| | |
|---|---|
| Topic: | Weather Station |
| Author: | JOKI |
| Date: | 25.08.2021 |

# 0. Before you start

This document will give you an overview of the source code for the example package Weather Station. Before you can work with it you need to set up your working environment as explained in the document “ExamplePackage_GettingStarted”. Make sure you have read this document beforehand and executed all the steps to configure your Workbench. Especially, that you have added all the paths to the *Source Location* and *Includes* properties.

Note, that there are **two different sets of code for the STM32 microcontroller**; one for the 5.0-inch display board and one for the 4.3-inch display board. The screenshots in this documentation are from the code for the 5.0-inch display board. The 4.3-inch code varies mainly in the size and position parameters for drawing the weather interface.

For a more detailed explanation of how the microcontroller works, please refer to the STM32F429 Reference Manual (RM0090) and the Standard Peripheral Library documentation provided by STMicroelectronics.

# 1. Introduction

In this example package we will use the BME680 environmental sensor to measure pressure, humidity, temperature and air quality. The sensor is not connected directly to the STM32 microcontroller but to the ESP32-WROOM-32 module. The ESP32 is as well a programmable microcontroller, but it also provides WIFI and Bluetooth connectivity. We will program the ESP32 module to retrieve the sensor data from the BME680. The STM32 will ask the ESP32 for this data and get this via UART. Parallel, the ESP32 will act as a Webserver. A WiFi capable device will be able to connect to the board directly and see the sensor data in any web browser.

# 2. Additional Requirements for this Example

## 2.1 Additional required Hardware

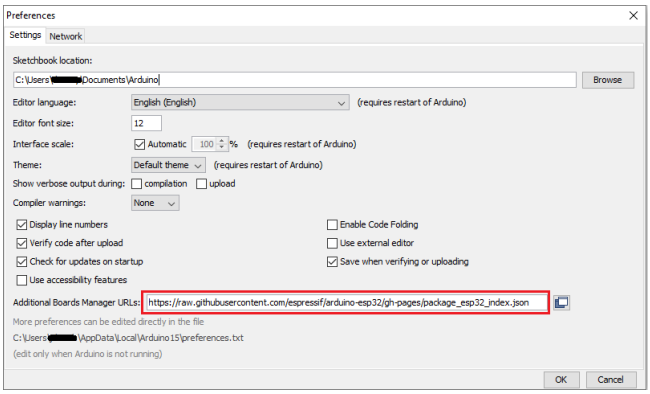
- FT232 USB-to-TTL Serial Converter
- MiniBridge Female 6-pin Connector by ERNI (for an easy connection to the 1.27 mm programming pins of the ESP32 module)

## 2.2 Arduino IDE – ESP32

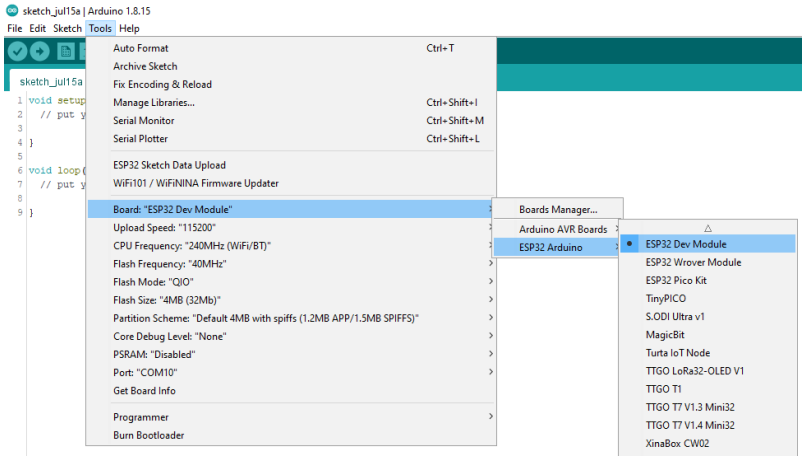
In this example we also program the ESP32 module of our display board. For programming we use the Arduino IDE. Therefore, you need to **download the latest version** of this free software (for this example the version 1.8.15 was used). Some adjustments need to be done, so you can program the ESP32 module with this IDE. Open the Arduino IDE and open the preferences window with *File -> Preferences*. Here, you have to add the URL for the ESP32 board manager.

Write
*https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json*

at “Additional Boards Manager URLs:” and click *OK*.




After restarting the IDE you need to open the *Boards Manager* with *Tools -> Board -> Boards Manager…* and install ***esp32*** by Espressif Systems. Now you should be able to select the ESP32 module as your board. Choose “ESP32 Dev Module”, like in the following picture.




To host the webpage for external devices, we will store a HTML file and an image on the built-in flash of the ESP32 module. For this you need to install a plugin for your Arduino IDE. An instruction for installing this plugin can be found here: [Install ESP32 Filesystem Uploader in Arduino IDE | Random Nerd Tutorials].
Now you can upload external files to the SPIFFS filesystem of the ESP32 module. This is done by clicking *Tools->ESP32 Sketch Data Upload*. The files, that shall be uploaded, have to be in a directory called “data” in the same location as your Arduino sketch file.

## 2.3 Arduino IDE – Webserver Libraries

As we are going to implement an asynchronous webserver with the ESP32 module, so that you can monitor the sensor data remotely, two additional libraries are required which are not accessible via the Arduino Library Manager. They have to be added manually.

We need the AsyncTCP library and the ESPAsyncWebServer library by me-no-dev which can be downloaded for free on GitHub. Follow the subsequent links and download each library code as a ZIP file.

GitHub - me-no-dev/AsyncTCP: Async TCP Library for ESP32

GitHub - me-no-dev/ESPAsyncWebServer: Async Web Server for ESP8266 and ESP32

Extract each ZIP file into */”Arduino-installation-folder”/libraries/*. This directory should now contain the two directories *AsyncTCP-master* and *ESPAsyncWebServer-master*.

## 2.4 Arduino IDE- Bosch BSEC

Bosch provides an API to access its BME680 sensor. The BSEC Software Library. This can be installed inside Arduino IDE. Open the Library Manager and search for “BSEC Software Library” by Bosch Sensortec. Install the latest version. The IDE won’t find the library files at the beginning. At first, you need to modify the *platform.txt* file of the ESP32 board. The file should be located at:
C:\Users*username*\AppData\Local\Arduino15\packages\esp32\hardware\esp32*version_number*\

You need to edit the line which specifies the parameter *recipe.c.combine.pattern*. This should be line 88. Replace this line with:

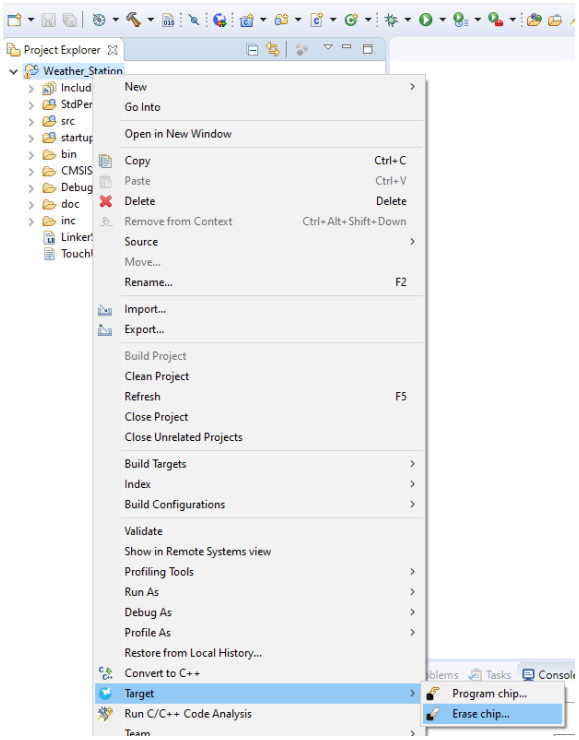
```
recipe.c.combine.pattern="{compiler.path}{compiler.c.elf.cmd}"
{compiler.c.elf.flags} {compiler.c.elf.extra_flags} -Wl,--start-group
{object_files} "{archive_file_path}" {compiler.c.elf.libs} {build.extra_libs}
{compiler.libraries.ldflags} -Wl,--end-group -Wl,-EL -o
"{build.path}/{build.project_name}.elf"
```

After a restart of your Arduino IDE, everything should be ready to work.

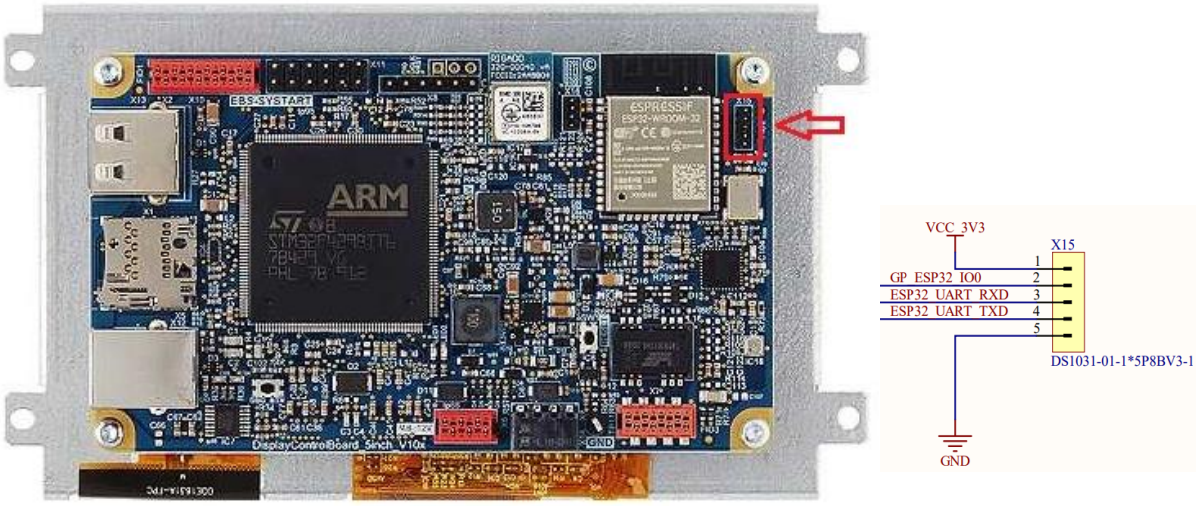
# 3. Upload Routine

For uploading the whole project, a certain procedure has to be followed. At first, you need to erase the STM32 chip. You can do this within your Workbench. Open the Workbench and connect to the display board with the ST-LINK/V2 debugger and with its power supply. Right-click on the project file and press *Target->Erase chip…* Afterwards you can disconnect the ST-LINK/V2.

Alternatively, you can erase the STM32 chip with the tool ST-LINK-Utility.




Now you can flash the ESP32 module. For this you need to connect the USB-to-Serial converter with the 1.27mm pin connector to the right of the ESP32. The location and the pin assignments can be found on the pictures below. Pin 1 is marked with a little dot. Therefor the bottom one is Pin 1. Connect the RX pin of the connector to the TX pin of the USB-to-Serial converter, the TX pin of the connector to the RX pin of the USB-to-Serial converter and both ground pins. The GP_ESP32_IO0 pin is important to the flashing process since this pin defines the boot mode of the ESP32 module. For flashing, this pin must be connected to ground **before** you turn on the power. This will signal the ESP32 to prepare for an upload. Then you can upload the external data to the SPIFFS and the sketch to the microcontroller in the Arduino-typical way. Between uploading the external data and flashing the sketch you need to turn the power off and on again.




After uploading everything to the ESP32 turn off the power, disconnect the GP_ESP32_IO0 pin from ground and turn the power on again.

Now, you can program and flash the STM32 microcontroller as usual.

# 4. Explanation of Example Code

## 4.1 STM32 Code

### 4.1.1 Main function

main.c

```
static weather_data_struct weather_data;

/*************************************************************
 *
 * Function: int main(void)
 *
 * Summary: Entry point of display control application
 *
 * Parameters:
 *
 * Return:
 *
 * Revision History:
 * Creation Date: 21062021
 *
 *************************************************************/
int main(void) {

    /* Initialize the peripherals */
    HW_Init();
    //input_Init();
    display_init();

    comm_err_t err = check_communication_esp32();
    if (err != FCN_OK){
        // communication with esp32 module fails
        while(1){
            status_led_blink();
        }
    }

    get_ip_address_esp32(&weather_data);
    draw_weather_interface(weather_data.ip_address);
    for(;;){
        send_cmd_esp32(GET_ALL);
        get_data_esp32(&weather_data);
        draw_weather_data(weather_data);
        delay_ms(1000);
    }
    return 0;
}
```

We will start our walk through the code at the main function since this gives a perfect overview of the steps that we will take. At the beginning, the used peripherals are initialized. *HW_Init()* initializes and turns on the display backlight. Also, it sets up the different system clocks, the timer, which counts every microsecond, and the GPIO pins for two of the user LEDs.

Afterwards we initialize the display which also includes the initialization of the SDRAM, because this is where the buffer for the displayed images will be located. The DMA2D is used to send the image data directly from the buffer to the LCD, which speeds up the image manipulation. Part of the display initialization is the drawing of the EBS-SYSTART logo and the logo of our 2-in-1 display line.

After the initialization process, we first check if the communication between the STM32 and the ESP32 works correctly. In case it doesn't, an infinite while-loop is entered and the two user LEDs on the back of the board start blinking.

In case the communication works fine, at first the display background is drawn. The background also shows the IP address, where you can find the webpage for remote access. The address is set in the code for the ESP32 module and therefore has to be retrieved beforehand.

Then we enter the main loop. Here we get the sensor data, that was collected by the ESP32, and show it on the display.

### 4.1.2 draw_weather_interface

```
/*************************************************************
 *
 * Function: void draw_weather_interface(char* ip_addr)
 *
 * Summary: Draws the background for the weather station
 *
 * Parameters:
 *
 * Return:
 *
 * Revision History:
 * Creation Date: 28062021
 *
 *************************************************************/
void draw_weather_interface(char* ip_addr){
    DMA2D_Fill_Color(0xFFFF, Layer_1, Buffer_1);
    DMA2D_Draw_Image((HDP-Display21_Logo_Y.width)/2, (VDP-Display21_Logo_Y.height)/2, Display21_Logo_Y.width, Display21_Logo_Y.height,
           0x5F, REPLACE_ALPHA_VALUE, (uint32_t)Display21_Logo_Y.data, CM_RGB565, Layer_1, Buffer_1);
    //temperature box
    DMA2D_Draw_Rectangle(0, 0, HDP*2/5, VDP/2, 0xCF07, 6, Layer_1, Buffer_1);
    DMA2D_write_string("Temperature:", 10, 15, 0x95D6, &courier_new_23, Layer_1, Buffer_1, Layer_1, Buffer_1);
    //humidity box
    DMA2D_Draw_Rectangle(0, VDP/2-3, HDP*2/5, VDP/2 -27, 0xCF07, 6, Layer_1, Buffer_1);
    DMA2D_write_string("Humidity:", 10, VDP/2 +15, 0x95D6, &courier_new_23, Layer_1, Buffer_1, Layer_1, Buffer_1);
    //pressure box
    DMA2D_Draw_Rectangle(HDP*2/5-3, 0, HDP*3/5+3, VDP*2/3, 0xCF07, 6, Layer_1, Buffer_1);
    DMA2D_write_string("Pressure:", HDP*2/5 +10, 15, 0x95D6, &courier_new_23, Layer_1, Buffer_1, Layer_1, Buffer_1);
    DMA2D_write_string("approx. Altitude:", HDP*2/5 +30, VDP/2, 0x8000, &courier_new_15_2, Layer_1, Buffer_1, Layer_1, Buffer_1);
    //gas resistance box
    DMA2D_Draw_Rectangle(HDP*2/5-3, VDP*2/3-3, HDP*3/5+3, VDP*1/3 -27, 0xCF07, 6, Layer_1, Buffer_1);
    DMA2D_write_string("Gas Resistance:", HDP*2/5 +10, VDP*2/3 +15, 0x95D6, &courier_new_23, Layer_1, Buffer_1, Layer_1, Buffer_1);

    //IP-Address
    DMA2D_write_string("IP:", 10, VDP-28, 0xBDEF, &courier_new_15_2, Layer_1, Buffer_1, Layer_1, Buffer_1);
    DMA2D_write_string(ip_addr, 50, VDP-28, 0xBDEF, &courier_new_15_2, Layer_1, Buffer_1, Layer_1, Buffer_1);
}
```

This function uses the basic drawing functions, introduced in the example package *Drawing Text and Images*, to create the background for the display. In this example we use the two-layer mode for the display. The background is drawn on Layer 1, since this one lays behind Layer 2. The data is drawn on Layer 2, as you will see later.

### 4.1.3 send_cmd_esp32

This function sends a one-byte command to the ESP32 module. Since the STM32 and the ESP32 are connected by their UART pins, we can simply use the basic UART functions to communicate.

The commands are specified in an enumeration and must match the definition in the ESP32 code.

```
/*************************************************************
 *
 * Function: comm_err_t send_cmd_esp32(uint8_t cmd)
 *
 * Summary: Sends a one-byte command to the esp32 module
 *
 * Parameters:
 *     cmd - one-byte command
 *
 * Return:
 *  0 - if success
 *
 * Revision History:
 * Creation Date: 28062021
 *
 *************************************************************/
comm_err_t send_cmd_esp32(uint8_t cmd) {

    tTimer send_timeout;

    send_timeout = timer_get_time_ms();
    uart_SendByte(COM3, cmd);
    if(timer_get_time_ms() - send_timeout > 10) return FCN_TIMEOUT;

    return FCN_OK;
}
```

### 4.1.4 get_data_esp32

After the command is sent to the ESP32, we immediately get the response, which is stored in a buffer until it is read. This reading of the response happens in the function *get_data_esp32()*. Since the individual values are floats, we read the four bytes, combine them back to a float by using the *union* data type and store them in a struct for the whole sensor data. This is done for the parameters temperature, humidity, pressure, approximate altitude and gas resistance.

```
/*****************************************************************
 *
 * Function: comm_err_t get_data_esp32(weather_data_struct* w_data)
 *
 * Summary: Get the temperature, humidity, pressure,
 * approximate altitude and gas resistance
 * from bme680 sensor via esp32
 *
 * Parameters:
 *
 * Return:
 *  0 - if success
 *
 * Revision History:
 * Creation Date: 28062021
 *
 *****************************************************************/
comm_err_t get_data_esp32(weather_data_struct* w_data) {
    uint8_t data[4];
    //temperature
    while(uart_Receive4Byte(COM3, &data[0], BIG_ENDIAN_T) == 0);
    float_bytes.bytes[0] = data[0];
    float_bytes.bytes[1] = data[1];
    float_bytes.bytes[2] = data[2];
    float_bytes.bytes[3] = data[3];
    w_data->temperature = float_bytes.floatVal;
    //humidity
    while(uart_Receive4Byte(COM3, &data[0], BIG_ENDIAN_T) == 0);
    float_bytes.bytes[0] = data[0];
    float_bytes.bytes[1] = data[1];
    float_bytes.bytes[2] = data[2];
    float_bytes.bytes[3] = data[3];
    w_data->humidity = float_bytes.floatVal;
    //pressure
    while(uart_Receive4Byte(COM3, &data[0], BIG_ENDIAN_T) == 0);
    float_bytes.bytes[0] = data[0];
    float_bytes.bytes[1] = data[1];
    float_bytes.bytes[2] = data[2];
    float_bytes.bytes[3] = data[3];
    w_data->pressure = float_bytes.floatVal;
    //approx altitude
    while(uart_Receive4Byte(COM3, &data[0], BIG_ENDIAN_T) == 0);
    float_bytes.bytes[0] = data[0];
    float_bytes.bytes[1] = data[1];
    float_bytes.bytes[2] = data[2];
    float_bytes.bytes[3] = data[3];
    w_data->approx_altitude = float_bytes.floatVal;
    //gas resistance
    while(uart_Receive4Byte(COM3, &data[0], BIG_ENDIAN_T) == 0);
    float_bytes.bytes[0] = data[0];
    float_bytes.bytes[1] = data[1];
    float_bytes.bytes[2] = data[2];
    float_bytes.bytes[3] = data[3];
    w_data->gas_resistance = float_bytes.floatVal;

    return FCN_OK;
}
```

### 4.1.5 draw_weather_data

```
/*************************************************************
 *
 * Function: draw_weather_data(weather_data_struct w_data)
 *
 * Summary: Draws the values retrieved from the ESP32 on Layer 2 of the display
 *
 * Parameters: -
 *
 *
 * Return: -
 *
 * Revision History:
 * Creation Date: 28062021
 *
 *************************************************************/
void draw_weather_data(weather_data_struct w_data){
    char temp[11];
    DMA2D_Fill_Color(0x7FFF, Layer_2, Buffer_1);
    //temperature
    sprintf(temp, "%4.2f %cC", w_data.temperature, 0x7f);
    //DMA2D_Draw_FilledRectangle(20, VDP*1/8, HDP*2/5-50, VDP*2/8, 0xFFFFFFFF, Layer_2, Buffer_1);
    DMA2D_write_string(temp, HDP/5 -25*4, VDP/5, 0xFF000000, &arial_56_deg, Layer_2, Buffer_1, Layer_2, Buffer_1);

    //humidity
    sprintf(temp, "%2.0f %%", w_data.humidity);
    //DMA2D_Draw_FilledRectangle(20, VDP*5/8, HDP*2/5-50, VDP*2/8, 0xFFFFFFFF, Layer_2, Buffer_1);
    DMA2D_write_string(temp, HDP/5 -arial_56.chars->image->width*2, VDP*2/3, 0xFF000000, &arial_56, Layer_2, Buffer_1, Layer_2, Buffer_1);

    //pressure
    sprintf(temp, "%6.2f hPa", w_data.pressure);
    //DMA2D_Draw_FilledRectangle(HDP*2/5+20, VDP/5, HDP*3/5-50, VDP*1/6, 0xFFFFFFFF, Layer_2, Buffer_1);
    DMA2D_write_string(temp, HDP*7/10 -arial_56.chars->image->width*6, VDP/5, 0xFF000000, &arial_56, Layer_2, Buffer_1, Layer_2, Buffer_1);

    //approx altitude
    sprintf(temp, "%3.0f m", w_data.approx_altitude);
    //DMA2D_Draw_FilledRectangle(HDP*2/5+20+13*20, VDP/2-10, HDP/4-10, VDP/6-5, 0xFFFFFFFF, Layer_2, Buffer_1);
    DMA2D_write_string(temp, HDP*2/5 +30+13*20, VDP/2, 0xFF000000, &courier_new_23, Layer_2, Buffer_1, Layer_2, Buffer_1);

    //gas resistance
    sprintf(temp, "%5.2f kOhm", w_data.gas_resistance);
    //DMA2D_Draw_FilledRectangle(HDP*2/5+20, VDP*9/12, HDP*2/5, VDP*1/6, 0xFFFFFFFF, Layer_2, Buffer_1);
    DMA2D_write_string(temp, HDP*7/10 -courier_new_23.chars->image->width*6, VDP*3/4+30, 0xFF000000, &courier_new_23,
            Layer_2, Buffer_1, Layer_2, Buffer_1);
}
```

This function now draws the retrieved values from the sensor as a string. Note, that this time we draw on Layer 2 which lies on top of Layer 1. Layer 2 is at first completely transparent as you can see in line 66. The color format used in the two-layer mode is ARGB1555, therefore the color 0x7FFF is white but transparent. This is necessary, because otherwise we would cover the background. Only the strings for the sensor value are opaque.

## 4.2 ESP32 code

```
float temperature;  // deg Celsius
float humidity;     //%
float pressure;     //hPa
float approx_altitude;  //meters
float gas_resistance;   //kOhm
float breath_voc; //ppm
float iaq_value;
float co2_equivalent; //ppm

// Replace with your own network credentials
const char* apSSID = "Weather Station";
const char* apPASSWORD = "";  // only needed for key protected network
IPAddress apIP(192, 168, 0, 1);

AsyncWebServer server(80);
```

At the beginning of the Arduino code for the ESP32, we declare some variables that are needed. At first, we have the floats for the individual sensor values. Afterwards, we define the parameters for our webserver. The name of the WiFi-network generated by the ESP32 module is here called “Weather Station”. The password string is kept empty, because we don’t establish a key protected network. But both strings can be changed as you like. The IP address for the webpage is stored in an *IPAddress* data type. We chose 192.168.0.1, but again, you can change this if you want to. At last, we declare an asynchronous Webserver object which listens on port 80.

```
void setup() {
  Serial.begin(115200);
  hspi.begin();
  iaqSensor.begin(15, hspi);

  bsec_virtual_sensor_t sensorList[10] = {
    BSEC_OUTPUT_RAW_TEMPERATURE,  // iaqSensor.rawTemperature
    BSEC_OUTPUT_RAW_PRESSURE, // iaqSensor.pressure
    BSEC_OUTPUT_RAW_HUMIDITY, // iaqSensor.rawHumidity
    BSEC_OUTPUT_RAW_GAS,  // iaqSensor.gasResistance
    BSEC_OUTPUT_IAQ,  // iaqSensor.iaq
    BSEC_OUTPUT_STATIC_IAQ, // iaqSensor.staticIaq
    BSEC_OUTPUT_CO2_EQUIVALENT, // iaqSensor.co2Equivalent
    BSEC_OUTPUT_BREATH_VOC_EQUIVALENT,  // iaqSensor.breathVocEquivalent
    BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE,  // iaqSensor.temperature
    BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_HUMIDITY, // iaqSensor.humidity
  };
  iaqSensor.updateSubscription(sensorList, 10, BSEC_SAMPLE_RATE_LP);

  SPIFFS.begin();
  WiFi.mode(WIFI_AP);
  WiFi.softAPConfig(apIP, apIP, IPAddress(255, 255, 255, 0));
  WiFi.softAP(apSSID); //open network; to set up pre-shared key protected network: WiFi.softAP(appSSID, apPASSWORD)
```

Let's take a look at the setup function. It starts with setting up the Serial connection which is Arduino's name for the UART connection to the STM32. Then we initialize the SPI pins, since the BME680 sensor is using this interface to communicate, and the sensor object *iaqSensor*. The SPIFFS filesystem also needs to be started, because the HTML page and an image for the page are stored there and need to be accessed.
Next, we configure our access point with the IP address, we specified before and then open the network with the name stored in *apSSID*. You could also add a password.

```
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/webpage.html", "text/html");
  });
  server.on("/ebssystart_logo", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/ebssystart_logo.jpg", "image/jpg");
  });
  server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", String(temperature));
  });
  server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", String(humidity));
  });
  server.on("/pressure", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", String(pressure));
  });
  server.on("/altitude", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", String(approx_altitude));
  });
  server.on("/gasResistance", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", String(gas_resistance));
  });

  server.on("/co2equivalent", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", String(co2_equivalent));
  });
  server.on("/iaq", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", String(iaq_value));
  });
  server.on("/bVOC", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", String(breath_voc));
  });
  //Start server
  server.begin();
}
```

The second part of the setup function configures the webserver. At first, we define that when you type in the predefined IP address in a web browser, which means a "GET /"-request, the HTML file in the SPIFFS is send to the user. In the HTML file, a java script will send every second multiple GET requests, asking for a new temperature, humidity, … value. These requests will be answered by sending the respective data. After configuration, we start the server.

Now, let's take a look at the loop function.

```
void loop() {
  if (iaqSensor.run()) { // If new data is available
    return;
  }
  temperature = iaqSensor.temperature;  // deg Celsius
  humidity = iaqSensor.humidity;  //percent
  pressure = iaqSensor.pressure/100.0;  //hPa
  approx_altitude = calcAltitude(pressure, 1013.25);  // meters
  gas_resistance = iaqSensor.gasResistance/1000.0;  //kOhm
  co2_equivalent = iaqSensor.co2Equivalent;  //ppm
  iaq_value = iaqSensor.iaq;
  breath_voc = iaqSensor.breathVocEquivalent;  //ppm
```

At the beginning, the BME680 sensor is asked if there is new data. If not, we start again at the top. If there is new data, we store the values in the respective variables.

The second part of the loop function manages the UART communication with the STM32. If the ESP32 received an input, it responds depending on the value of the byte.

```
  if(Serial.available()>0){
    input = Serial.read();
    switch(input){
      case 0x00:
        Serial.write(0xAA);
        break;
      case 0x01:
        float_bytes.floatVal = temperature;
        Serial.write(float_bytes.bytes, 4);
        float_bytes.floatVal = humidity;
        Serial.write(float_bytes.bytes, 4);
        float_bytes.floatVal = pressure;
        Serial.write(float_bytes.bytes, 4);
        float_bytes.floatVal = approx_altitude;
        Serial.write(float_bytes.bytes, 4);
        float_bytes.floatVal = gas_resistance;
        Serial.write(float_bytes.bytes, 4);
        break;
      case 0x02:
        float_bytes.floatVal = temperature;
        Serial.write(float_bytes.bytes, 4);
        break;
      case 0x03:
        float_bytes.floatVal = humidity;
        Serial.write(float_bytes.bytes, 4);
        break;
      case 0x04:
        float_bytes.floatVal = pressure;
        Serial.write(float_bytes.bytes, 4);
        break;
      case 0x05:
        float_bytes.floatVal = approx_altitude;
        Serial.write(float_bytes.bytes, 4);
        break;
      case 0x06:
        float_bytes.floatVal = gas_resistance;
        Serial.write(float_bytes.bytes, 4);
        break;
      case 0x07:
        Serial.println(WiFi.softAPIP());
        break;
    }
  }
}
```

## 5. Ideas for Exercise Project

Here, we want to give you some suggestions how you could modify our example code and make your own little project.

You could add little icons to the weather interface, which change depending of the values e.g. of the temperature. So that you could graphically indicate, if it is warm, cold, wet, …