EUI the lsb-Format as well as the App-Key and the msb-Format on the TTN-Homepage.



Selected ID's and Keys

▼ **Arduino Sketch für senseBoxMCU**

```
/*******************************************************************************
 * Copyright (c) 2015 Thomas Telkamp and Matthijs Kooijman.
 * Edited by: senseBox
 *
 *******************************************************************************/
#include <LoraMessage.h>
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include <senseBoxIO.h>

#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>
#include <HDC100X.h>
#include <Makerblog_TSL45315.h>
#include <SDS011-select-serial.h>
#include <VEML6070.h>

// Number of serial port the SDS011 is connected to. Either Serial1 or Serial2
#define SDS_UART_PORT (Serial1)

//Load sensors / instances
Makerblog_TSL45315 TSL = Makerblog_TSL45315(TSL45315_TIME_M4);
HDC100X HDC(0x40);
Adafruit_BMP280 BMP;
VEML6070 VEML;
SDS011 SDS(SDS_UART_PORT);

bool hdc, bmp, veml, tsl = false;

//measurement variables
float temperature = 0;
float humidity = 0;
float pm10 = 0;
float pm25 = 0;
double tempBaro, pressure;
uint32_t lux;
uint16_t uv;

// This EUI must be in little-endian format, so least-significant-byte
// first. When copying an EUI from ttnctl output, this means to reverse
// the bytes. For TTN issued EUIs the last bytes should be 0xD5, 0xB3,
// 0x70.
static const u1_t PROGMEM APPEUI[8]={ 'Your APP ID Here' };
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}

// This should also be in little endian format, see above.
static const u1_t PROGMEM DEVEUI[8]={ 'YOUR DEVICE ID HERE '};
void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}

// This key should be in big endian format (or, since it is not really a
```

```
// number but a block of memory, endianness does not really apply). In
// practice, a key taken from ttnctl can be copied as-is.
// The key shown here is the semtech default key.
static const u1_t PROGMEM APPKEY[16] = { 'YOUR APP KEY HERE '};
void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}

static osjob_t sendjob;

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 300;

// Pin mapping
const lmic_pinmap lmic_pins = {
    .nss = PIN_XB1_CS,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = LMIC_UNUSED_PIN,
    .dio = {PIN_XB1_INT, PIN_XB1_INT, LMIC_UNUSED_PIN},
};

void checkI2CSensors() {
  byte error;
  int nDevices = 0;
  byte sensorAddr[] = {41, 56, 57, 64, 118};
  tsl = false; veml = false; hdc = false; bmp = false;
  Serial.println("\nScanning...");
  for (int i = 0; i < sizeof(sensorAddr); i++) {
    Wire.beginTransmission(sensorAddr[i]);
    error = Wire.endTransmission();
    if (error == 0) {
      nDevices++;
      switch (sensorAddr[i])
      {
        case 0x29:
          Serial.println("TSL45315 found.");
          tsl = true;
          break;
        case 0x38: // &0x39
          Serial.println("VEML6070 found.");
          veml = true;
          break;
        case 0x40:
          Serial.println("HDC1080 found.");
          hdc = true;
          break;
        case 0x76:
          Serial.println("BMP280 found.");
          bmp = true;
          break;
      }
    }
    else if (error == 4)
    {
      Serial.print("Unknown error at address 0x");
      if (sensorAddr[i] < 16)
        Serial.print("0");
      Serial.println(sensorAddr[i], HEX);
    }
  }
  if (nDevices == 0) {
    Serial.println("No I2C devices found.\nCheck cable connections and press Reset.");
    while(true);
  } else {
    Serial.print(nDevices);
    Serial.println(" sensors found.\n");
  }
  //return nDevices;
}

void onEvent (ev_t ev) {
    senseBoxIO.statusGreen();
    Serial.print(os_getTime());
```

```
    Serial.print(": ");
    switch(ev) {
        case EV_SCAN_TIMEOUT:
            Serial.println(F("EV_SCAN_TIMEOUT"));
            break;
        case EV_BEACON_FOUND:
            Serial.println(F("EV_BEACON_FOUND"));
            break;
        case EV_BEACON_MISSED:
            Serial.println(F("EV_BEACON_MISSED"));
            break;
        case EV_BEACON_TRACKED:
            Serial.println(F("EV_BEACON_TRACKED"));
            break;
        case EV_JOINING:
            Serial.println(F("EV_JOINING"));
            break;
        case EV_JOINED:
            Serial.println(F("EV_JOINED"));

            // Disable link check validation (automatically enabled
            // during join, but not supported by TTN at this time).
            LMIC_setLinkCheckMode(0);
            break;
        case EV_RFU1:
            Serial.println(F("EV_RFU1"));
            break;
        case EV_JOIN_FAILED:
            Serial.println(F("EV_JOIN_FAILED"));
            break;
        case EV_REJOIN_FAILED:
            Serial.println(F("EV_REJOIN_FAILED"));
            break;
            break;
        case EV_TXCOMPLETE:
            Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
            if (LMIC.txrxFlags & TXRX_ACK)
              Serial.println(F("Received ack"));
            if (LMIC.dataLen) {
              Serial.println(F("Received "));
              Serial.println(LMIC.dataLen);
              Serial.println(F(" bytes of payload"));
            }
            // Schedule next transmission
            os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTERVAL), do_send);
            break;
        case EV_LOST_TSYNC:
            Serial.println(F("EV_LOST_TSYNC"));
            break;
        case EV_RESET:
            Serial.println(F("EV_RESET"));
            break;
        case EV_RXCOMPLETE:
            // data received in ping slot
            Serial.println(F("EV_RXCOMPLETE"));
            break;
        case EV_LINK_DEAD:
            Serial.println(F("EV_LINK_DEAD"));
            break;
        case EV_LINK_ALIVE:
            Serial.println(F("EV_LINK_ALIVE"));
            break;
         default:
            Serial.println(F("Unknown event"));
            break;
    }
}

void do_send(osjob_t* j){
    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
```

```
    } else {
        LoraMessage message;

        //-----Temperature-----//
        //-----Humidity-----//
        if (hdc) {
          Serial.print("Temperature: ");
          temperature = HDC.getTemp();
          Serial.println(temperature);
          message.addUint16((temperature + 18) * 771);
          delay(2000);

          Serial.print("Humidity: ");
          humidity = HDC.getHumi();
          Serial.println(humidity);
          message.addHumidity(humidity);
        }
        delay(2000);

        if (bmp) {
          float altitude;
          tempBaro = BMP.readTemperature();
          pressure = BMP.readPressure()/100;
          altitude = BMP.readAltitude(1013.25); //1013.25 = sea level pressure
          Serial.print("Pressure: ");
          Serial.println(pressure);
          message.addUint16((pressure - 300) * 81.9187);
          delay(2000);
        }

        if (tsl) {
          //-----Lux-----//
          Serial.print("Illuminance: ");
          lux = TSL.readLux();
          Serial.println(lux);
          message.addUint8(lux % 255);
          message.addUint16(lux / 255);
          delay(2000);
        }

        if (veml) {
          //-----UV intensity-----//
          Serial.print("UV: ");
          uv = VEML.getUV();
          Serial.println(uv);
          message.addUint8(uv % 255);
          message.addUint16(uv / 255);
          delay(2000);
        }

        uint8_t attempt = 0;

        while (attempt < 5) {
          bool error = SDS.read(&pm25, &pm10);
          if (!error) {
            Serial.print("PM10: ");
            Serial.println(pm10);
            message.addUint16(pm10 * 10);
            Serial.print("PM2.5: ");
            Serial.println(pm25);
            message.addUint16(pm25 * 10);
            break;
          }
          attempt++;
        }

        // Prepare upstream data transmission at the next possible time.
        LMIC_setTxData2(1, message.getBytes(), message.getLength(), 0);
        Serial.println(F("Packet queued"));
    }
    // Next TX is scheduled after TX_COMPLETE event.
}
```

```
void setup() {
    Serial.begin(9600);
    delay(10000);

    // RFM9X (LoRa-Bee) in XBEE1 Socket
    senseBoxIO.powerXB1(false); // power off to reset RFM9X
    delay(250);
    senseBoxIO.powerXB1(true);  // power on

    // init I2C/wire library
    Wire.begin();

    // Sensor initialization
    Serial.println(F("Initializing sensors..."));
    SDS_UART_PORT.begin(9600);
    checkI2CSensors();

    if (veml)
    {
      VEML.begin();
      delay(500);
    }
    if (hdc)
    {
      HDC.begin(HDC100X_TEMP_HUMI, HDC100X_14BIT, HDC100X_14BIT, DISABLE);
      HDC.getTemp();
    }
    if (tsl)
    {
      TSL.begin();
    }
    if (bmp)
    {
      BMP.begin(0x76);
    }
    Serial.println(F("Sensor initializing done!"));
    Serial.println(F("Starting loop in 3 seconds."));
    delay(3000);

    // LMIC init
    os_init();
    // Reset the MAC state. Session and pending data transfers will be discarded.
    LMIC_reset();

    // Start job (sending automatically starts OTAA too)
    do_send(&sendjob);
}

void loop() {
    os_runloop_once();
}
```

Decoding Profile

A decoding-profile fitting to the measuring data has to be selected and defined for a box. The decoding profile selection is based on the encoding of the messages on the Microcontroller. And whether in the TTN a payload function has been set dependent.

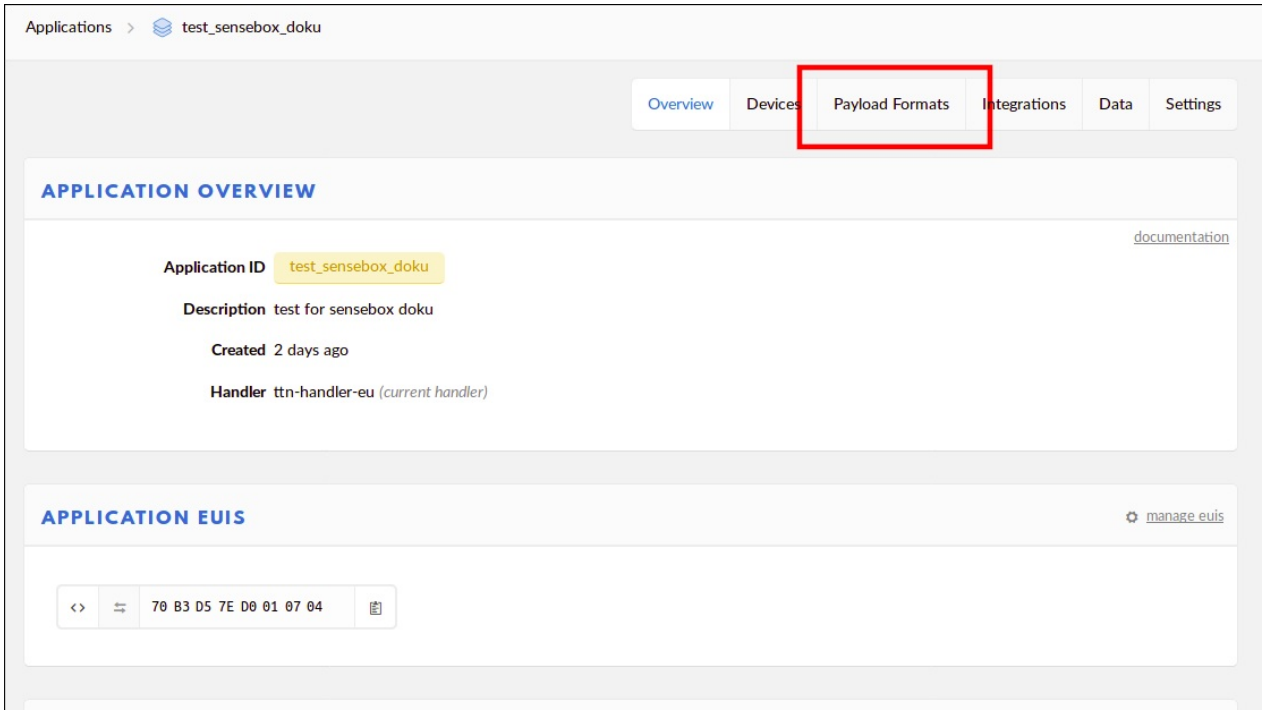
- For the senseBox: home (without extensions) the `senseBox: home` profile be used.
- If the measurements will be encoded on the LoRa node using the `lora-serialization` library , the `lora-serialization` profile should be used.
- The `json` profile supports any other encodings, if one Payload function in the TTN Console decodes the messages appropriately.

The following explains how to configure the supported profiles:

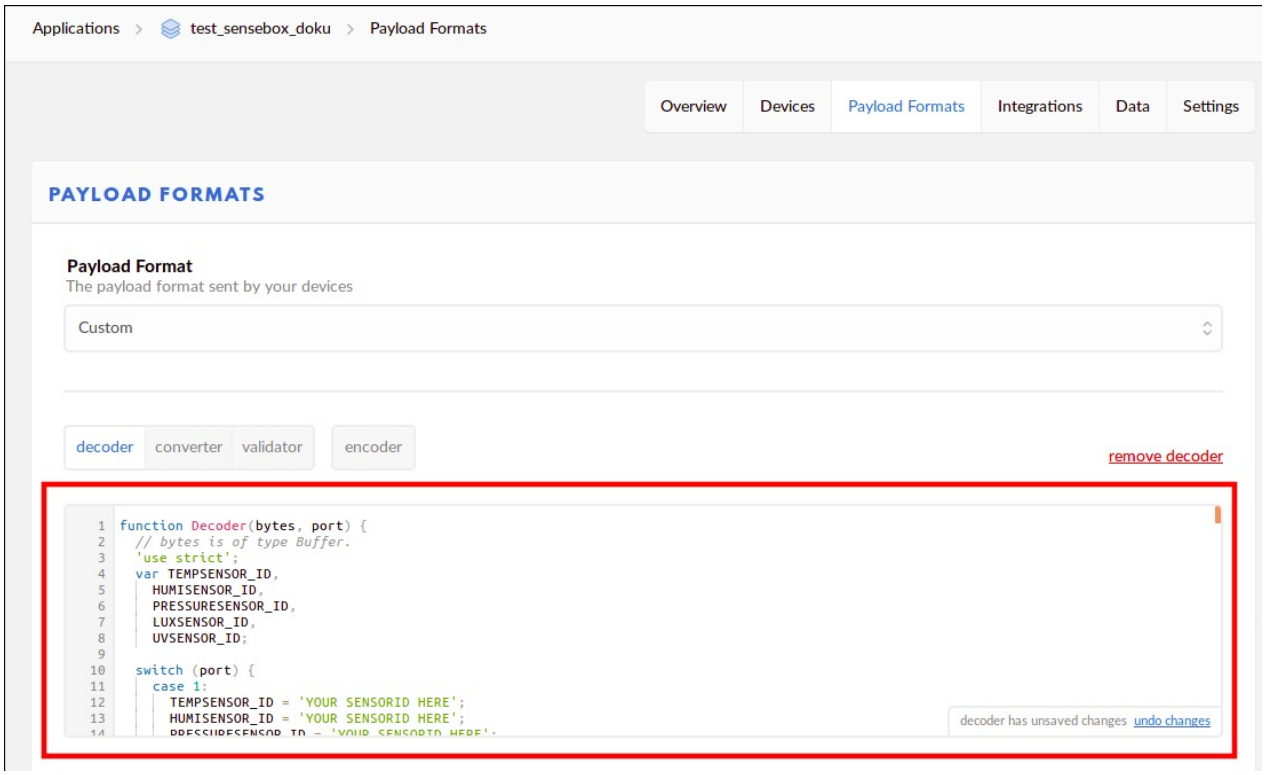
`sensebox/home`

This profile is tailored to the sensors supplied with the senseBox: home. Besides the specification `sensebox / home` under `profile` there is no further configuration necessary.
This works only without the fine dust sensors(PM2.5 und PM10)

In addition to the Arduino Sketch, you'll need to set up a decoder on the TTN homepage so that your metrics are sent to the openSenseMap in the correct format.



Navigate to Payload Formats in the Overview window



The decoder must now be inserted in the text box

▼ Decoder für das TTN

> Important: Here you have to add your sensor ID's.

```
function Decoder(bytes, port) {
  // bytes is of type Buffer.
  'use strict';
  var TEMPSENSOR_ID = 'YOUR TEMPERATURE SENSOR ID HERE',
    HUMISENSOR_ID = 'YOUR HUMIDITY SENSOR ID HERE',
    PRESSURESENSOR_ID = 'YOUR PRESSURE SENSOR ID HERE ',
    LUXSENSOR_ID = 'YOUR LUXSENSOR ID HERE ',
    UVSENSOR_ID = 'YOUR UV SENSOR ID HERE';

  var bytesToInt = function (bytes) {
    var i = 0;
    for (var x = 0; x < bytes.length; x++) {
      i |= +(bytes[x] << (x * 8));
    }
    return i;
  };

  var uint8 = function (bytes) {
    if (bytes.length !== uint8.BYTES) {
      throw new Error('int must have exactly 1 byte');
    }
    return bytesToInt(bytes);
  };
  uint8.BYTES = 1;
```

```
var uint16 = function (bytes) {
  if (bytes.length !== uint16.BYTES) {
    throw new Error('int must have exactly 2 bytes');
  }
  return bytesToInt(bytes);
};
uint16.BYTES = 2;

var humidity = function (bytes) {
  if (bytes.length !== humidity.BYTES) {
    throw new Error('Humidity must have exactly 2 bytes');
  }

  var h = bytesToInt(bytes);
  return h / 1e2;
};
humidity.BYTES = 2;

var decode = function (bytes, mask, names) {

  var maskLength = mask.reduce(function (prev, cur) {
    return prev + cur.BYTES;
  }, 0);
  if (bytes.length < maskLength) {
    throw new Error('Mask length is ' + maskLength + ' whereas input is ' + bytes.length);
  }

  names = names || [];
  var offset = 0;
  return mask
    .map(function (decodeFn) {
      var current = bytes.slice(offset, offset += decodeFn.BYTES);
      return decodeFn(current);
    })
    .reduce(function (prev, cur, idx) {
      prev[names[idx] || idx] = cur;
      return prev;
    }, {});
};

var bytesToSenseBoxJson = function (bytes) {
  var json;

  try {
    json = decode(bytes,
      [
        uint16,
        humidity,
        uint16,
        uint8,
        uint16,
        uint8,
        uint16
      ],
      [
        TEMPSENSOR_ID,
        HUMISENSOR_ID,
        PRESSURESENSOR_ID,
        LUXSENSOR_ID + '_mod',
        LUXSENSOR_ID + '_times',
        UVSENSOR_ID + '_mod',
        UVSENSOR_ID + '_times'
      ]);

    //temp
    json[TEMPSENSOR_ID] = parseFloat(((json[TEMPSENSOR_ID] / 771) - 18).toFixed(1));

    //hum
    json[HUMISENSOR_ID] = parseFloat(json[HUMISENSOR_ID].toFixed(1));

    // pressure
```

```
      if (json[PRESSURESENSOR_ID] !== '0') {
        json[PRESSURESENSOR_ID] = parseFloat(((json[PRESSURESENSOR_ID] / 81.9187) + 300).toFixed(1));
      } else {
        delete json[PRESSURESENSOR_ID];
      }

      // lux
      json[LUXSENSOR_ID] = (json[LUXSENSOR_ID + '_times'] * 255) + json[LUXSENSOR_ID + '_mod'];
      delete json[LUXSENSOR_ID + '_times'];
      delete json[LUXSENSOR_ID + '_mod'];

      // uv
      json[UVSENSOR_ID] = (json[UVSENSOR_ID + '_times'] * 255) + json[UVSENSOR_ID + '_mod'];
      delete json[UVSENSOR_ID + '_times'];
      delete json[UVSENSOR_ID + '_mod'];

    } catch (e) {
      json = { payload: bytes };
    }

    return json;
  };

  return bytesToSenseBoxJson(bytes);
}
```

`lora-serialization`

The `lora-serialization` profile can accept almost any data, even sensor stations, which have a special sensor configuration. For this we use the [`lora-serialization`] (https://github.com/thesolarnomad/lora-serialization) Library, which provides a unified encoding on the microcontroller, and Decoding on the other end of the line.

The encodings `temperature` , `humidity` , `unixtime` , `uint8` and `uint16` are supported, which need to be indicated per sensor under decoding options . The assignment of the sensor can be made via one of the properties `sensor_id` , `sensor_title` , `sensor_unit` , `sensor_type` .

An example of two sensors looks like this:

```
[
  { "decoder": "temperature", "sensor_title": "Temperatur" },
  { "decoder": "humidity", "sensor_unit": "%" }
]
```

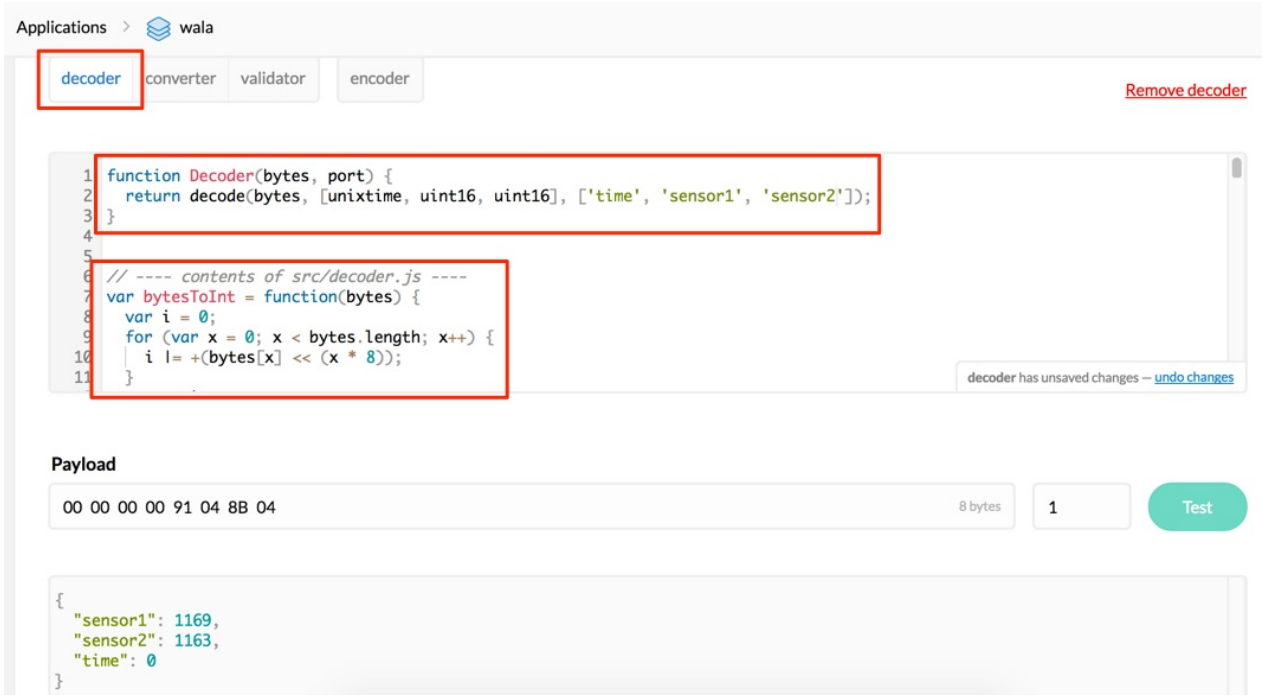
> Information: The order of the sensors must be the same here as well as on the Arduino and the >openSenseMap!

If a `unixtime` decoder is specified, its timestamp will be used for all of the following measurements. Otherwise, the moment is used in which the first gateway receives the message. Example:

```
[
  { "decoder": "unixtime" },
  { "decoder": "temperature", "sensor_title": "Temperatur" }
]
```

`json` - Decoding mit TTN Payload Function

If the `lora-serialization` library is not available, measurements can still get decoded on the TTN side by means of a Payload Function , so that any data formats are supported here.



In the TTN Console, a payload function must be defined

The resulting JSON must be compatible with that of theopenSenseMap-API verstandenen Measurement Formaten sein. A simple example:

```
{ "sensor_id1": "value1, "sensor_id2: "value2" }
```

An example of this is indicated for you you above[1].

On the side of the openSenseMap no configuration is necessary.

---

[1]. https://console.thethingsnetwork.org/ ↩

[1]. See *5.1.2.4 LoRa-Bee* > *decoder* ↩

Sensors

With sensors you can measure different environmental phenomena and thus recognize, observe and analyze.

In our senseBox-Shop[1] you will find a list of sensors we work with and therefore have written small instructions. For these sensors we offer you here information, but also help with connecting to the senseBox.

Here you will find information about the following sensors, just click on the name of the sensor and you will be redirected to a separate page

- Temperature & air humidity (HDC1080)[1]
- Air pressure & temperature[2]
- Exposure and UV[3]
- Fine dust[4]

Of course, you can also connect any other sensor you know to the senseBox. However then you are asked as a tinkerer to figure out the wiring and programming ;) We would love to see you sharing information about your sensor and how you connected it to your senseBox so we can add it to our instruction.

---

1. https://sensebox.kaufen/ ↩

1. See *5.1.3.1 Temperature & Humidity (HDC1080)* ↩

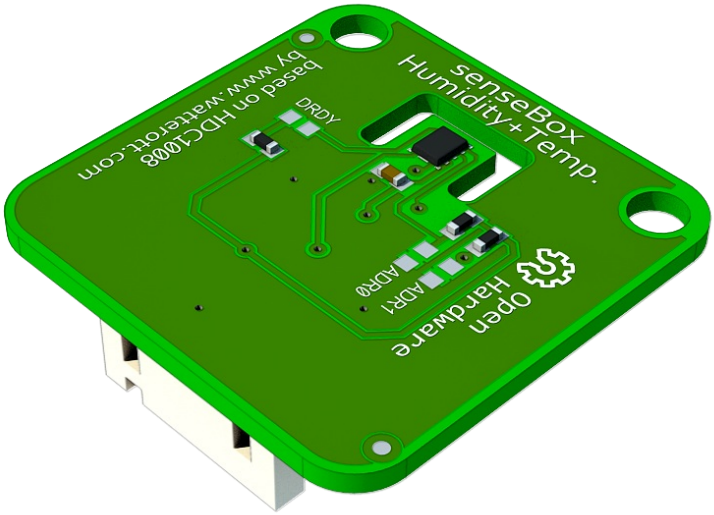
2. See *5.1.3.2 Air Pressure & Temperature* ↩

3. See *5.1.3.3 Light and UV Intensity* ↩

4. See *5.1.3.4 Fine Dust* ↩

Temperature- and Air Humidity Sensor (HDC1080)

The HDC1080 is a digital humidity and temperature sensor. The sensor has a high accuracy and a very low power consumption and thus fits perfectly with the senseBox. The sensors are factory calibrated and can be used directly.



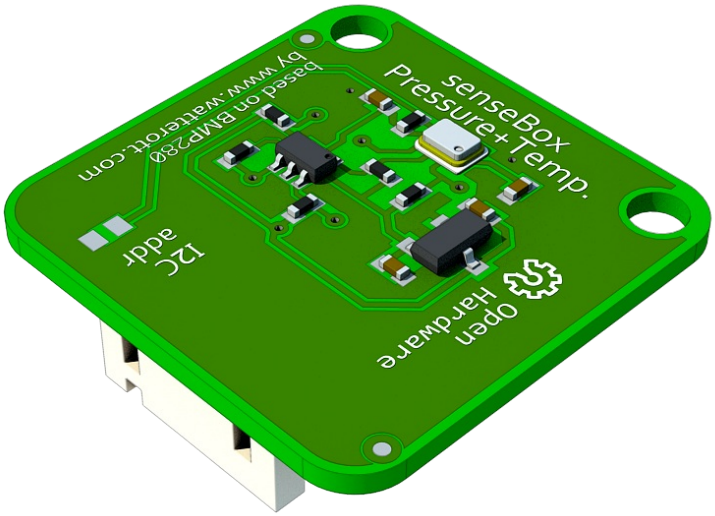
Temperature and Air Humidity Sensor

Technical Details

- Relative Humidity (RH) Operating range 0% to 100%
- 14-bit Measurement Resolution
- Relative humidity accuracy ± 4%
- Temperature accuracy ± 0.2 ° C
- 2100nA Sleep Mode Current
- Operating voltage 2.7 V to 5.5 V
- I2C interface
- "Plug-in-and-Go" senseBox compatible
- Average Supply Current: 710nA @ 1sps, 11bit RH Measurement 1.3µA @ 1sps, 11bit RH and Temperature Measurement

Airpressure- and Temperature Sensor

This sensor measures air pressure and temperature and is based on the BMP280 sensor from Bosch.



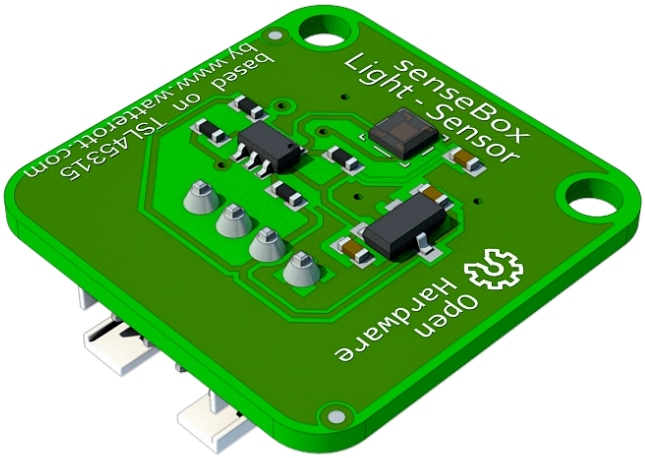
Airpressure- and Temperature Sensor

Technical Detail

- Dimensions: 25mm x 25mm x 9mm
- Weight: 2.4 g
- "Plug-in-and-Go" senseBox compatible
- Operating pressure 300 to 1100 hPa
- Relative precision ± 0.12 hPa
- Absolute precision ± 1 hPa
- Operating power 2.7µA at 1Hz sampling frequency

Illumination and UV-Radiation Sensor

Two sensors are put together on this senseBox component. The light intensity is measured with the TSL45315 sensor from AMS-TAOS. This sensor detects the light conditions similar to the human eye and outputs the brightness values directly in lux, with a large dynamic range (3 lux to 220k lux). The second sensor is a Vishay VEML6070 Ultraviolet (UV) light sensor. This converts the intensity of the UV light of the sun into digital data. The sensor has excellent UV sensitivity and linearity via Filtron ™ technology. It has a good UV radiation measurement even with long solar UV exposure and can compensate for excellent temperature fluctuations.



Illumination and UV-Radiation

Technical Details

Exposure Sensor

- 3,3V - 5V tolerant I2C/TWI Interface
- Input voltage range: 3,3V - 5V
- On-board 2,5V voltage regulator
- On-board levelconverter

UV-Sensor

- Operating voltage: 2,7V - 5,5V I2C Interface
- Supports confirmation function (Active Acknowledge-Function)
- Temperature compensation: -40°C to +85°C
- Software-switching of control for immunity with flickering fluorescent lamps

Measurement

- 25mm x 25mm x 9mm
- Weight: 2,5 g

Fine Dust Sensor

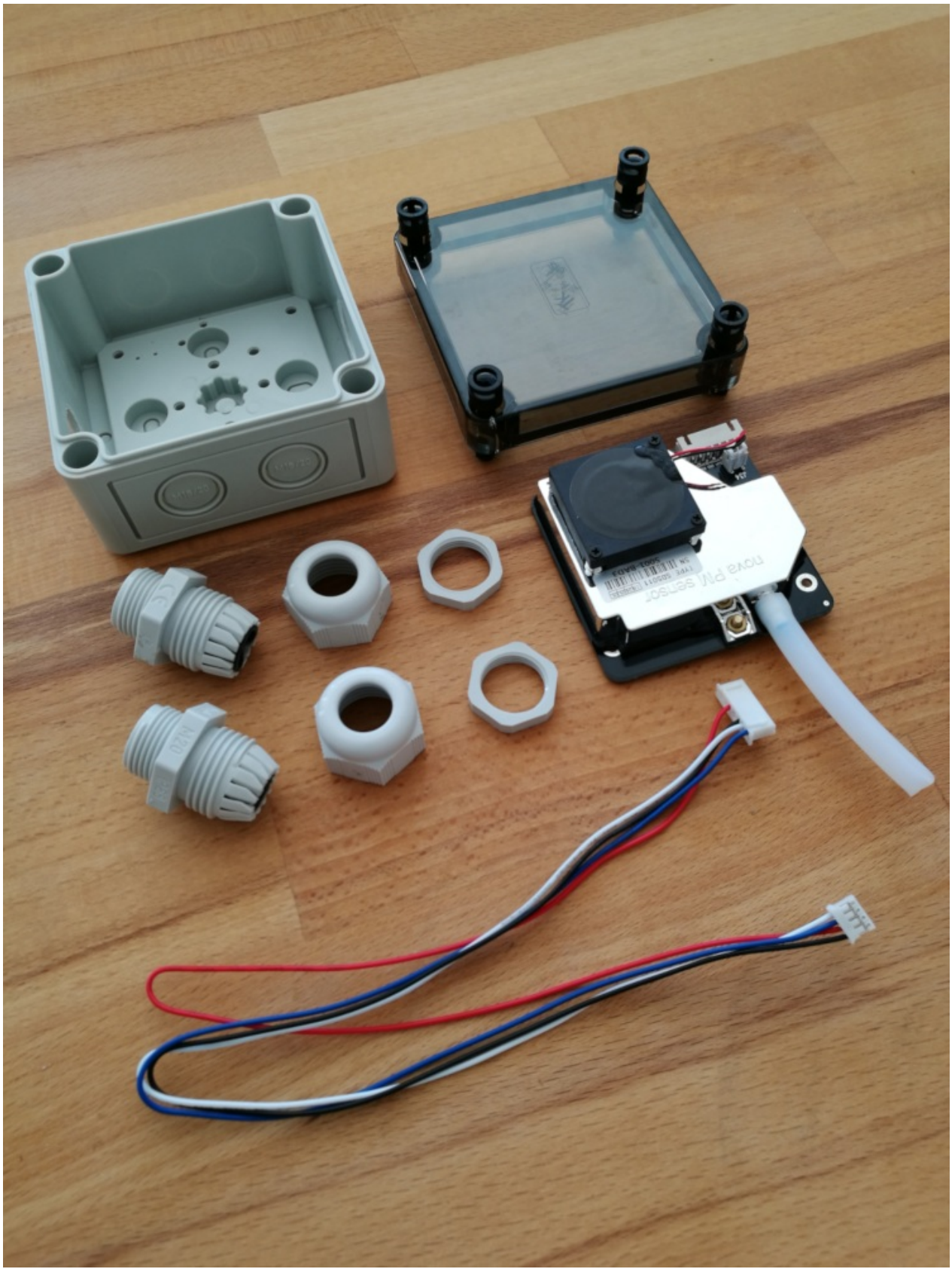
With this sensor SDS011 it is possible to determine the fine dust concentration in the air. The sensor outputs two values: the concentration of PM2.5 (particle < 2.5 µm) and PM10 (particle < 10 µm). This sensor is equipped with a small fan to suck in air. Inside is a laser that measures the number of particles together with a photodiode. The results of the measurements are given in µg / m³ (micrograms per cubic meter)

Fine dust sensor for PM10 und PM2.5

Technical Details

- Fast reaction time less than 10 seconds
- "Plug-in-and-Go" senseBox compatibel
- High resolution up to 0.3µg/m3
- Multiple scientific verification of data accuracy

Parts required for the Set-Up



Required (provided) Parts

- SDS011 fine dust sensor
- connecting wire

- an unit of teflon tube Ø = 6mm inside and Ø = 8mm outside
- Case
- Cable gland 16mm

Connection and Programming

With the supplied connection cable you can connect your fine dust sensor with the "UART / Serial" port of the senseBoxMCU. Once this is done, we can now initialize the sensor in the program code and have the first measured values output

> Make sure you have the latest board support package installed because you need the correct software libraries. How this works was shown in Step 2.

First, create an instance of the sensor. For this we create 2 variables in which we save our two readings for PM10 and PM2.5.

```
#include "SenseBoxMCU.h"
SDS011 my_sds(Serial1) // Serial1 indicates the serial port where you connected the sensor
float p10,p25
```

▼ setup() Funktion

The sensor should now start in the `setup()` -Function:

```
void setup(){
    // Initialise normal seriel Port
    Serial.begin(9600);
    while(!Serial);
    // Initialize the serial port where our sensor is connected
    Serial1.begin(9600);
    delay(5000);

}
```

▼ loop() Funktion

In the `loop ()` function we can use the command 'getPm10 ()' and 'getPm25 () to retrieve the currently measured fine dust values:

```
void loop(){
    // Assign variables to measured particulate matter values
    p10 = my_sds.getPm10();
    p25 = my_sds.getPm25();
    //Print values in the console
    Serial.println("P2.5: "+String(p25));
    Serial.println("P10:  "+String(p10));
    delay(1000);
}
```

Accessories

Here you will find all other accessory parts that can not be assigned to the sensors or Bees. These include parts such as GPS module, housing or radiation protection, which are not included as standard in the edu version of the senseBox. Nevertheless, to find out what they are useful for, you can still find information about these parts.

Overview

There are the following components offered and tested for you with the senseBox:

- Radiation protection[1]
- Housing[2]
- AC adapter and USB cable[3]
- LED display[4]
- HUB[5]
- Micro SD card[6]
- GPS[7]

---

1. See *5.1.4.1 Radiation Protection* ↩
2. See *5.1.4.2 Housing* ↩
3. See *5.1.4.3 Power Supply and USB Cable* ↩
4. See *5.1.4.4 LED-Display* ↩
5. See *5.1.4.5 Expander* ↩
6. See *5.1.4.6 Micro-SD Card* ↩
7. See *5.1.4.7 GPS* ↩

Radiation Protection

The radiation protection protects the outdoor transmitter for temperature + humidity (BMP280) from the weather. It serves as protection against precipitation and direct sunlight. It is easy to mount and has an opening for the sensor cables.

Advice for the Location of the Radiation Protection

- Choose a shady spot outdoors for radiation protection . (Direct sunshine falsifies the measured values).
- Please check wheter a transmission from the transmitter to the desired installation location is possible (in massive walls, especially with metal parts, the transmission range can get considerably reduced).

Mounting of the Protective Cover

- Turn off the protective cover clockwise from the bottom plate .
- The protective cover can be be screwed on a suitable surface for a secure fit.
- Choose a smooth, horizontal position.
- Pass a screw through the opening in the middle and screw the floorplate firmly.

Attachment to a Wall or to a Pole

- Remove the wall bracket from the case by sliding it down.
- Attach the wall bracket to the wall with the screws and dowels. Please mind the marking "UP"
- If you want to attach the wall bracket to a pole, you can also use the cable ties.
- Note: When mounting, please make sure that the protective cover can be easily inserted into the Wall bracket
- Put the protective cover with the base plate and the attached transmitter in the wall bracket from above.

Attachment of the Transmitter

- To secure the transmitter to the base plate, attach the Velcro strip with the double-sided tape to the base on the stand and on the back of the transmitter and fix the transmitter. Alternatively, you can also use the cable ties or work with hot glue.
- Guide the cables through the provided opening in the bottom plate.
- Place the protective cover on the base plate with the attached transmitter and turn it counterclockwise.

Care and Maintenance

- Clean the protective cover with a soft, slightly damp cloth. Do not use scouring or solvents.

Technical Data

Interior dimensions: Höhe 160 mm Interior diameter: 60 mm Case measurements: 95 x 102 (108) x 180 mm Weight: 163 g

Housing

The housing protects the microcontroller and some sensors from the weather.

The Housing

The transparent polycarbonate housing protects the senseBox MCU and the belonging sensors from the weather.

Measurements: 82mm x 162mm x 85mm Weight: 300 g

Power Supply and USB-Cable

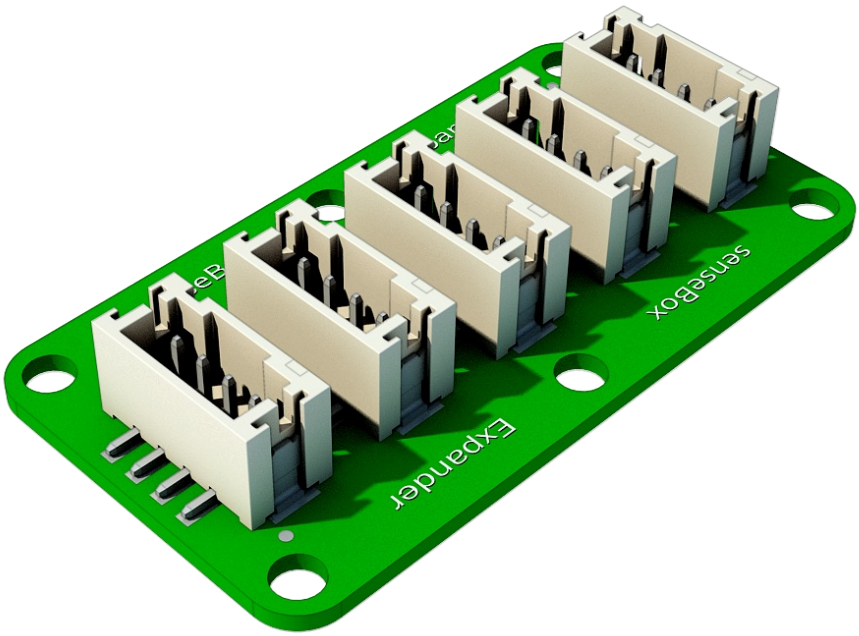
The senseBox is connected to a conventional micro USB cable and connected to the computer or to a power source

USB-cable

With the enclosed USB cable, the senseBox can be configured and connected to the computer. The power supply ensures the continuous power supply. The enclosed cable has a length of 1.5 meters.

LED-Display

Expander

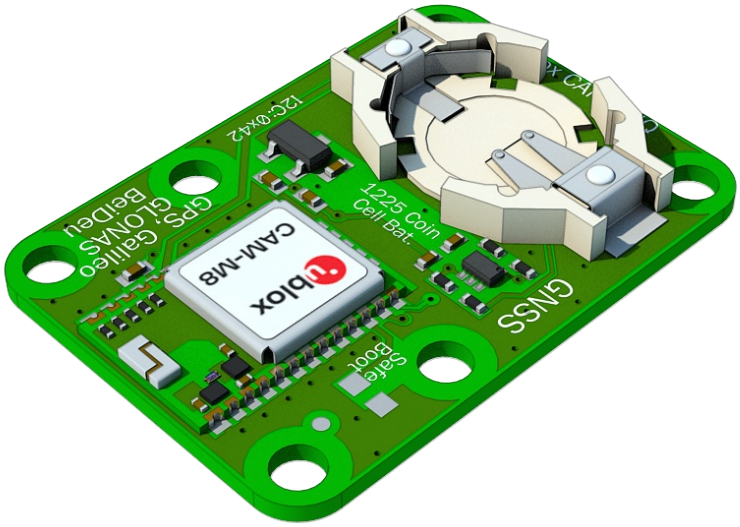


Expander

Micro-SD-Karte

GPS

Sorry, unfortunately we have not yet managed to document the GPS sensor. We hope to implement this as soon as possible



The GPS-sensor

Help

Here you can find frequently asked questions and answers. You can first take a look here and see if your question can be answered. If not have a look in our Forum and ask a question.

Error compiling or transferring code in Arduino

Errors that occur by compiling or transferring codes are often caused by simple mistakes which are easy to fix. Often a `;` `(Semikolon)` a `} (geschweifte Klammer)` or other small mistakes appear. But the hardest part of troubleshooting is not fixing the error, but finding the error or source of it. To help find the bug, the Arduino IDE gives you an error message. These are, however, without previous experience with programming, often more cryptic than the actual error.

You will therefore find some common errors along with the error message and instructions for correcting the error. If you can not solve your problem with these, look in the forum and create there if necessary a contribution with the description of your error. For this you should provide the best error message and your sketch.

> Of course, if you publicly present your sketch (for example, in our forum), you should always make sure that you do not disclose personal information to the public. Therefore, you should first look in the code, if you have your Wifi name (SSID) and the password built into the sketch. If this is the case, please delete it from the sketch before publishing it on the internet.

▼ I get an error when I connect the fine dust sensor. But I did not change the sketch, what is wrong here?

First, look at the error message. The message can be found in the Arduino IDE below your sketch, in a separate area. Here is your error message. You can scroll through this message with your mouse or drag the area around to see the entire error message.

Even if you have the sketch directly from the openSenseMap without changig it, it can happen that an error with a similar or identical error message like the following occurs:

```
'undefined' was not declared in this scope

#define SDS_UART_PORT (undefined)
                       ^
 /Users/user-name/Documents/Arduino/libraries/sketch_jun27a/example_sketch123.ino:77:12: note: in expansion of macro
'SDS_UART_PORT'
   SDS011 SDS(SDS_UART_PORT);
              ^
 /Users/user-name/Documents/Arduino/libraries/example_sketch123/example_sketch123.ino: In function 'void setup()':
example_sketch123:247: error: 'undefined' was not declared in this scope
     undefined.begin(9600);
     ^
exit status 1
'undefined' was not declared in this scope
```

The error is caused here by

```
#define SDS_UART_PORT (undefined)
```

and

```
undefined.begin(9600);
```

An error in the configuration has crept in here, because where `undefined` is displayed should stand `serial1` or `serial2`. The senseBox MCU gets here from your sketch the information to which port the fine dust sensor was connected. However, the senseBox MCU can not handle the information `undefined` and thus does not know which port the fine dust sensor sends the measured values to.

Please follow the steps to solve the problem:

- Look where the cable from the fine dust sensor is connected at your senseBox MCU.

Look at which of the two color-coded ports you've connected your fine dust sensor.

- If you have connected the fine dust sensor like in the picture to the serial port 1 (red marked port), you can exchange the `undefined` in the sketch with `Serial1`.
- If you have connected your fine dust sensor besides to serial port 2 (yellow marked in the picture), you can exchange the `undefined` in the sketch with `Serial1`
- Please conduct the change in both lines. So, at the point where `#define SDS_UART_PORT (undefined)` is in the text and where `undefined.begin (9600);` stands.
- Then compile the sketch again.

Your problem should be solved - Have fun with your fine dust sensor.

Please visit the Forum[1] if you have any problems with this instruction.

Questions about Programming

▼ I would like to include an extern library. Is it possible?

Yes it is. The senseBox is not limited to the included sensors. You can extend it with any sensors. However externl libraries from the suppliers are often needed.

This help website[1]! explains how to integrate them manually in the Arduino IDE.

Questions about the Data transmission

▼ I have problems with the transmission of data via wifi. What can I do?

It is possible with some of our Wifi Bees of the type WINC1500 that an old firmware (Version 19.4.4) is installed. This can lead to problems with the data transmission. If this is the case, please visit this help website[2] to refresh the firmware.

Questions to the Connection between the senseBox and the Computer

▼ I have a problem with the connection of my sensBox MCU and my windows-Computer. What can I do?

On some of our Windows computer, it may happen that the USB bootloader drivers are not installed correctly. This can lead to the computer not recognizing the senseBox MCU as a USB device and is therefore unable to transfer files. If you encounter these problems, check this help page[3] to see if your drivers work and update them if necessary.

Questions about the senseBox-Project

▼ Where can I buy the new senseBox version 2

That is easy, visit sensebox.kaufen and configurate the senseBox fitting to your needs.

▼ How can I ensure to be at the latest status considering the senseBox?

Just sign up to our newsletter and never miss any news.

---

1. https://forum.sensebox.de ↩

1. See *6.3 Manual Integration of Libraries* ↩

2. See *6.4 Firmware Update Wifi-Bee* ↩

3. See *6.5 Update Windows USB-Bootloader Drivers* ↩

Downloads

Here you can find available downloads that may help you with your senseBox.

Documentation as PDF

This book is available as PDF, too! Download PDF version to create yourself a printed version of this book.

PDF Download: senseBox:edu - Documentation[1]

More useful downloads are coming soon ;)

[1]. https://github.com/sensebox/books-v2/raw/gh-pages/senseBox:edu_en.pdf ↩

Manual Integration of Libraries

To be able to connect own sensors to the senseBox, the manufacturers of many sensors provide suitable libraries. Here we show you how to download libraries from a Github repository and integrate them manually. You can use this guide for any external libraries you want to include in Arduino.

ATTENTION: The libraries required for the senseBox are already in the board support package, which is downloaded in Step 2. You should not longer integrate them manually like it was suggested in the previous books. This creates duplications that can lead to errors.

Download and add Libraries

▼ 'Library' - What is it and why do I need it?

As the name suggests, a library is a collection of something - a collection of methods to be more specific. Methods are programming smaller sections of code that can be applied to an object. For example, with the senseBox, a method can be invoked to turn the LEDs on and off on the MCU. There are a lot of such standard methods that are used by a variety of programs. In order not to have to transfer these methods individually into the program code, they can be stored in libraries. So a library is a file that stores many methods. You can include libraries in your code. For this it is enough if they are stored in the Arduino folder for libraries and then they are integrated with a single line at the beginning of the program code. This is how it looks like in Arduino for the library named "senseBoxIO":

```
#include <senseBoxMCU.h>;
```

If the library is included, all methods contained in it can be used in the code.

The manual installation of the libraries can cause errors very quickly, so be sure to pay close attention to each step. To help you with the installation as well as possible, we have written a separate manual for each operating system. Choose the right system for your computer and follow the steps given.

- Libraries einfügen Windows
- Libraries einfügen Mac
- Libraries einfügen Linux

Insert Libraries Windows

1. Most external libraries can be found in Github repositories. To download them, you have to click the green button `Clone or download` and click `Download ZIP` afterwards.

*Example download of the *

2. If the download does not start on its own, a window opens in which you have to select the field `Save file` and place the folder anywhere on your computer (this is the Downloads folder by default).
3. The downloaded file is a `.zip` archive, which is a compressed version of the library. The next step is to unpack this `.zip` archive. To do this, open the archive location and right-click it and select 'Extract All ...' in the menu that appears. Select the same folder as the download location (for example, the Downloads folder).
4. Now open the Arduino IDE. Go to `File` -> `Preferences` :

Click `File` and then` Preferences`

and look in the box under `Sketchbook location 'for the location of the Sketchbook folder.

Look in the red-marked box to see where your sketchbook location is

Remember the path to this folder, ie the location where it is stored.

> You need to move the library folder you have already downloaded and unpacked to the sketchbook location in the next step. It is therefore very important that you remember exactly the corresponding location from point 4, to avoid errors that could occur later.

5.Now navigate to the sketchbook location in your file explorer (see 4.). Note that the destination folder is named 'Arduino' in the File Explorer at the sketchbook location. Double-click on the folder to see its contents. The folder contains another folder named "libraries".

▶ **What do I do if there is no 'libraries' folder?**

Now copy or drag the downloaded and unzipped folder into the `libraries` -folder.

1. Close the Arduino program completely and start it again to complete the installation of the respective libraries.

> Unfortunately, a typical error is that the senseBox library is not placed in the correct folder. Please check again if you put the file in the right folder from 4..

Insert Libraries Mac

1. Most external libraries can be found in Github repositories. To download them, you have to click the green button `Clone or download` and then in the window `Download ZIP` which opens.

   ![Example download of the "Adafruit_HDC_Library" from github.com] (pictures/libraries/github_download.png)

2. The download should start automatically and the file should be automatically unzipped and placed in your "Downloads" folder. Open the Downloads folder and see if the downloaded folder exists there. If there is only one . Zip file instead of one folder, double-click to unpack it.

3. Now open the Arduino IDE. Go to the top of `Arduino` -> `Settings ...` :

   Click `Arduino` and then` Settings ... `

   and look in the box under ` Sketchbook location 'for the location of the Sketchbook folder.

   Look in the red-marked box to see where your sketchbook location is

   Remember the path to this folder, ie the location where it is stored.

   > You need to move the libraries you have already downloaded to the sketchbook location in the next step. It is therefore very important that you remember exactly the corresponding location from point 3, to avoid errors that could occur later.

4. Now you navigate to the sketchbook location in your Finder. Note that the destination folder in the Finder is named 'Arduino' at the sketchbook location. Double-click on the folder to see its contents. The folder contains another folder named "libraries".

   ▶ **What do I do if there is no 'libraries' folder?**

   Copy or drag now the downloaded (unzipped) folder into the `libraries` -folder.

5. Close the Arduino program completely and start it again to complete the installation of the respective libraries.

> Unfortunately, a typical error is that the senseBox library is not placed in the correct folder. Please check again if you have placed the file in the correct folder from 3.

Insert Libraries Linux

1. Most external libraries can be found in Github repositories. To download them, you have to click the green button `Clone or download` and then in the window `Download ZIP` which opens.

*Example download of the *

2.The download starts automatically and saves a `.zip` archive in your Downloads folder. Open the Downloads folder and unpack the `.zip` file with right-click -> `Extract Here` (*`Extract Here`*).

1. Now open the Arduino IDE. Go to `File` -> `Preferences` :

Click `File` and then` Preferences`

and look in the box under ` Sketchbook location 'for the location of the Sketchbook folder.

Look in the red-marked box to see where your sketchbook location is

Remember the path to this folder, ie the location where it is stored.

> You need to move the libraries you have already downloaded and unzipped to the sketchbook location in the next step. It is therefore very important that you remember exactly the corresponding location from point 3, to avoid errors that could occur later.

2. Now navigate to the sketchbook location in your file explorer (see 3.). Note that the destination folder is named 'Arduino' in the File Explorer at the sketchbook location. Double-click on the folder to see its contents. The folder contains another folder named "libraries".

   ▶ **What do I do if there is no 'libraries' folder?**

   Copy or drag now the downloaded (unzipped) folder into the `libraries` -folder.

3. Close the Arduino program completely and start it again to complete the installation of the respective libraries.

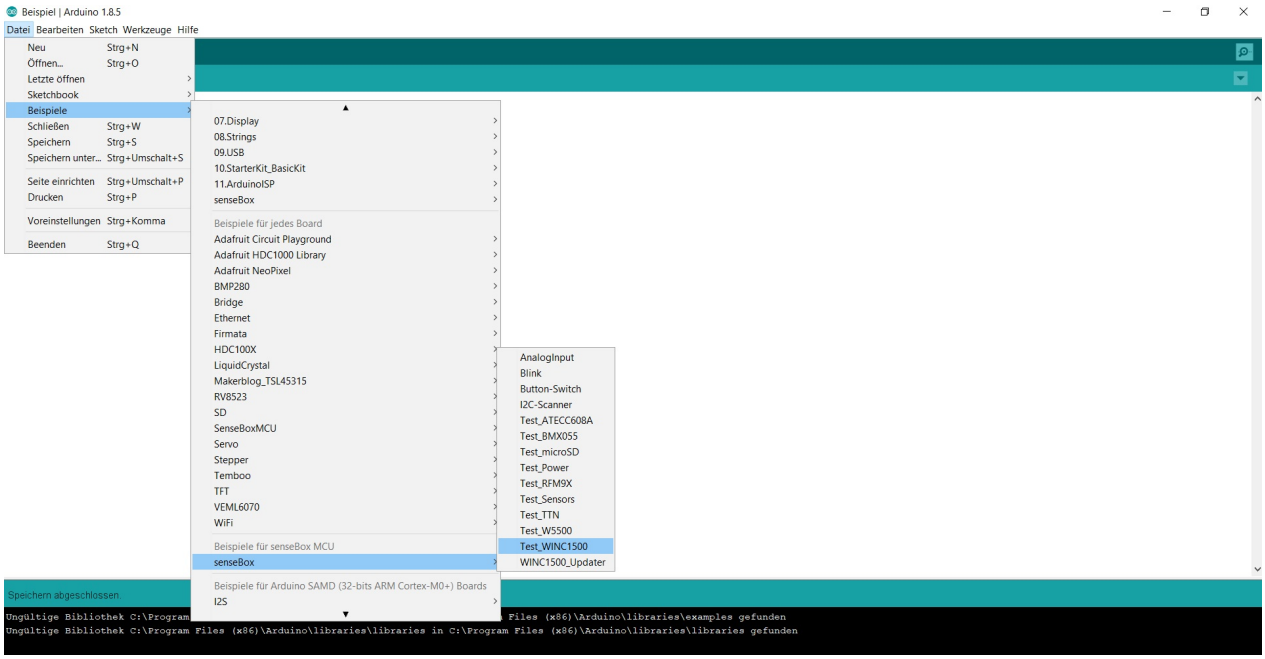
> Unfortunately, a typical error is that the senseBox library is not placed in the correct folder. Please check again if you have placed the file in the correct folder from 3..

Firmware Update Wifi-Bee

Unfortunately, some of our WINC1500 WiFi Bees have outdated firmware (version 19.4.4) installed. Unfortunately, there is no other way to update this firmware than to do it manually. The following chapter explains how to find out which firmware you are using and (if you have an outdated version) how to update it.

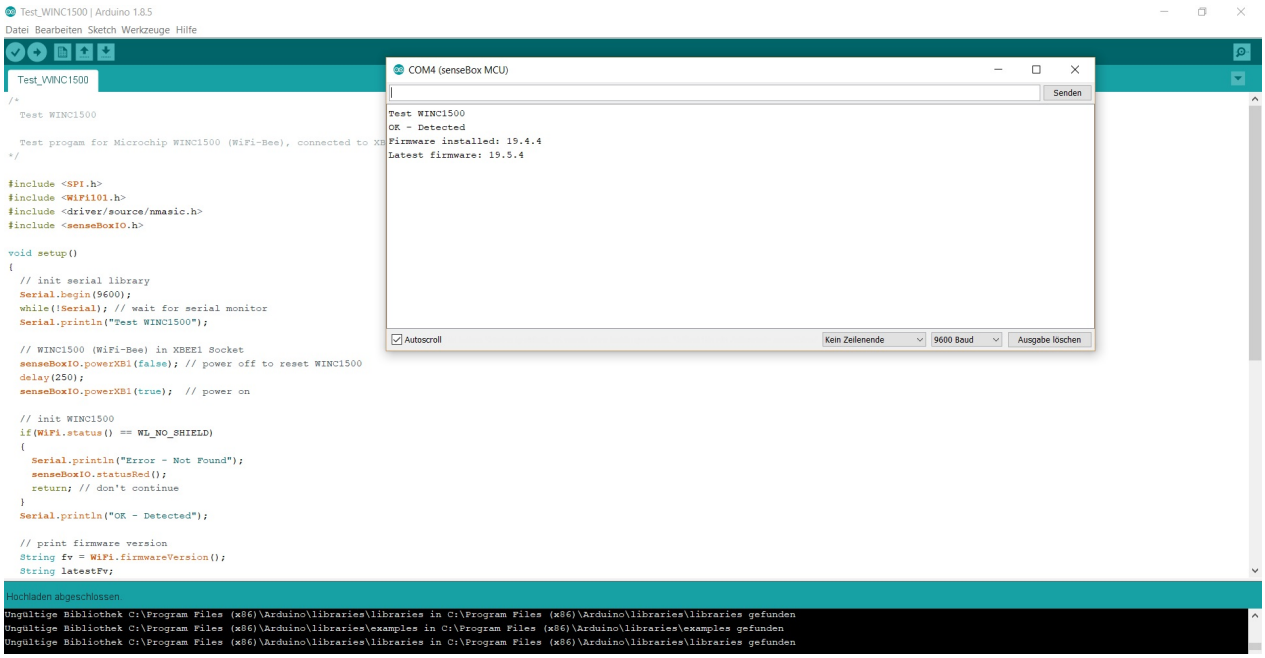
Test the Version

First, it must be found out which version the supplied WiFi Bee has. Go to"file" " Examples " and under "Examples for senseBox MCU" on"Test_WINC1500"



Open the WiFi-Test

Now upload the sketch to your board (by clicking the arrow icon). Note that the WiFi Bee must be plugged into your senseBox board (please plug it in to XBEE1). Then click on the serial monitor (by clicking the magnifying glass icon) and it will be checked if your Wifi-Bee is functional and shows which firmware is installed on it.



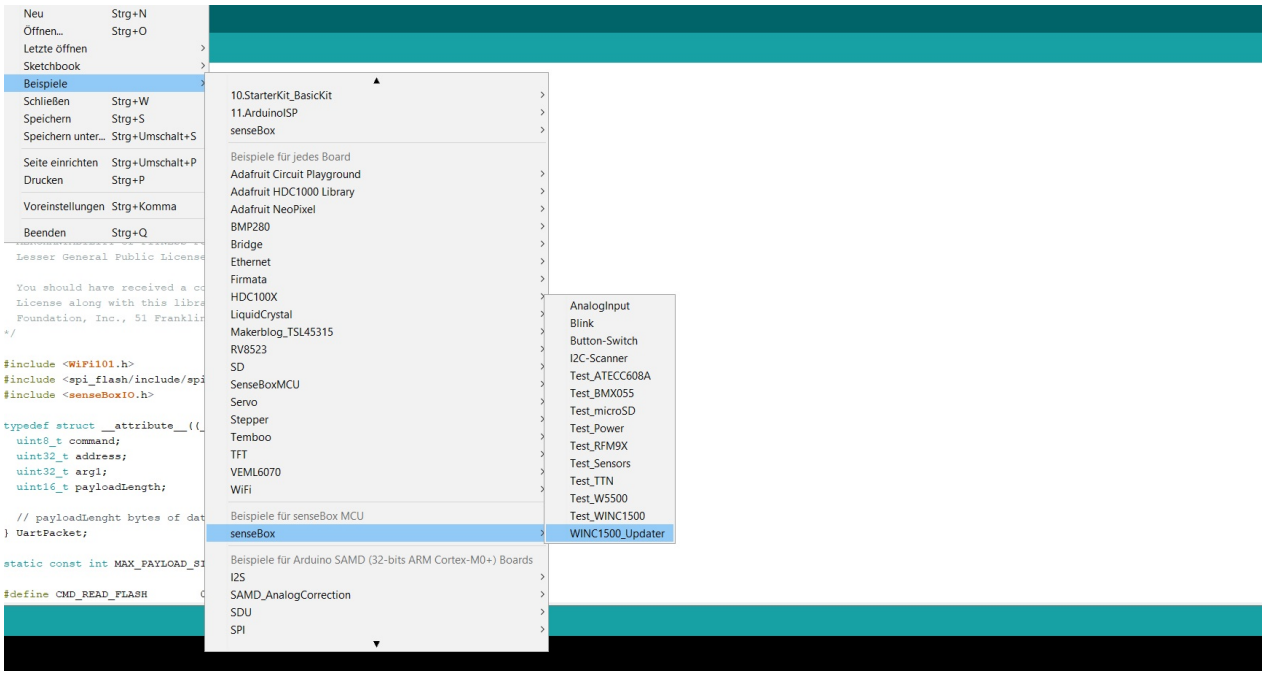
Test results with a non-current firmware

If you have a firmware ** 19.5.2, or higher ** you can cancel here. Your WiFi Bee works perfectly.

If you have a firmware lower than 19.5.2 you unfortunately have to update the firmware. You will find out how this works in the next step.

WiFi-Bee Firmware Update

To update the firmware, follow the path from the top:"file"; " Examples " and under "Examples for senseBox MCU" on"WINC1500_Updater"

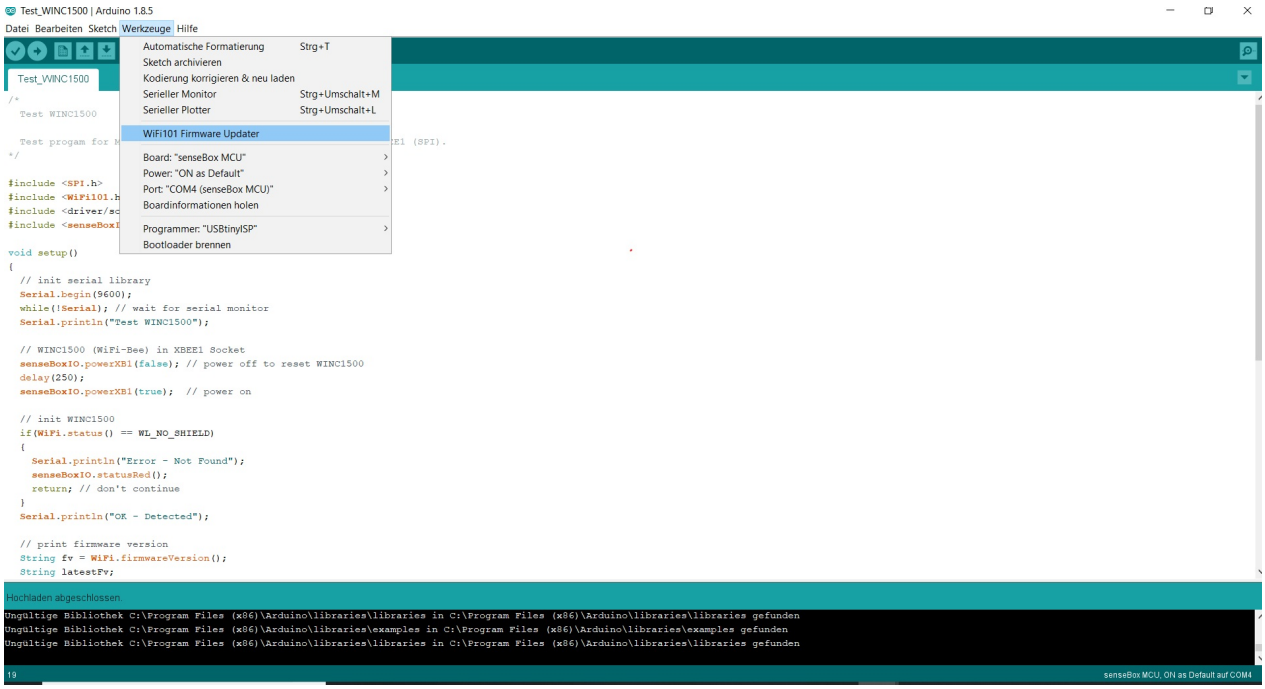


Open the WINC1500_Updater

Now load the sketch on your board (by clicking the arrow icon), on which the WiFi Bee is plugged (please plug it on XBEE1)

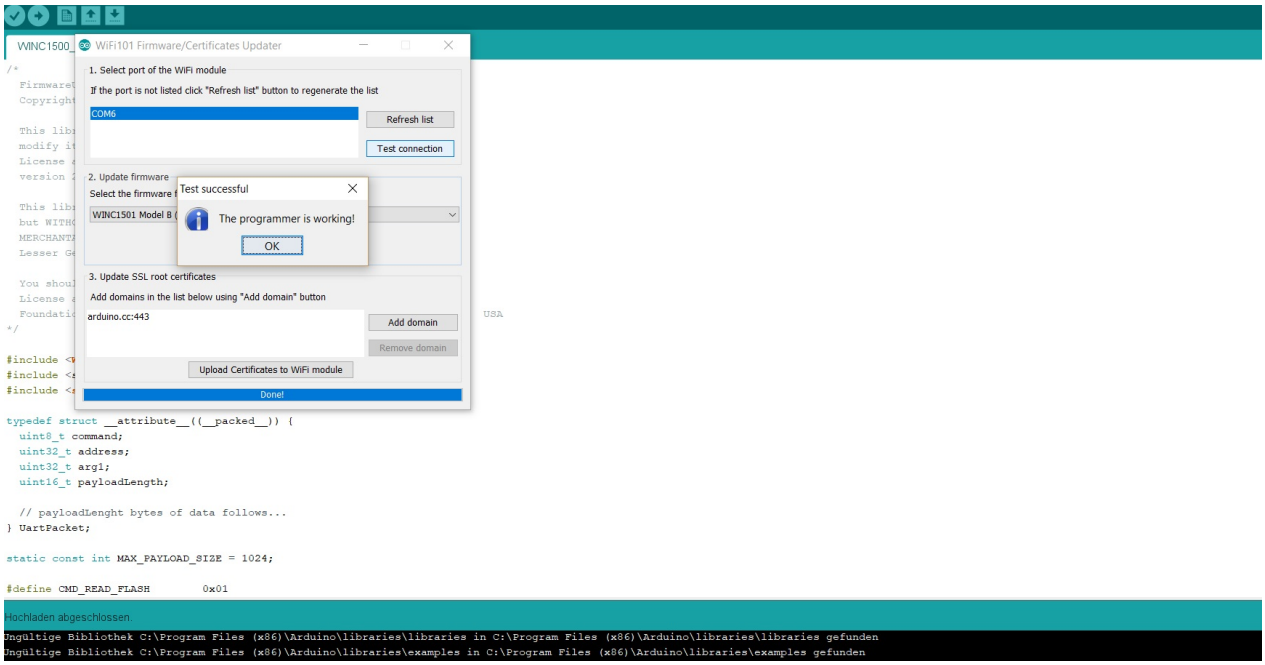
> Do not open the serial monitor this time (do not click on the magnifying glass icon)

Now click `Tools` and select `Wifi101 Firmware Updater` .



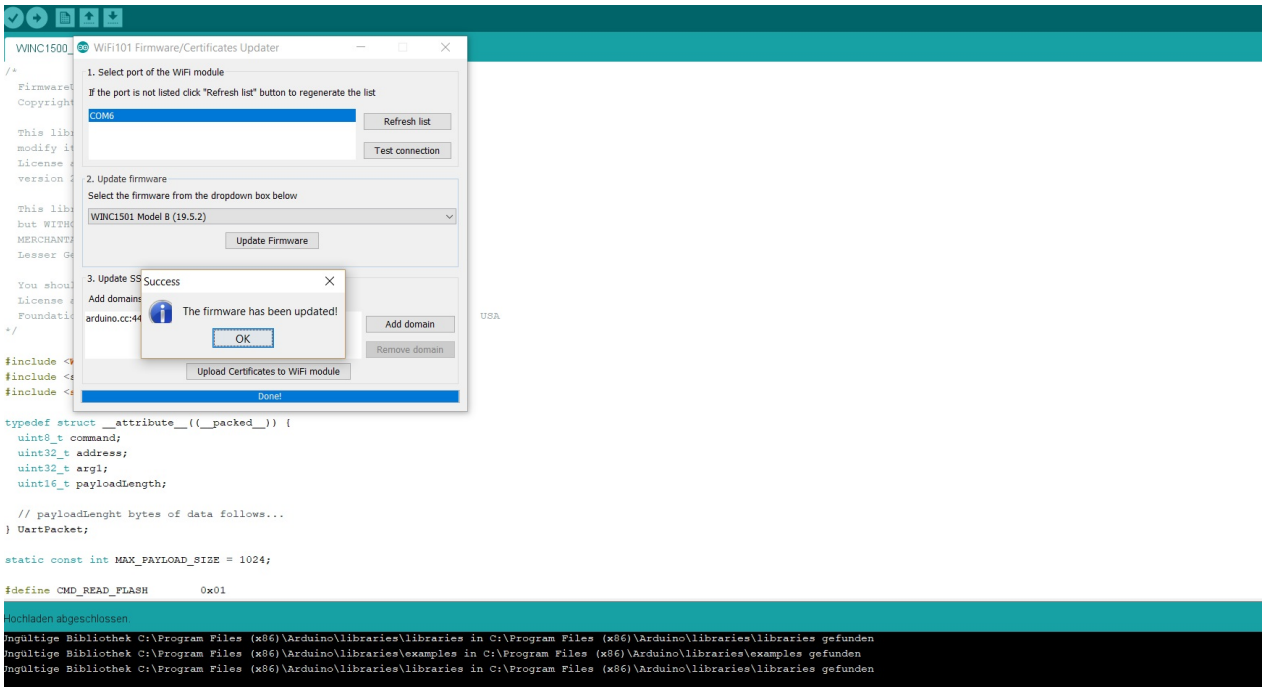
Choose Wifi 101 Firmware Updater

First, you can test the connection by clicking on the `COM Port` and then on `Test connection`. The following information should come back: "The programmer is working!"



Test connection of the Wifi-Bee

Almost done, now just click on `Update Firmware` and the upload starts. Then a success message should come "The firmware has been updated!".



Finally Update Firmware

> Do not let yourself be confused that there is also a version of the firmware 19.5.4, but this is not found in Arduino if you want to update the firmware. This firmware will be integrated with the Arduino Release 1.8.6 and will be available from then on. Nicht davon irritieren lassen, dass es auch eine Version der Firmware 19.5.4 gibt, diese aber nicht in Arduino zu finden ist wenn man die Firmware updaten möchte.

We apologize for the detour and wish you lots of fun with the senseBox.

If this article has not helped you, you can try on www.forum.sensebox.de to look for a solution, or if necessary submit yourself a contribution.

Update Windows USB Bootloader Driver

In some rare cases Windows can cause problems with the USB-Bootloader drivers. What you can do to check if your drivers are installed correctly and how you can install them if you find out here.

Check Driver Status

To check if your USB bootloader drivers work, follow these steps:

- Connect your senseBox MCU board via USB cable to your Windows computer
- Open the Device Manager by searching for `Device Manager` in the Windows bar and open it with a click.
- Activate the bootloader mode of the senseBox MCU by pressing the `Reset` button (red mechanical button on the senseBox MCU) twice in quick succession
- The device manager should now show a point `Connections (COM & LPT)`
- Click on the item to open a list of connected devices, where the senseBox MCU should be listed
- If the senseBox MCU is not listed, disconnect the USB cable from the computer and reconnect it - leave the device manager open and look what happens

If there is no corresponding and no new device displayed on reconnect, your USB bootloader drivers are not installed correctly. Download the current drivers here with one click:

Download senseBox MCU Driver[1]

Now go to Device Manager and select `Update Driver` -> `Find on Computer` and select the drivers you just downloaded.

Restart the computer and check the driver status again, as stated above. The senseBox MCU should now be recognized.

> If this article did not help you, you can try to search for a solution in our forum or create a post there yourself.

[1]. https://github.com/watterott/senseBox-MCU/raw/master/arduino/driver.zip ↩

Update of Board Support Package and Libraries

We have changed the installation steps, to make the installation and updates of the senseBox libraries more user-friendly. On this page, we'll show you the steps you need to take to update your board support package and your senseBox libraries. This guide only applies to you if you have completed the first steps of this book before June 23, 2018 .

What is new?

We have developed a new board support package that combines the old board support package with the senseBox libraries. This bypasses the error-prone manual installation of the senseBox libraries. At the same time, Ardunio IDE's built-in update support for board support packages can be used to bring the libraries up to date. So updates can be recorded in the future with much less effort.

Instructions for updating

The update consists of 2 steps:

1. Deleting the senseBox libraries from the Sketchbook folder to avoid duplication of libraries and the use of old versions.
2. The installation of the new board support package to integrate the libraries in Arduino. Wähle dein Betriebssystem, um die passende Anleitung zu sehen:

- Windows
- Mac(OSX)
- Linux

Step 1: Delete the senseBox libraries from the sketchbook folder

1. Now open the Arduino IDE. Go to `File` -> `Preferences` :

Click `File` and then` Preferences`

and look in the box under `Sketchbook location 'to see where the Sketchbook folder is stored.

Look in the red-marked box to see where your sketchbook location is

Remember the path to this folder, ie the location where it is stored.

1. Now navigate to the sketchbook location in your file explorer (see 1.). Note that the destination folder is named 'Arduino' in the File Explorer at the sketchbook location. Open this folder. Inside the `Arduino` folder is a folder called `libraries`. Within this folder are the senseBox libraries. Delete the `libraries` folder to remove them.

> If you are an experienced Arduino user and have in the past included other external libraries that are not part of the senseBox libraries. Go to the `libraries' folder and delete all the libraries that were not external to you, rather than deleting the entire folder.

2. Now completely close the program Arduino and start it again to complete the deletion of the old senseBox libraries.

Step 2: Integrate new board support package

To incorporate the new board support package, please proceed similar, as in the first steps. There will just be a few small changes.

1. Paste the following URL in your Arduino IDE under File -> Preferences into the field Additional Board Administrator URLs :

```
https://github.com/sensebox/senseBoxMCU-core/raw/master/package_sensebox_index.json
```

> An der Stelle steht im Normalfall vorher schon folgende URL: https://github.com/watterott/senseBox-MCU/raw/master/package_sensebox_index.json diese sieht der obigen sehr ähnlich, ist aber nicht die gleiche URL. Sie muss aber unbedingt durch die oben stehende URL ausgetauscht werden.

Open the preferences and paste the URL

2.Now open the board administrator under Tools -> Board: "..." -> Board Administrator and search there for the senseBox SAMD Boards - Package.

Search for the red highlighted package

If you click on the entry in the list, there appears an update button.

> It is important to click on the entry first. Otherwise, the update button is not displayed, even if there is already a new version.

1. Click on this button and make sure that the installed version is higher than 1.1.0.

Click 'Update' to update the board support package

Since we regularly update the senseBox SAMD boards package for you, you should always go back to the board administrator and see if the senseBox SAMD boards package is still up to date. Open the board administrator as described above, look for senseBox SAMD boards and click there on 'update'.

Delete the senseBox Libraries from the Sketchbook Folder

1. Now open the Arduino IDE. Go to `Arduino` -> `Settings ...` :

Click `Arduino` -> ` Settings ... `

and look in the box under ` Sketchbook location 'to see where the Sketchbook folder is stored.

Look in the red-marked box where your sketchbook location is

Remember the path to this folder, ie the location where it is stored.

1. Now navigate to the sketchbook location in your file explorer (see 1.). Note that the destination folder is named 'Arduino' in the File Explorer at the sketchbook location. Open this folder. Inside the `Arduino` folder is a folder called `libraries`. Within this folder are the senseBox libraries. Delete the `libraries` folder to remove them.

> aria-hidden = "true" style = "color: # f0ad4e"> If you are an experienced Arduino user and have in the past included other external libraries that are not part of the senseBox libraries. Go to the `libraries ` folder and delete all the libraries that were not external to you, rather than the delete entire folder.

2. Now completely close the Arduino program and restart it to finish deleting the old senseBox libraries.

Step 2: Integrate new board support package

To incorporate the new board support package, it's similar, as in the first steps, with a few small changes.

1. Paste the following URL in your Arduino IDE under 'Arduino `->` Settings ... ` in the field Additional Board Administrator URLs :

```
https://github.com/sensebox/senseBoxMCU-core/raw/master/package_sensebox_index.json
```

> An der Stelle steht im Normalfall vorher schon folgende URL: https://github.com/watterott/senseBox-MCU/raw/master/package_sensebox_index.json diese sieht der obigen sehr ähnlich, ist aber nicht die gleiche URL. Sie muss aber unbedingt durch die oben stehende URL ausgetauscht werden.

Open the preferences and paste the URL there

2. Now open the board administrator under Tools -> Board: "..." -> Board Administrator and search there for the senseBox SAMD Boards - Package.

Search for the red highlighted package

If you click on the entry in the list, there appears an update button.

> It is important to click on the entry first. Otherwise, the update button is not displayed, even if there is already a new version.

3.Click on this button and make sure that the installed version is higher than 1.1.0.

Since we regularly update the senseBox SAMD boards package for you, you should always go back to the board administrator and see if the senseBox SAMD boards package is still up to date. Open the board administrator as described above, look for senseBox SAMD boards and click there on 'update'.

Step 1: Delete the senseBox libraries from the sketchbook folder

1. Now open the Arduino IDE. Go to `File` -> `Preferences` :

Click `File` -> ` Preferences`

and look in the box under ` Sketchbook location 'to see where the Sketchbook folder is stored.

Look in the red-marked box to see where your sketchbook location is

Remember the path to this folder, ie the location where it is stored.

1. Now navigate to the sketchbook location in your file explorer (see 1.). Note that the destination folder is named 'Arduino' in the File Explorer at the sketchbook location. Open this folder. Inside the `Arduino` folder is a folder called `libraries`. Within this folder are the senseBox libraries. Delete the `libraries` folder to remove them.

> aria-hidden = "true" style = "color: # f0ad4e"> If you are an experienced Arduino user and have in the past included other external libraries that are not part of the senseBox libraries. Go to the `libraries` folder and delete all the libraries that were not external to you, rather than the delete entire folder

2. Now completely close the program Arduino and start it again to complete the deletion of the old senseBox libraries.

Step 2: Integrate new board support package

To incorporate the new board support package, it's similar, as in the first steps, with a few small changes.

1. Paste the following URL in your Arduino IDE under File -> Preferences into the field Additional Board Administrator URLs :

```
https://github.com/sensebox/senseBoxMCU-core/raw/master/package_sensebox_index.json
```

> Normally, the following URL already exists at this point: https://github.com/watterott/senseBox-MCU/raw/master/package_sensebox_index.json this looks very similar to the above, but is not the same URL. However, it must necessarily be replaced by the above URL.

Open the preferences and paste the URL

2.Now open the board administrator under Tools -> Board: "..." -> Board Administrator and search there for the senseBox SAMD Boards - Package.

Search for the red highlighted package

If you click on the entry in the list, there appears an update button.

> It is important to click on the entry first. Otherwise, the update button is not displayed, even if there is already a new version.

1. Click on this button and make sure that the installed version is higher than 1.1.0.

Click 'Update' to update the board support package

Since we regularly update the senseBox SAMD boards package for you, you should always go back to the board administrator and see if the senseBox SAMD boards package is still up to date. Open the board administrator as described above, look for senseBox SAMD boards and click there on 'update'.