
TUM-GIS Sensor Nodes

Release v0.0.1

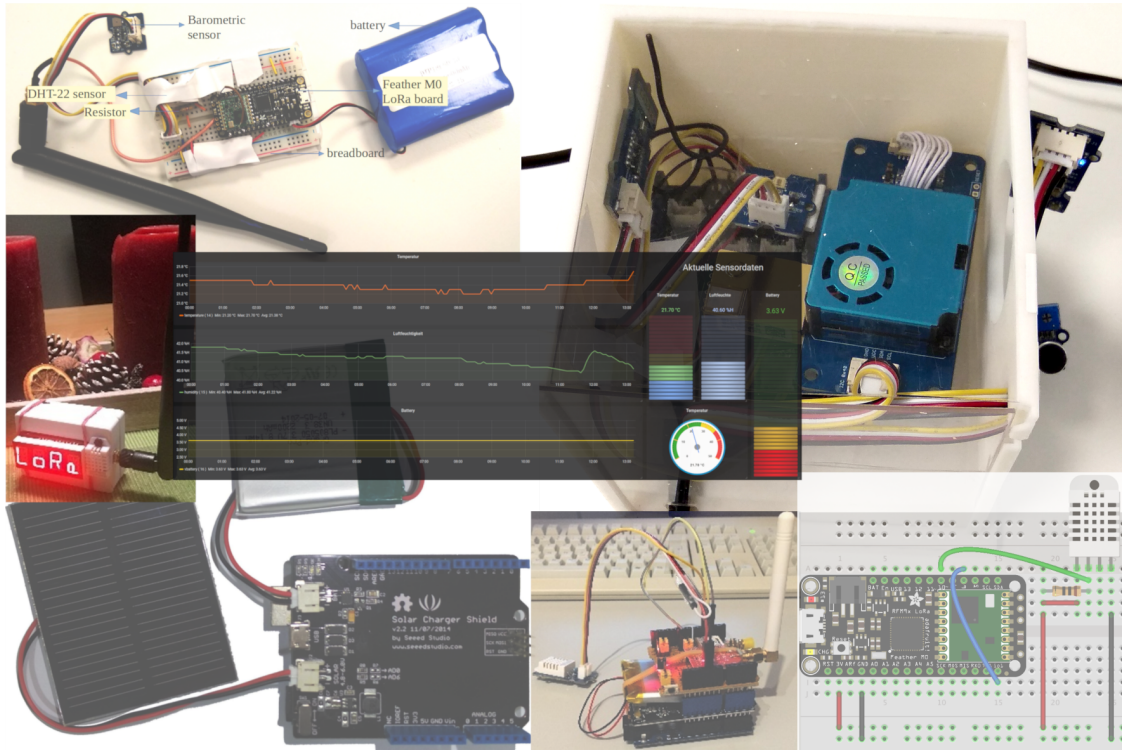
Jan 08, 2020

1	Contact and contribution	3
2	Contents	5
2.1	Solar powered Seeeduino	5
2.1.1	Hardware	5
2.1.2	Wiring setup	5
2.1.3	Software	7
2.1.4	Services	9
2.1.5	Code files	9
2.1.6	References	17
2.2	Indoor Ambient Monitoring	17
2.2.1	Hardware	18
2.2.2	Wiring setup	18
2.2.3	Software	19
2.2.4	Services	21
2.2.5	Code files	21
2.2.6	References	34
2.3	Feather M0 LoRa in TFA Housing	34
2.3.1	Hardware	34
2.3.2	Wiring setup	36
2.3.3	Software	38
2.3.4	Services	40
2.3.5	Code files	40
2.3.6	References	53
2.4	Adafruit 32u4 LoRa	53
2.4.1	Hardware	53
2.4.2	Software	54
2.4.3	Services	56
2.4.4	Code files	57
2.4.5	References	68
2.5	Adafruit 32u4 LoRa with Display	68
2.5.1	Hardware	68
2.5.2	Software	70
2.5.3	Services	72
2.5.4	Code files	74
2.5.5	References	87

2.6	Adafruit M0 LoRa	88
2.6.1	Hardware	88
2.6.2	Software	90
2.6.3	Services	91
2.6.4	Code files	92
2.6.5	References	100
2.7	Dragino LoRa Arduino Shield	100
2.7.1	Hardware	100
2.7.2	Software	104
2.7.3	Services	104
2.7.4	Code files	105
2.7.5	References	119
2.8	Pycom LoPy4	119
2.8.1	Hardware	119
2.8.2	Software	121
2.8.3	Services	122
2.8.4	Code files	123
2.8.5	References	127
2.9	Seeeduino LoRaWAN	128
2.9.1	Hardware	128
2.9.2	Software	130
2.9.3	Services	130
2.9.4	Code files	132
2.9.5	References	139
2.10	Seeeduino LoRaWAN with GPS	139
2.10.1	Hardware	139
2.10.2	Software	141
2.10.3	Services	142
2.10.4	Code files	144
2.10.5	References	150
2.11	Sodaq ONE	151
2.11.1	Hardware	151
2.11.2	Software	153
2.11.3	Services	153
2.11.4	Code files	153
2.11.5	References	153
2.12	Wemos TTGO T-Beam	153
2.12.1	Hardware	153
2.12.2	Software	155
2.12.3	Services	155
2.12.4	Code files	155
2.12.5	References	156

3 Indices and tables 157

This repo contains documentation, Arduino sketches, and images of our sensor nodes and the sensor services we used.



CHAPTER 1

Contact and contribution

We are happy for any kind of comments, questions, corrections, and own contributions. Please visit the [Github Repo](#) of this documentation to report a [correction](#), [bug](#), or [question](#) or contribute with a [pull request](#).

2.1 Solar powered Seeeduino

This sensor node is made to showcase a use-case of LoRaWAN sensor node powered using a solar panel. For achieving this a Seeeduino LoRaWAN microcontroller was used along with a solar panel connected using a solar shield. To show a generic use-case we have used a temperature and humidity sensor in this case, but it can be easily replaced with some other sensor as well. The entire setup was carefully placed in the [ABS Waterproof case](#) which is an easy to install water-proof and dust-proof case for an indoor or outdoor sensor installations. However, this case has no provision for the ventilation unlike the [TFA case](#) and so the readings obtained by the sensor may not accurately represent the outdoor weather conditions. In this example, we measure parameters such as temperature, humidity, and battery voltage.

2.1.1 Hardware

To build this sensor node we have used following hardware components:

- [Seeeduino LoRaWAN board V4.2](#)
- [Grove - DHT-22 Temperature & Humidity Sensor](#)
- [Solar charger shield](#)
- [1.5 W Solar panel](#)
- [0 ohm resistor](#)
- [ABS Waterproof case](#)
- [2000 mAH Battery](#)

2.1.2 Wiring setup

First of all, the solar panel is connected with the SOLAR pin and a battery is connected with a BAT pin on the solar charger shield as shown in the figure below. A DHT-22 Sensor is connected to A2 pin on the Seeeduino board using a connector cable and then the solar charger shield prepared in the previous step is mounted on the board.

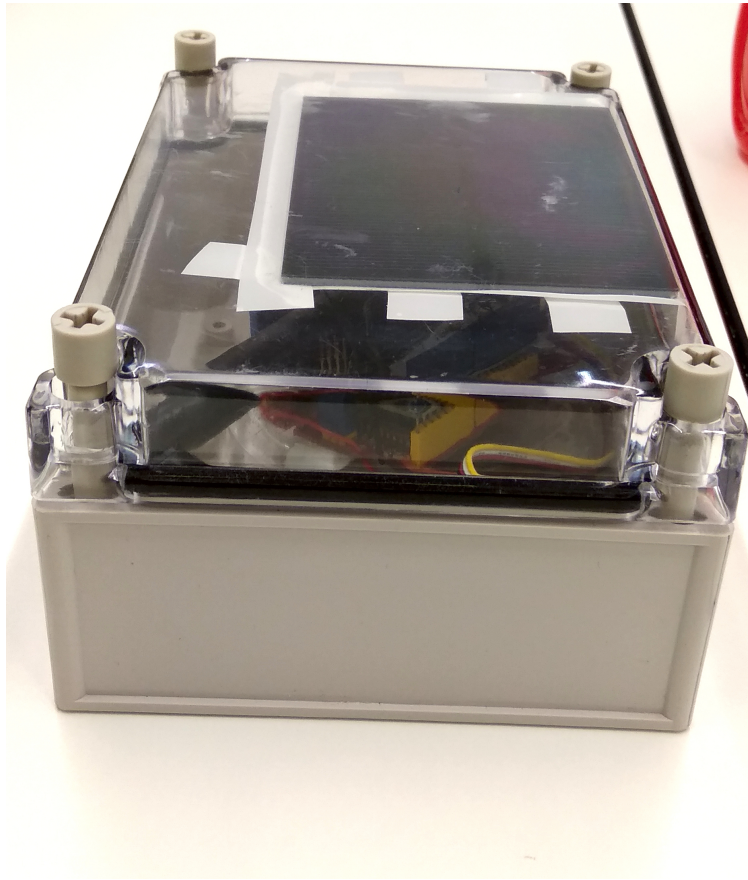


Fig. 1: Sensor node in ABS Waterproof case.



Fig. 2: Solar shield connections with the solar panel and a battery.

Apart from this, to measure the voltage of Lipo Battery we need to connect the VBAT pin to Analog pin A0, so that we can read the data from A0 pin. To achieve this, we need to Short R7 using a 0ohm resistor as shown in the figure here.

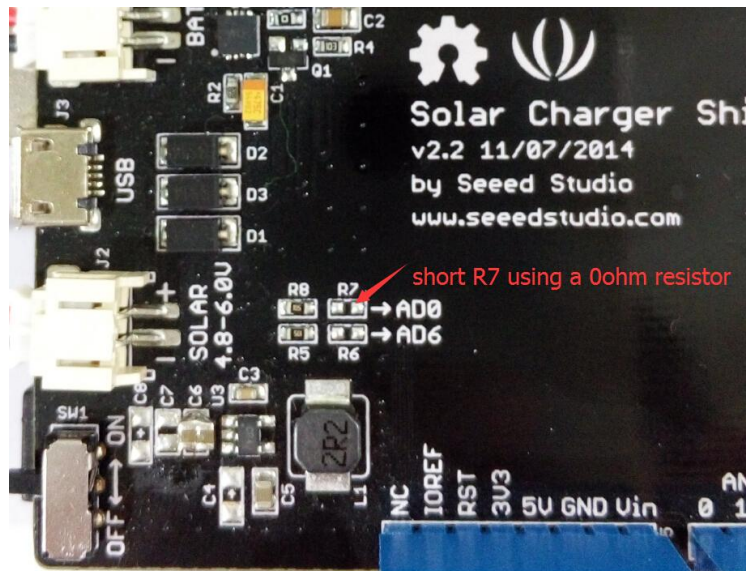


Fig. 3: Short R7 using a 0 ohm resistor for battery voltage measurement.

Final hardware setup looked as following:

Once all these connection were made, the board is connected with a computer using a USB cable. Further, steps of software part needs to be followed.

2.1.3 Software

To create this node, we use Arduino IDE for setting up the Seeeduino LoRaWAN device. First, install the [Seeeduino LoRaWAN board](#) to your Arduino IDE and select the correct port. Then following libraries needs to be installed before compiling the code:

- [Wire.h](#) to communicate with I2C devices.
- [DHT.h](#) for reading DHT-22 sensor.
- [RTCZero.h](#) for controlling internal clock for time.
- [CayenneLPP.h](#) for Cayenne Protocol.

Apart from this LoRaWan.h library is also used but it is bundled within Seeeduino Board and is not required to be separately installed.

Now download and run the [Arduino Sketch for Solar powered Seeeduino sensor node](#) file in the Arduino IDE. This code was created by merging the example code of both the sensors and the ttn-otaa example from the Imic library. Some required changes were made while merging the example codes. The user should change the network session key, app session key and device address in the code before compiling. These keys can be obtained from the TTN, SWM or other service providers.

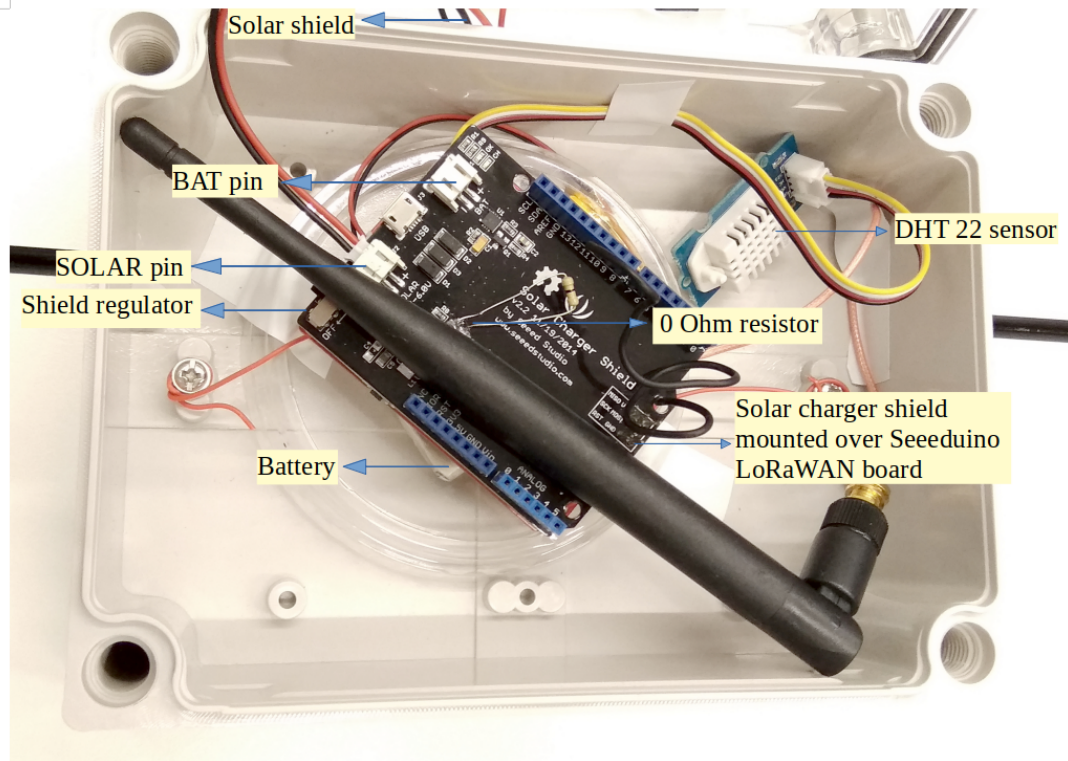


Fig. 4: Final hardware wiring setup.

Listing 1: Modify the keys in highlighted lines.

```

1 // The EUIs and the AppKey must be given in big-endian format, i.e. the
2 // most-significant-byte comes first (as displayed in the TTN console).
3 // For TTN issued AppEUIs the first bytes should be 0x70, 0xB3, 0xD5.
4
5 // void setId(char *DevAddr, char *DevEUI, char *AppEUI);
6 lora.setId(NULL, "00942FBXXXXXXXX", "70B3D57XXXXXXXX");
7
8 // setKey(char *NwksKey, char *AppSKey, char *AppKey);
9 lora.setKey(NULL, NULL, "CB89A0AA43F6C5XXXXXXXXXXXXXXXXXXXX");

```

Following is the example code that can be used to measure the battery voltage of the Seed solar charger shield:

Listing 2: Code for measuring the battery voltage

```

1  BatteryValue = analogRead(analogInPin);
2  // Calculate the battery voltage value
3  outputValue = (float)(BatteryValue*5)/1023*2;
4  // print the results to the serial monitor:
5  SerialUSB.print("Analog value = " );
6  SerialUSB.print(BatteryValue);
7  SerialUSB.print("\t voltage = ");
8  SerialUSB.println(outputValue);
9  SerialUSB.println("V \n");

```

2.1.4 Services

This node is connected using the TheThingsNetwork service. Further, a node-red work bench is used to forward this collected data from the TTN platform to the OGC Sensor Things API configured on the FROST Server. The node-red workbench that was used for forwarding the data is available at *Node red flow for Solar powered Seeeduino sensor node* for Solar powered Seeeduino. To use this node-red-workbench go to the node-red platform <https://iot.gis.bgu.tum.de:1885/>, login with the credentials, go to the options and select Import>Clipboard. Select the downloaded .json file with the given option and click on import. Make necessary changes and deploy the flow.

Datastreams setup for this sensor node on the FROST server can be seen at: [http://iot.gis.bgu.tum.de:8081/FROST-Server-gi3/v1.0/Things\(19\)/Datastreams](http://iot.gis.bgu.tum.de:8081/FROST-Server-gi3/v1.0/Things(19)/Datastreams)

The node-red workbench for this sensor node could be found at: <https://iot.gis.bgu.tum.de:1885/#flow/58838bc1.4ce6a4>

The GRAFANA dash-board for visualizing the collected data is available at: <https://iot.gis.bgu.tum.de:3050/d/TfCVFRNWz/solar-powered-seeeduino-with-dht22?orgId=1&refresh=10s>

2.1.5 Code files

Listing 3: Arduino Sketch for Solar powered Seeeduino sensor node

```

1  #include <DHT.h>
2  #include <RTCZero.h>
3  #include <LoRaWAN.h>
4  #include <Wire.h>
5  #include <CayenneLPP.h>
6
7  // Keep the following line, if the board is a Seeeduino LoRaWAN with GPS,
8  // otherwise comment the line out
9
10 // #define HAS_GPS 1
11
12 const int analogInPin = A0;
13 #define DHTPIN A2
14 #define DHTTYPE DHT22
15
16 DHT dht(DHTPIN, DHTTYPE);
17
18 int BatteryValue = 0;
19 float outputValue = 0;
20

```

(continues on next page)

(continued from previous page)

```

21 RTCZero rtc;
22 char buffer[256]; // buffer for text messages received from the
   ↳ LoRaWAN module for display
23
24 CayenneLPP lpp(51);
25
26 void setup(void)
27 {
28     digitalWrite(38, HIGH); // Provide power to the 4 Grove connectors of
   ↳ the board
29
30     for(int i = 0; i < 26; i++) // Set all pins to HIGH to save power
   ↳ (reduces the
31         { // current drawn during deep sleep by around
   ↳ 0.7mA).
32         if (i!=13) { // Don't switch on the onboard user LED (pin
   ↳ 13).
33             pinMode(i, OUTPUT);
34             digitalWrite(i, HIGH);
35         }
36     }
37
38     delay(5000); // Wait 5 secs after reset/booting to give
   ↳ time for potential upload
39     dht.begin(); // of a new sketch (sketches
   ↳ cannot be uploaded when in sleep mode)
40     SerialUSB.begin(115200); // Initialize USB/serial connection
41     delay(500);
42     // while(!SerialUSB);
43     SerialUSB.println("Seeeduino LoRaWAN board started!");
44
45     // nrgSave.begin(WAKE_RTC_ALARM);
46     // rtc.begin(TIME_H24);
47
48 #ifdef HAS_GPS
49     Serial.begin(9600); // Initialize serial connection to the GPS
   ↳ module
50     delay(500);
51     Serial.write("$PMTK161,0*28\r\n"); // Switch GPS module to standby mode as we don
   ↳ 't use it in this sketch
52 #endif
53
54     lora.init(); // Initialize the LoRaWAN module
55
56     memset(buffer, 0, 256); // clear text buffer
57     lora.getVersion(buffer, 256, 1);
58     memset(buffer, 0, 256); // We call getVersion() two times, because
   ↳ after a reset the LoRaWAN module can be
59     lora.getVersion(buffer, 256, 1); // in sleep mode and then the first call only
   ↳ wakes it up and will not be performed.
60     SerialUSB.print(buffer);
61
62     memset(buffer, 0, 256);
63     lora.getId(buffer, 256, 1);
64     SerialUSB.print(buffer);
65
66     // The following three constants (AppEUI, DevEUI, AppKey) must be changed

```

(continues on next page)

(continued from previous page)

```

67 // for every new sensor node. We are using the LoRaWAN OTAA mode (over the
68 // air activation). Each sensor node must be manually registered in the
69 // TTN console at https://console.thethingsnetwork.org before it can be
70 // started. In the TTN console create a new device with the DevEUI also
71 // being automatically generated. After the registration of the device the
72 // three values can be copied from the TTN console. A detailed explanation
73 // of these steps is given in
74 // https://learn.adafruit.com/the-things-network-for-feather?view=all
75
76 // The EUIs and the AppKey must be given in big-endian format, i.e. the
77 // most-significant-byte comes first (as displayed in the TTN console).
78 // For TTN issued AppEUIs the first bytes should be 0x70, 0xB3, 0xD5.
79
80 // void setId(char *DevAddr, char *DevEUI, char *AppEUI);
81 lora.setId(NULL, "00942FBXXXXXXX", "70B3D57XXXXXXX");
82
83 // setKey(char *NwkSKey, char *AppSKey, char *AppKey);
84 lora.setKey(NULL, NULL, "CB89A0AA43F6C5XXXXXXXXXXXXXXXXXXXX");
85
86 lora.setDeciveMode(LWOTAA); // select OTAA join mode (note that
↪setDeciveMode is not a typo; it is misspelled in the library)
87 // lora.setDataRate(DR5, EU868); // SF7, 125 kbps (highest data rate)
88 lora.setDataRate(DR3, EU868); // SF9, 125 kbps (medium data rate and
↪range)
89 // lora.setDataRate(DR0, EU868); // SF12, 125 kbps (lowest data rate,
↪highest max. distance)
90
91 // lora.setAdaptiveDataRate(false);
92 lora.setAdaptiveDataRate(true); // automatically adapt the data rate
93
94 lora.setChannel(0, 868.1);
95 lora.setChannel(1, 868.3);
96 lora.setChannel(2, 868.5);
97 lora.setChannel(3, 867.1);
98 lora.setChannel(4, 867.3);
99 lora.setChannel(5, 867.5);
100 lora.setChannel(6, 867.7);
101 lora.setChannel(7, 867.9);
102
103 // The following two commands can be left commented out;
104 // TTN works with the default values. (It also works when
105 // uncommenting the commands, though.)
106 // lora.setReceiceWindowFirst(0, 868.1);
107 // lora.setReceiceWindowSecond(869.525, DR0);
108
109 lora.setDutyCycle(false); // for debugging purposes only - should
↪normally be activated
110 lora.setJoinDutyCycle(false); // for debugging purposes only - should
↪normally be activated
111
112 lora.setPower(14); // LoRa transceiver power (14 is the
↪maximum for the 868 MHz band)
113
114 // while(!lora.setOTAAJoin(JOIN));
115 while(!lora.setOTAAJoin(JOIN, 20)); // wait until the node has successfully
↪joined TTN
116

```

(continues on next page)

(continued from previous page)

```

117     lora.setPort(33); // all data packets are sent to LoRaWAN_
    ↪port 33
118 }
119
120 void loop(void)
121 {
122     bool result = false;
123     float temp_hum_val[2] = {0};
124     float temperature, humidity;
125     // Reading temperature or humidity takes about 250 milliseconds!
126     // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
127
128
129     if(!dht.readTempAndHumidity(temp_hum_val)){
130         SerialUSB.print("Humidity: ");
131         SerialUSB.print(humidity = temp_hum_val[0]);
132         SerialUSB.print(" %\t");
133         SerialUSB.print("Temperature: ");
134         SerialUSB.print(temperature = temp_hum_val[1]);
135         SerialUSB.println(" *C");
136     }
137     else{
138         SerialUSB.println("Failed to get temprature and humidity value.");
139     }
140
141     BatteryValue = analogRead(analogInPin);
142     // Calculate the battery voltage value
143     outputValue = (float(BatteryValue)*5)/1023*2;
144     // print the results to the serial monitor:
145     SerialUSB.print("Analog value = ");
146     SerialUSB.print(BatteryValue);
147     SerialUSB.print("\t voltage = ");
148     SerialUSB.println(outputValue);
149     SerialUSB.println("V \n");
150
151     SerialUSB.println("-- LOOP");
152     lpp.reset();
153     lpp.addTemperature(1, temperature);
154     lpp.addRelativeHumidity(2, humidity);
155     lpp.addAnalogInput(3, outputValue);
156
157     result = lora.transferPacket(lpp.getBuffer(), lpp.getSize(), 5); // send the_
    ↪data packet (n byts) with a default timeout of 5 secs
158
159     if(result)
160     {
161         short length;
162         short rssi;
163
164         memset(buffer, 0, 256);
165         length = lora.receivePacket(buffer, 256, &rssi);
166
167         if(length)
168         {
169             SerialUSB.print("Length is: ");
170             SerialUSB.println(length);
171             SerialUSB.print("RSSI is: ");

```

(continues on next page)

(continued from previous page)

```

172     SerialUSB.println(rssi);
173     SerialUSB.print("Data is: ");
174     for(unsigned char i = 0; i < length; i++)
175     {
176         SerialUSB.print("0x");
177         SerialUSB.print(buffer[i], HEX);
178         SerialUSB.print(" ");
179     }
180     SerialUSB.println();
181 }
182 }
183
184 lora.setDeviceLowPower();    // bring the LoRaWAN module to sleep mode
185 doSleep((5*60-8)*1000);    // deep sleep for 292 secs (+ 3 secs transmission_
186 ↪time + 5 secs timeout = 300 secs period)
187 lora.setPort(33);          // send some command to wake up the LoRaWAN module_
188 ↪again
189 }
190
191 // The following function implements deep sleep waiting. When being called the
192 // CPU goes into deep sleep mode (for power saving). It is woken up again by
193 // the CPU-internal real time clock (RTC) after the configured time.
194 //
195 // A similar function would also be available in the standard "ArduinoLowPower"_
196 ↪library.
197 // However, in order to be able to use that library with the Seeeduno LoRaWAN board,
198 // four files in the package "Seeed SAMD boards by Seeed Studio Version 1.3.0" that is
199 // installed using the Arduino IDE board manager need to be patched. The reason is_
200 ↪that
201 // Seeed Studio have not updated their files to a recent Arduino SAMD version yet
202 // and the official "ArduinoLowPower" library provided by the Arduino foundation is
203 // referring to some missing functions. For further information see here:
204 // https://forum.arduino.cc/index.php?topic=603900.0 and here:
205 // https://github.com/arduino/ArduinoCore-samd/commit/
206 ↪b9ac48c782ca4b82ffd7e65bf2c956152386d82b
207
208 void doSleep(uint32_t millis) {
209     if (!rtc.isConfigured()) {    // if called for the first time,
210         rtc.begin(false);        // then initialize the real time clock (RTC)
211     }
212
213     uint32_t now = rtc.getEpoch();
214     rtc.setAlarmEpoch(now + millis/1000);
215     rtc.enableAlarm(rtc.MATCH_HHMMSS);
216
217     rtc.standbyMode();            // bring CPU into deep sleep mode (until woken up_
218     ↪by the RTC)
219 }

```

Listing 4: Node red flow for Solar powered Seeeduno sensor node

```

1  [
2  {
3      "id": "58838bc1.4ce6a4",
4      "type": "tab",
5      "label": "Device1",

```

(continues on next page)

(continued from previous page)

```

6     "disabled": false,
7     "info": ""
8 },
9 {
10    "id": "daeb7602.698d18",
11    "type": "switch",
12    "z": "58838bc1.4ce6a4",
13    "name": "Separate",
14    "property": "key",
15    "propertyType": "msg",
16    "rules": [
17      {
18        "t": "cont",
19        "v": "temperature",
20        "vt": "str"
21      },
22      {
23        "t": "cont",
24        "v": "humidity",
25        "vt": "str"
26      },
27      {
28        "t": "cont",
29        "v": "analog",
30        "vt": "str"
31      },
32      {
33        "t": "else"
34      }
35    ],
36    "checkall": "true",
37    "repair": false,
38    "outputs": 4,
39    "x": 220,
40    "y": 180,
41    "wires": [
42      [
43        "a3a522a5.a81a9"
44      ],
45      [
46        "367717e8.191318"
47      ],
48      [
49        "466fd2c5.586efc"
50      ],
51      []
52    ]
53  },
54  {
55    "id": "e2798231.c9314",
56    "type": "split",
57    "z": "58838bc1.4ce6a4",
58    "name": "",
59    "splt": "\\n",
60    "spltType": "str",
61    "arraySplt": 1,
62    "arraySpltType": "len",

```

(continues on next page)

(continued from previous page)

```

63     "stream": false,
64     "addname": "key",
65     "x": 90,
66     "y": 180,
67     "wires": [
68         [
69             "daeb7602.698d18"
70         ]
71     ]
72 },
73 {
74     "id": "5c3e3ed9.0b4dd",
75     "type": "debug",
76     "z": "58838bc1.4ce6a4",
77     "name": "",
78     "active": false,
79     "tosidebar": true,
80     "console": false,
81     "tostatus": false,
82     "complete": "false",
83     "x": 810,
84     "y": 180,
85     "wires": []
86 },
87 {
88     "id": "367717e8.191318",
89     "type": "function",
90     "z": "58838bc1.4ce6a4",
91     "name": "Humidity",
92     "func": "var humValue = msg.payload.valueOf();\nvar newMessage = { payload:
↪ { \"result\": humValue, \"Datastream\": {\"@iot.id\": 102} } };\nnewMessage.headers
↪ = {\"Content-type\" : \"application/json\"}\nreturn newMessage;",
93     "outputs": 1,
94     "noerr": 0,
95     "x": 440,
96     "y": 200,
97     "wires": [
98         [
99             "c777922b.84784"
100         ]
101     ]
102 },
103 {
104     "id": "c777922b.84784",
105     "type": "http request",
106     "z": "58838bc1.4ce6a4",
107     "name": "POST Observation",
108     "method": "POST",
109     "ret": "obj",
110     "paytoqs": false,
111     "url": "http://iot.gis.bgu.tum.de:8081/FROST-Server-gi3/v1.0/Observations",
112     "tls": "",
113     "proxy": "",
114     "authType": "basic",
115     "x": 630,
116     "y": 180,
117     "wires": [

```

(continues on next page)

(continued from previous page)

```

118         [
119             "5c3e3ed9.0b4dd"
120         ]
121     ],
122 },
123 {
124     "id": "a3a522a5.a81a9",
125     "type": "function",
126     "z": "58838bc1.4ce6a4",
127     "name": "Temperature",
128     "func": "var tempValue = msg.payload.valueOf();\nvar newMessage = { payload:
↪{ \"result\": tempValue, \"Datastream\": {\"@iot.id\": 101}} }; \nnewMessage.
↪headers = {\"Content-type\" : \"application/json\"}\nreturn newMessage;",
129     "outputs": 1,
130     "noerr": 0,
131     "x": 450,
132     "y": 160,
133     "wires": [
134         [
135             "c777922b.84784"
136         ]
137     ],
138 },
139 {
140     "id": "41ae6239.73f9bc",
141     "type": "ttn uplink",
142     "z": "58838bc1.4ce6a4",
143     "name": "TTN Input",
144     "app": "58ceff1f.8576a",
145     "dev_id": "tum-gis-device1",
146     "field": "",
147     "x": 80,
148     "y": 60,
149     "wires": [
150         [
151             "491bb4da.0eb58c"
152         ]
153     ],
154 },
155 {
156     "id": "491bb4da.0eb58c",
157     "type": "cayennelp-decoder",
158     "z": "58838bc1.4ce6a4",
159     "name": "",
160     "x": 280,
161     "y": 60,
162     "wires": [
163         [
164             "e2798231.c9314",
165             "f2d3534b.0f44f"
166         ]
167     ],
168 },
169 {
170     "id": "466fd2c5.586efc",
171     "type": "function",
172     "z": "58838bc1.4ce6a4",

```

(continues on next page)

(continued from previous page)

```

173     "name": "Battery Voltage",
174     "func": "var batteryvolt = msg.payload.valueOf();\nvar newMessage = {
↪ payload: { \"result\": batteryvolt, \"DataStream\": {\"@iot.id\": 104}} };
↪ \nnewMessage.headers = {\"Content-type\" : \"application/json\"}\nreturn newMessage;
↪ ",
175     "outputs": 1,
176     "noerr": 0,
177     "x": 440,
178     "y": 240,
179     "wires": [
180         [
181             "c777922b.84784"
182         ]
183     ],
184 },
185 {
186     "id": "f2d3534b.0f44f",
187     "type": "debug",
188     "z": "58838bc1.4ce6a4",
189     "name": "",
190     "active": true,
191     "tosidebar": true,
192     "console": false,
193     "tostatus": false,
194     "complete": "false",
195     "x": 490,
196     "y": 60,
197     "wires": []
198 },
199 {
200     "id": "58ceff1f.8576a",
201     "type": "ttn app",
202     "z": "",
203     "appId": "gis-tum-sensors",
204     "accessKey": "ttn-account-ACCESSKEY_HERE",
205     "discovery": "discovery.thethingsnetwork.org:1900"
206 }
207 ]

```

2.1.6 References

- *Arduino Sketch for Solar powered Seeeduino sensor node*
- *Node red flow for Solar powered Seeeduino sensor node*
- [Wiki guide for Seeeduino LoRaWAN board](#)
- [Adding Seeed boards to Arduino IDE](#)
- [Seeed Solar charger shield guide](#)

2.2 Indoor Ambient Monitoring

This sensor node is made to showcase a use-case of LoRaWAN for indoor ambience monitoring. For achieving this a multitude of sensors were used which can monitor the quality of the ambience. In this example we measure parameters

such as temperature, humidity, air pressure, air quality, CO2, loudness, gas, PM2.5, and light.

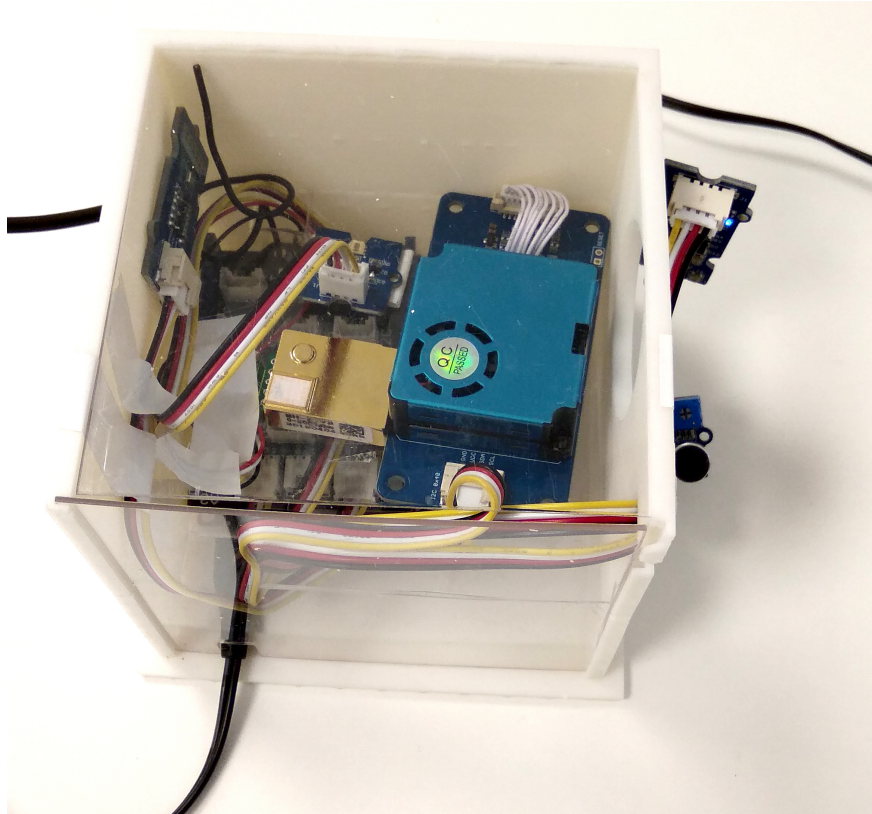


Fig. 5: Hardware setup.

2.2.1 Hardware

To realize the objective, following components were used:

- Seeeduino LoRaWAN board V4.2
- Grove Base Shield Arduino V2
- Grove - Air Quality Sensor
- Grove - Loudness Sensor
- Grove - Digital Light Sensor
- Grove - BME680 Sensor
- Grove - Laser PM2.5 Sensor (HM3301)
- MHZ19B CO2 Sensor
- Micro USB Charger

2.2.2 Wiring setup

First of all, the grove base shield was connected over the Seeeduino LoRaWAN board. The board was set at the 5V mode. Then, the sensor connections were made using the connector cables as following:

- Loudness Sensor – Analog Pin A0
- PM 2.5 Sensor – I2C pin
- Digital Light Sensor – I2C pin
- BME680 Sensor – I2C pin
- MHZ19B CO2 Sensor – Digital Pin D4
- Air Quality Sensor - A2

Apart from this, there is no need of any other wiring in this case.

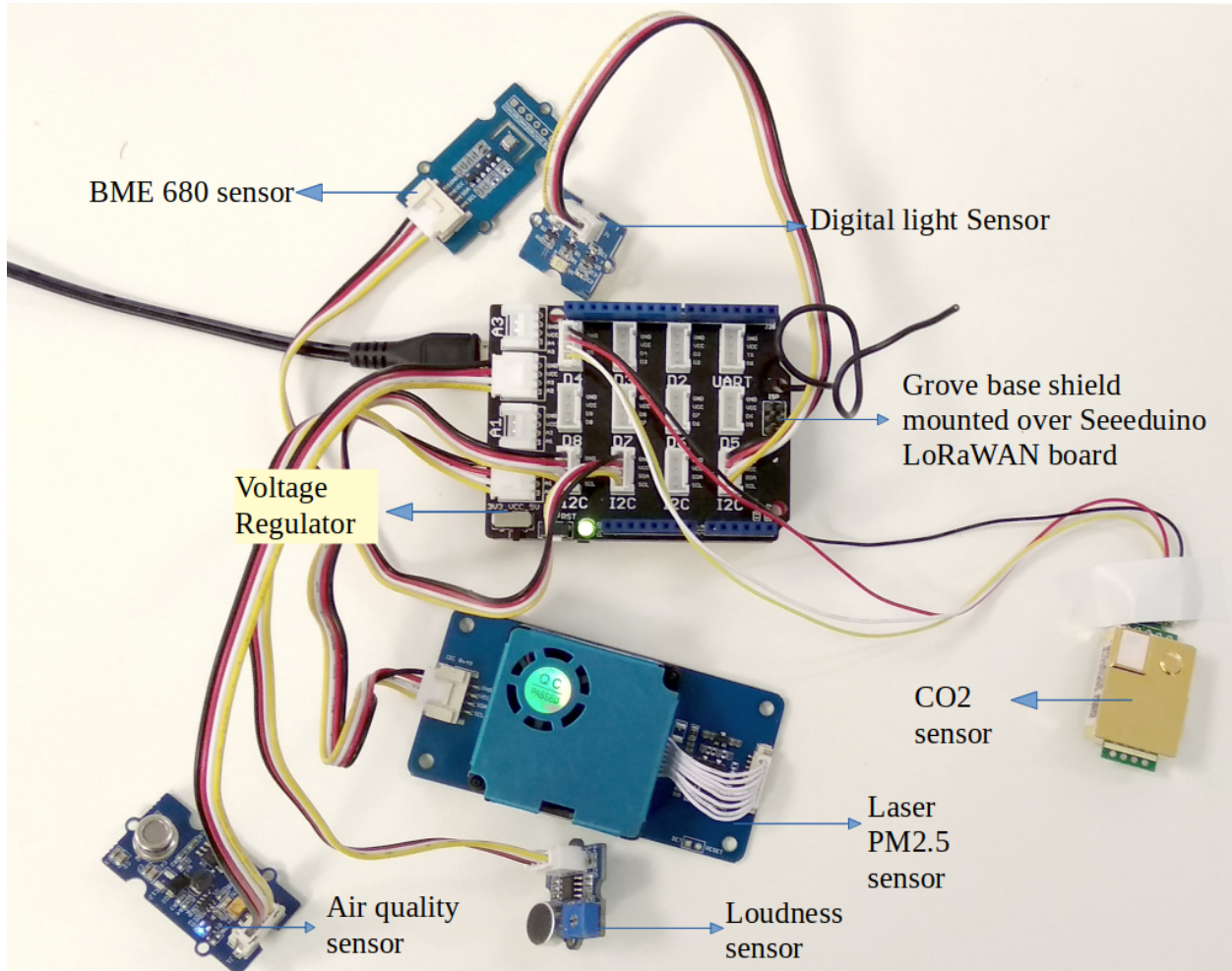


Fig. 6: Hardware connections.

Once all these connection were made, the board is connected with a computer using a USB cable. Further, steps of software part needs to be followed.

2.2.3 Software

To create this node, we use Arduino IDE for setting up the Seedeuino LoRaWAN device. First, install the [Seedeuino LoRaWAN board](#) board to your Arduino IDE and select the correct port. Then following libraries needs to be installed before compiling the code:

- [Digital_Light_TSL2561.h](#) for Digital Light Sensor
- [Air_Quality_Sensor.h](#) for Air Quality Sensor
- [Seeed_bme680.h](#) for BME680 Sensor
- [Seeed_HM330X.h](#) for Laser PM2.5 Sensor
- [MHZ19.h](#) for MHZ19B CO2 Sensor
- [Wire.h](#) to communicate with I2C devices
- [SoftwareSerial.h](#) for Serial Communication
- [RTCZero.h](#) for controlling internal clock for time
- [CayenneLPP.h](#) for Cayenne Protocol

Apart from this LoRaWan.h library is also used but it is bundled within Seeeduino Board and is not required to be separately installed.

Now download and run the *Arduino Sketch for Indoor Ambient Monitoring sensor node* file in the Arduino IDE. This code was created by merging the example code of each of these attached sensor and the ttn-otaa example from the Seeeduino board. Some required changes were made while merging the example codes. For example, as there are multiple sensors each needs to be defined with a unique name. So, here HM330X was named as sensor while AirqualitySensor as sensors.

Listing 5: Setup the sensors in *Arduino Sketch for Indoor Ambient Monitoring sensor node*

```
1 AirQualitySensor sensors(A2);
2
3 SoftwareSerial ss(4,5);
4 MHZ19 mhz(&ss);
5
6 #define BME_SCK 13
7 #define BME_MISO 12
8 #define BME_MOSI 11
9 #define BME_CS 10
10
11 #define IIC_ADDR uint8_t(0x76)
12 Seeed_BME680 bme680(IIC_ADDR);
13
14 int loudness,a;
15
16 HM330X sensor;
17 u8 buf[30];
```

The user should change the network session key, app session key and device address in the code before compiling. These keys can be obtained from the TTN, SWM or any other service providers.

Listing 6: Modify the keys in highlighted lines

```
1 // The EUIs and the AppKey must be given in big-endian format, i.e. the
2 // most-significant-byte comes first (as displayed in the TTN console).
3 // For TTN issued AppEUIs the first bytes should be 0x70, 0xB3, 0xD5.
4
5 // void setId(char *DevAddr, char *DevEUI, char *AppEUI);
6 lora.setId(NULL, "00942FBXXXXXXXXXX", "70B3D57XXXXXXXXXX");
7
```

(continues on next page)

(continued from previous page)

```

8 // setKey(char *NwkSKey, char *AppSKey, char *AppKey);
9 lora.setKey(NULL, NULL, "CB89A0AA43F6C5XXXXXXXXXXXXXXXXXXXX");

```

2.2.4 Services

This node is connected using the TheThingsNetwork service. Further, a node-red work bench is used to forward this collected data from the TTN platform to the OGC Sensor Things API configured on the FROST Server. The node-red workbench that was used for forwarding the data is available at [Node red flow for Indoor Ambient Monitoring sensor node](https://iot.gis.bgu.tum.de:1885/). To use this node-red-workbench go to the node-red platform <https://iot.gis.bgu.tum.de:1885/>, login with the credentials, go to the options and select Import>Clipboard. Select the downloaded .json file with the given option and click on import. Make necessary changes and deploy the flow.

Datastreams setup for this sensor node on the FROST server can be seen at: [http://iot.gis.bgu.tum.de:8081/FROST-Server-gi3/v1.0/Things\(21\)/Datastreams](http://iot.gis.bgu.tum.de:8081/FROST-Server-gi3/v1.0/Things(21)/Datastreams)

The node-red workbench for this sensor node could be found at: <https://iot.gis.bgu.tum.de:1885/#flow/7d5c6b14.d2af94>

The GRAFANA dash-board for visualizing the collected data is available at: <https://iot.gis.bgu.tum.de:3050/d/jDJ1li1Wz/indoor-ambient-monitoring-with-seeeduino-lorawan-and-sensors?orgId=1>

2.2.5 Code files

Listing 7: Arduino Sketch for Indoor Ambient Monitoring sensor node

```

1  #include <Wire.h>
2  #include <Digital_Light_TSL2561.h>
3  #include "Air_Quality_Sensor.h"
4  #include "seeed_bme680.h"
5  #include "Seeed_HM330X.h"
6  #include <SoftwareSerial.h>
7  #include <MHZ19.h>
8  #include <RTCZero.h>
9  #include <LoRaWAN.h>
10 #include <CayenneLPP.h>
11
12 RTCZero rtc;
13 char buffer[256]; // buffer for text messages received from the
14   ↳ LoRaWAN module for display
15
16 CayenneLPP lpp(51);
17
18 AirQualitySensor sensors(A2);
19
20 SoftwareSerial ss(4,5);
21 MHZ19 mhz(&ss);
22
23 #define BME_SCK 13
24 #define BME_MISO 12
25 #define BME_MOSI 11
26 #define BME_CS 10
27
28 #define IIC_ADDR uint8_t(0x76)
29 Seeed_BME680 bme680(IIC_ADDR);

```

(continues on next page)

(continued from previous page)

```

29
30 int loudness,a;
31
32 HM330X sensor;
33 u8 buf[30];
34
35 const char *str[]={"sensor num: ", "PM1.0 concentration(CF=1,Standard particulate_
↪matter,unit:ug/m3): ",
36                  "PM2.5 concentration(CF=1,Standard particulate matter,unit:ug/
↪m3): ",
37                  "PM10 concentration(CF=1,Standard particulate matter,unit:ug/m3):
↪",
38                  "PM1.0 concentration(Atmospheric environment,unit:ug/m3): ",
39                  "PM2.5 concentration(Atmospheric environment,unit:ug/m3): ",
40                  "PM10 concentration(Atmospheric environment,unit:ug/m3): ",
41                  };
42
43 err_t print_result(const char* str,u16 value)
44 {
45     if(NULL==str)
46         return ERROR_PARAM;
47     SerialUSB.print(str);
48     SerialUSB.println(value);
49     return NO_ERROR;
50 }
51
52 /*parse buf with 29 u8-data*/
53 err_t parse_result(u8 *data)
54 {
55     u16 value=0;
56     err_t NO_ERROR;
57     if(NULL==data)
58         return ERROR_PARAM;
59     for(int i=1;i<8;i++)
60     {
61         value = (u16)data[i*2]<<8|data[i*2+1];
62         print_result(str[i-1],value);
63         if(i==6)
64         {
65             a=value;
66             SerialUSB.println(a);
67         }
68     }
69
70 err_t parse_result_value(u8 *data)
71 {
72     if(NULL==data)
73         return ERROR_PARAM;
74     for(int i=0;i<28;i++)
75     {
76         SerialUSB.print(data[i],HEX);
77         SerialUSB.print(" ");
78         if((0==(i)%5) || (0==i))
79         {
80             SerialUSB.println(" ");
81         }
82     }

```

(continues on next page)

(continued from previous page)

```

83     u8 sum=0;
84     for(int i=0;i<28;i++)
85     {
86         sum+=data[i];
87     }
88     if(sum!=data[28])
89     {
90         SerialUSB.println("wrong checksum!!!!");
91     }
92     SerialUSB.println(" ");
93     SerialUSB.println(" ");
94     return NO_ERROR;
95 }
96
97
98 void setup()
99 {
100     Wire.begin();
101
102     for(int i = 0; i < 26; i ++)           // Set all pins to HIGH to save power (reduces
↪ the                                     // current drawn during deep sleep by around
103     {                                     // 0.7mA).
104         if (i!=13) {                     // Don't switch on the onboard user LED (pin
↪ 13).
105             pinMode(i, OUTPUT);
106             digitalWrite(i, HIGH);
107         }
108     }
109
110     delay(5000);
111
112     SerialUSB.begin(115200);
113     delay(100);
114     SerialUSB.println("SerialUSB start");
115
116     lora.init();                         // Initialize the LoRaWAN module
117
118     memset(buffer, 0, 256);              // clear text buffer
119     lora.getVersion(buffer, 256, 1);
120     memset(buffer, 0, 256);              // We call getVersion() two times, because
↪ after a reset the LoRaWAN module can be
121     lora.getVersion(buffer, 256, 1);     // in sleep mode and then the first call only
↪ wakes it up and will not be performed.
122     SerialUSB.print(buffer);
123
124     memset(buffer, 0, 256);
125     lora.getId(buffer, 256, 1);
126     SerialUSB.print(buffer);
127
128     // The following three constants (AppEUI, DevEUI, AppKey) must be changed
129     // for every new sensor node. We are using the LoRaWAN OTAA mode (over the
130     // air activation). Each sensor node must be manually registered in the
131     // TTN console at https://console.thethingsnetwork.org before it can be
132     // started. In the TTN console create a new device with the DevEUI also
133     // being automatically generated. After the registration of the device the
134     // three values can be copied from the TTN console. A detailed explanation

```

(continues on next page)

(continued from previous page)

```

135 // of these steps is given in
136 // https://learn.adafruit.com/the-things-network-for-feather?view=all
137
138 // The EUIs and the AppKey must be given in big-endian format, i.e. the
139 // most-significant-byte comes first (as displayed in the TTN console).
140 // For TTN issued AppEUIs the first bytes should be 0x70, 0xB3, 0xD5.
141
142 // void setId(char *DevAddr, char *DevEUI, char *AppEUI);
143 lora.setId(NULL, "00942FBXXXXXXX", "70B3D57XXXXXXX");
144
145 // setKey(char *NwkSKey, char *AppSKey, char *AppKey);
146 lora.setKey(NULL, NULL, "CB89A0AA43F6C5XXXXXXXXXXXXXXXXXXXX");
147
148 lora.setDeciveMode(LWOTAA); // select OTAA join mode (note that
↪setDeciveMode is not a typo; it is misspelled in the library)
149 // lora.setDataRate(DR5, EU868); // SF7, 125 kbps (highest data rate)
150 lora.setDataRate(DR3, EU868); // SF9, 125 kbps (medium data rate and
↪range)
151 // lora.setDataRate(DR0, EU868); // SF12, 125 kbps (lowest data rate,
↪highest max. distance)
152
153 // lora.setAdaptiveDataRate(false);
154 lora.setAdaptiveDataRate(true); // automatically adapt the data rate
155
156 lora.setChannel(0, 868.1);
157 lora.setChannel(1, 868.3);
158 lora.setChannel(2, 868.5);
159 lora.setChannel(3, 867.1);
160 lora.setChannel(4, 867.3);
161 lora.setChannel(5, 867.5);
162 lora.setChannel(6, 867.7);
163 lora.setChannel(7, 867.9);
164
165 // The following two commands can be left commented out;
166 // TTN works with the default values. (It also works when
167 // uncommenting the commands, though.)
168 // lora.setReceiceWindowFirst(0, 868.1);
169 // lora.setReceiceWindowSecond(869.525, DR0);
170
171 lora.setDutyCycle(false); // for debugging purposes only - should
↪normally be activated
172 lora.setJoinDutyCycle(false); // for debugging purposes only - should
↪normally be activated
173
174 lora.setPower(14); // LoRa transceiver power (14 is the
↪maximum for the 868 MHz band)
175
176 // while(!lora.setOTAAJoin(JOIN));
177 while(!lora.setOTAAJoin(JOIN, 20)); // wait until the node has successfully
↪joined TTN
178
179 lora.setPort(33);
180
181 if(sensor.init())
182 {
183     SerialUSB.println("HM330X init failed!!!");
184     while(1);

```

(continues on next page)

(continued from previous page)

```

185     }
186
187     if (sensors.init()) {
188         SerialUSB.println("Sensor ready.");
189     }
190     else {
191         SerialUSB.println("Sensor ERROR!");
192     }
193
194     TSL2561.init();
195
196     while (!bme680.init())
197     {
198         SerialUSB.println("bme680 init failed ! can't find device!");
199         delay(10000);
200     }
201
202     ss.begin(9600);
203
204 }
205
206 void loop()
207 {
208     bool result = false;
209     float temperature, humidity, pressure, airquality, light, gas, CO2;
210
211     loudness = analogRead(0);
212     SerialUSB.print("The Loudness Sensor value is: ");
213     SerialUSB.println(loudness);
214     SerialUSB.println();
215     delay(3000);
216
217     int quality = sensors.slope();
218
219     SerialUSB.print("Air Quality Sensor value is: ");
220     SerialUSB.println(airquality=sensors.getValue());
221
222     if (quality == AirQualitySensor::FORCE_SIGNAL) {
223         SerialUSB.println("High pollution! Force signal active.");
224     }
225     else if (quality == AirQualitySensor::HIGH_POLLUTION) {
226         SerialUSB.println("High pollution!");
227     }
228     else if (quality == AirQualitySensor::LOW_POLLUTION) {
229         SerialUSB.println("Low pollution!");
230     }
231     else if (quality == AirQualitySensor::FRESH_AIR) {
232         SerialUSB.println("Fresh air.");
233     }
234     SerialUSB.println();
235     delay(3000);
236
237     SerialUSB.print("The Light Sensor value is: ");
238     SerialUSB.println(light=TSL2561.readVisibleLux());
239     SerialUSB.println();
240     delay(3000);
241

```

(continues on next page)

(continued from previous page)

```

242     if(sensor.read_sensor_value(buf,29))
243     {
244         SerialUSB.println("HM330X read result failed!!!");
245     }
246     parse_result_value(buf);
247     parse_result(buf);
248     SerialUSB.println(" ");
249     delay(3000);
250
251     if (bme680.read_sensor_data())
252     {
253         SerialUSB.println("Failed to perform reading :(");
254         return;
255     }
256     SerialUSB.print("temperature ==>> ");
257     SerialUSB.print(temperature = bme680.sensor_result_value.temperature);
258     SerialUSB.println(" C");
259
260     SerialUSB.print("pressure ==>> ");
261     SerialUSB.print(pressure = bme680.sensor_result_value.pressure/ 1000.0);
262     SerialUSB.println(" KPa");
263
264     SerialUSB.print("humidity ==>> ");
265     SerialUSB.print(humidity = bme680.sensor_result_value.humidity);
266     SerialUSB.println(" %");
267
268     SerialUSB.print("gas ==>> ");
269     SerialUSB.print(gas = bme680.sensor_result_value.gas/ 1000.0);
270     SerialUSB.println(" Kohms");
271
272     SerialUSB.println();
273
274     delay(3000);
275
276     MHZ19_RESULT response = mhz.retrieveData();
277     if (response == MHZ19_RESULT_OK)
278     {
279         SerialUSB.print(F("CO2: "));
280         SerialUSB.println(CO2=mhz.getCO2());
281         SerialUSB.print(F("Min CO2: "));
282         SerialUSB.println(mhz.getMinCO2());
283         SerialUSB.print(F("Temperature: "));
284         SerialUSB.println(mhz.getTemperature());
285         SerialUSB.print(F("Accuracy: "));
286         SerialUSB.println(mhz.getAccuracy());
287         SerialUSB.println();
288     }
289     else
290     {
291         SerialUSB.print(F("Error, code: "));
292         SerialUSB.println(response);
293     }
294
295     lpp.reset();
296     lpp.addTemperature(1, temperature);
297     lpp.addRelativeHumidity(2, humidity);
298     lpp.addAnalogInput(3, airquality);

```

(continues on next page)

(continued from previous page)

```

299     lpp.addLuminosity(4, light);
300     lpp.addBarometricPressure(5, pressure);
301     lpp.addLuminosity(6, CO2);
302     lpp.addAnalogInput(7, gas);
303     lpp.addLuminosity(8, loudness);
304     lpp.addLuminosity(9, a);
305     result = lora.transferPacket(lpp.getBuffer(), lpp.getSize(), 5);    // send the_
↪data packet (n byts) with a default timeout of 5 secs
306
307     if(result)
308     {
309         short length;
310         short rssi;
311
312         memset(buffer, 0, 256);
313         length = lora.receivePacket(buffer, 256, &rssi);
314
315         if(length)
316         {
317             SerialUSB.print("Length is: ");
318             SerialUSB.println(length);
319             SerialUSB.print("RSSI is: ");
320             SerialUSB.println(rssi);
321             SerialUSB.print("Data is: ");
322             for(unsigned char i = 0; i < length; i++)
323             {
324                 SerialUSB.print("0x");
325                 SerialUSB.print(buffer[i], HEX);
326                 SerialUSB.print(" ");
327             }
328             SerialUSB.println();
329         }
330     }
331
332     lora.setDeviceLowPower();    // bring the LoRaWAN module to sleep mode
333     doSleep((5*60-8)*1000);    // deep sleep for 292 secs (+ 3 secs transmission_
↪time + 5 secs timeout = 300 secs period)
334     lora.setPort(33);
335
336 }
337
338 void doSleep(uint32_t millis) {
339     if (!rtc.isConfigured()) {    // if called for the first time,
340         rtc.begin(false);        // then initialize the real time clock (RTC)
341     }
342
343     uint32_t now = rtc.getEpoch();
344     rtc.setAlarmEpoch(now + millis/1000);
345     rtc.enableAlarm(rtc.MATCH_HHMMSS);
346
347     rtc.standbyMode();    // bring CPU into deep sleep mode (until woken up_
↪by the RTC)
348 }

```

Listing 8: Node red flow for Indoor Ambient Monitoring sensor node

```

1  [
2    {
3      "id": "7d5c6b14.d2af94",
4      "type": "tab",
5      "label": "Device3",
6      "disabled": false,
7      "info": ""
8    },
9    {
10     "id": "4d581d8.f14c0e4",
11     "type": "switch",
12     "z": "7d5c6b14.d2af94",
13     "name": "Separate",
14     "property": "key",
15     "propertyType": "msg",
16     "rules": [
17       {
18         "t": "cont",
19         "v": "temperature_1",
20         "vt": "str"
21       },
22       {
23         "t": "cont",
24         "v": "humidity",
25         "vt": "str"
26       },
27       {
28         "t": "cont",
29         "v": "analog_in_3",
30         "vt": "str"
31       },
32       {
33         "t": "cont",
34         "v": "luminosity_4",
35         "vt": "str"
36       },
37       {
38         "t": "cont",
39         "v": "barometric",
40         "vt": "str"
41       },
42       {
43         "t": "cont",
44         "v": "luminosity_6",
45         "vt": "str"
46       },
47       {
48         "t": "cont",
49         "v": "analog_in_7",
50         "vt": "str"
51       },
52       {
53         "t": "cont",
54         "v": "luminosity_8",
55         "vt": "str"

```

(continues on next page)

(continued from previous page)

```

56         },
57         {
58             "t": "cont",
59             "v": "luminosity_9",
60             "vt": "str"
61         }
62     ],
63     "checkall": "true",
64     "repair": false,
65     "outputs": 9,
66     "x": 220,
67     "y": 180,
68     "wires": [
69         [
70             "92425d9f.bca98"
71         ],
72         [
73             "9b919750.d6fff8"
74         ],
75         [
76             "620b1ab2.a69224"
77         ],
78         [
79             "eee677bf.fe16d8"
80         ],
81         [
82             "4b424590.139b4c"
83         ],
84         [
85             "bc3f8433.7b89f8"
86         ],
87         [
88             "13968bca.c2cd34"
89         ],
90         [
91             "c7fcb372.4b2df"
92         ],
93         [
94             "b52fa683.7c5fb8"
95         ]
96     ]
97 },
98 {
99     "id": "9010a80d.dfcd8b",
100     "type": "split",
101     "z": "7d5c6b14.d2af94",
102     "name": "",
103     "spltt": "\\n",
104     "splttType": "str",
105     "arraySpltt": 1,
106     "arraySplttType": "len",
107     "stream": false,
108     "addname": "key",
109     "x": 90,
110     "y": 180,
111     "wires": [
112         [

```

(continues on next page)

(continued from previous page)

```

113         "4d581d8.f14c0e4"
114     ]
115 ]
116 },
117 {
118     "id": "a3b86e6f.56637",
119     "type": "debug",
120     "z": "7d5c6b14.d2af94",
121     "name": "",
122     "active": false,
123     "tosidebar": true,
124     "console": false,
125     "tostatus": false,
126     "complete": "false",
127     "x": 870,
128     "y": 240,
129     "wires": []
130 },
131 {
132     "id": "9b919750.d6fff8",
133     "type": "function",
134     "z": "7d5c6b14.d2af94",
135     "name": "Humidity",
136     "func": "var humValue = msg.payload.valueOf();\nvar newMessage = { payload:
↵ { \"result\": humValue, \"Datastream\": {\"@iot.id\": 113}} }; \nnewMessage.headers_
↵ = {\"Content-type\" : \"application/json\"}\nreturn newMessage;",
137     "outputs": 1,
138     "noerr": 0,
139     "x": 440,
140     "y": 200,
141     "wires": [
142         [
143             "c67491fc.f4755"
144         ]
145     ]
146 },
147 {
148     "id": "c67491fc.f4755",
149     "type": "http request",
150     "z": "7d5c6b14.d2af94",
151     "name": "POST Observation",
152     "method": "POST",
153     "ret": "obj",
154     "paytoqs": false,
155     "url": "http://iot.gis.bgu.tum.de:8081/FROST-Server-gi3/v1.0/Observations",
156     "tls": "",
157     "proxy": "",
158     "authType": "basic",
159     "x": 690,
160     "y": 240,
161     "wires": [
162         [
163             "a3b86e6f.56637"
164         ]
165     ]
166 },
167 {

```

(continues on next page)

(continued from previous page)

```

168     "id": "92425d9f.bca98",
169     "type": "function",
170     "z": "7d5c6b14.d2af94",
171     "name": "Temperature",
172     "func": "var tempValue = msg.payload.valueOf();\nvar newMessage = { payload:
↪{ \"result\": tempValue, \"Datastream\": {\"@iot.id\": 112}} }; \nnewMessage.
↪headers = {\"Content-type\" : \"application/json\"}\nreturn newMessage;",
173     "outputs": 1,
174     "noerr": 0,
175     "x": 450,
176     "y": 160,
177     "wires": [
178         [
179             "c67491fc.f4755"
180         ]
181     ],
182 },
183 {
184     "id": "a16b3beb.1a5028",
185     "type": "debug",
186     "z": "7d5c6b14.d2af94",
187     "name": "",
188     "active": true,
189     "tosidebar": true,
190     "console": false,
191     "tostatus": false,
192     "complete": "payload",
193     "targetType": "msg",
194     "x": 490,
195     "y": 60,
196     "wires": []
197 },
198 {
199     "id": "681442b7.1266ac",
200     "type": "ttn uplink",
201     "z": "7d5c6b14.d2af94",
202     "name": "TTN Input",
203     "app": "58ceff1f.8576a",
204     "dev_id": "tum-gis-device3",
205     "field": "",
206     "x": 80,
207     "y": 60,
208     "wires": [
209         [
210             "531b0b35.751284"
211         ]
212     ],
213 },
214 {
215     "id": "531b0b35.751284",
216     "type": "cayennelp-decoder",
217     "z": "7d5c6b14.d2af94",
218     "name": "",
219     "x": 260,
220     "y": 60,
221     "wires": [
222         [

```

(continues on next page)

(continued from previous page)

```

223         "9010a80d.dfcd8b8",
224         "a16b3beb.1a5028"
225     ]
226 ]
227 },
228 {
229     "id": "620b1ab2.a69224",
230     "type": "function",
231     "z": "7d5c6b14.d2af94",
232     "name": "Air Quality",
233     "func": "var quality = msg.payload.valueOf();\nvar newMessage = { payload: {
↪ \"result\": quality, \"Datastream\": {\"@iot.id\": 119}} }; \nnewMessage.headers =
↪ {\"Content-type\" : \"application/json\"}\nreturn newMessage;",
234     "outputs": 1,
235     "noerr": 0,
236     "x": 450,
237     "y": 240,
238     "wires": [
239         [
240             "c67491fc.f4755"
241         ]
242     ],
243 },
244 {
245     "id": "eee677bf.fe16d8",
246     "type": "function",
247     "z": "7d5c6b14.d2af94",
248     "name": "Light",
249     "func": "var light = msg.payload.valueOf();\nvar newMessage = { payload: {
↪ \"result\": light, \"Datastream\": {\"@iot.id\": 117}} }; \nnewMessage.headers = {
↪ \"Content-type\" : \"application/json\"}\nreturn newMessage;",
250     "outputs": 1,
251     "noerr": 0,
252     "x": 430,
253     "y": 280,
254     "wires": [
255         [
256             "c67491fc.f4755"
257         ]
258     ],
259 },
260 {
261     "id": "4b424590.139b4c",
262     "type": "function",
263     "z": "7d5c6b14.d2af94",
264     "name": "Barometric Pressure",
265     "func": "var pressure = msg.payload.valueOf();\nvar newMessage = { payload:
↪ { \"result\": pressure, \"Datastream\": {\"@iot.id\": 114}} }; \nnewMessage.headers
↪ = {\"Content-type\" : \"application/json\"}\nreturn newMessage;",
266     "outputs": 1,
267     "noerr": 0,
268     "x": 480,
269     "y": 320,
270     "wires": [
271         [
272             "c67491fc.f4755"
273         ]

```

(continues on next page)

(continued from previous page)

```

274     ]
275 },
276 {
277     "id": "bc3f8433.7b89f8",
278     "type": "function",
279     "z": "7d5c6b14.d2af94",
280     "name": "co2",
281     "func": "var co2 = msg.payload.valueOf();\nvar newMessage = { payload: { \
↪ "result\": co2, \"Datastream\": {\"@iot.id\": 118}} }; \nnewMessage.headers = { \
↪ \"Content-type\" : \"application/json\" }\nreturn newMessage;",
282     "outputs": 1,
283     "noerr": 0,
284     "x": 430,
285     "y": 360,
286     "wires": [
287         [
288             "c67491fc.f4755"
289         ]
290     ]
291 },
292 {
293     "id": "13968bca.c2cd34",
294     "type": "function",
295     "z": "7d5c6b14.d2af94",
296     "name": "Gas",
297     "func": "var gas = msg.payload.valueOf();\nvar newMessage = { payload: { \
↪ \"result\": gas, \"Datastream\": {\"@iot.id\": 121}} }; \nnewMessage.headers = { \
↪ \"Content-type\" : \"application/json\" }\nreturn newMessage;",
298     "outputs": 1,
299     "noerr": 0,
300     "x": 430,
301     "y": 400,
302     "wires": [
303         [
304             "c67491fc.f4755"
305         ]
306     ]
307 },
308 {
309     "id": "c7fcb372.4b2df",
310     "type": "function",
311     "z": "7d5c6b14.d2af94",
312     "name": "Loudness",
313     "func": "var loudness = msg.payload.valueOf();\nvar newMessage = { payload:
↪ { \"result\": loudness, \"Datastream\": {\"@iot.id\": 120}} }; \nnewMessage.headers
↪ = { \"Content-type\" : \"application/json\" }\nreturn newMessage;",
314     "outputs": 1,
315     "noerr": 0,
316     "x": 440,
317     "y": 440,
318     "wires": [
319         [
320             "c67491fc.f4755"
321         ]
322     ]
323 },
324 {

```

(continues on next page)

(continued from previous page)

```

325     "id": "b52fa683.7c5fb8",
326     "type": "function",
327     "z": "7d5c6b14.d2af94",
328     "name": "Dust-PM2.5",
329     "func": "var Dust = msg.payload.valueOf();\nvar newMessage = { payload: { \
↪ "result": Dust, \"Datastream\": {\"@iot.id\": 128}} }; \nnewMessage.headers = { \
↪ \"Content-type\" : \"application/json\" }\nreturn newMessage; ",
330     "outputs": 1,
331     "noerr": 0,
332     "x": 450,
333     "y": 480,
334     "wires": [
335         [
336             "c67491fc.f4755"
337         ]
338     ],
339 },
340 {
341     "id": "58ceff1f.8576a",
342     "type": "ttn app",
343     "z": "",
344     "appId": "gis-tum-sensors",
345     "accessKey": "ttn-account-ACCESSKEY_HERE",
346     "discovery": "discovery.thethingsnetwork.org:1900"
347 }
348 ]

```

2.2.6 References

- *Arduino Sketch for Indoor Ambient Monitoring sensor node*
- *Node red flow for Indoor Ambient Monitoring sensor node*
- [Wiki guide for Seeeduino LoRaWAN board](#)
- [Adding Sced boards to Arduino IDE](#)

2.3 Feather M0 LoRa in TFA Housing

This sensor node is made to showcase a use-case of LoRaWAN technology for outdoor weather monitoring. For achieving this a Feather M0 LoRa module was used with temperature and pressure sensor. The entire setup was carefully placed in the [TFA Housing](#) which is an all-weather protective cover for outdoor transmitters. In this example we measure parameters such as temperature, humidity, altitude, and air pressure.

2.3.1 Hardware

To build this sensor node we have used following hardware components:

- [Adafruit Feather M0 LoRA board](#)
- [Grove - DHT-22 Temperature & Humidity Sensor](#)
- [Grove - Barometric Pressure Sensor](#)



Fig. 7: Sensor node in TFA Housing.

- Breadboard
- TFA Protective Cover
- 6600 mAH Battery

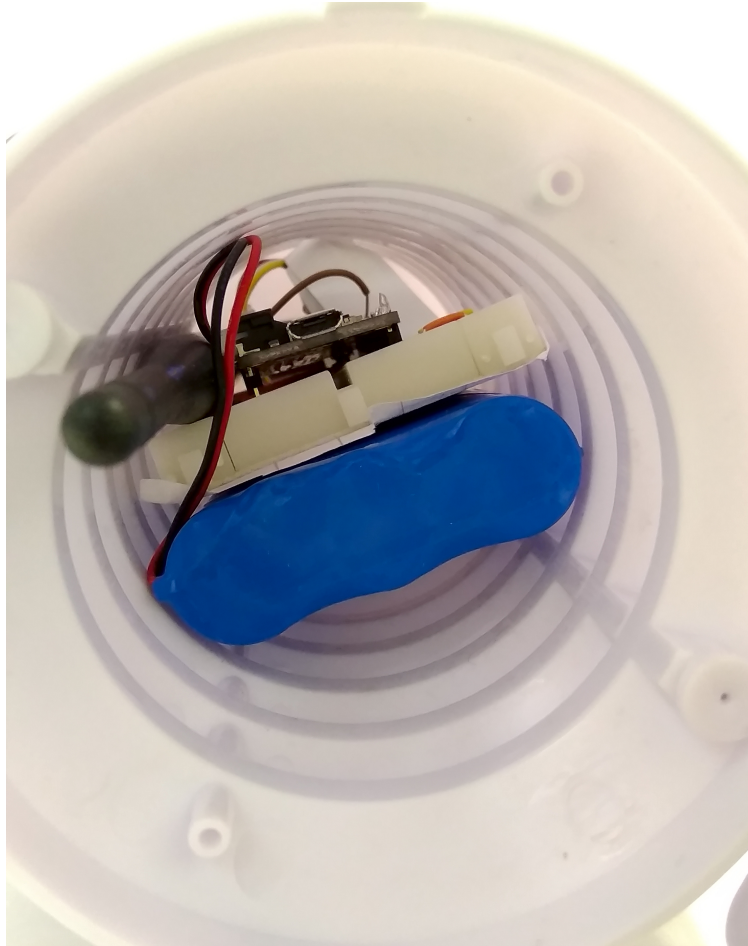


Fig. 8: Inside view of Sensor node in TFA Housing

Also, as the final hardware setup with antenna couldn't completely fit into the casing, a small hole was made at the bottom of the casing to allow the remaining portion of antenna to stay outside.

2.3.2 Wiring setup

First of all, the Feather M0 LoRa board was prepared by soldering the board with the provided grid of pins. Then the board is connected with the sensors using a breadboard. The sensor connections were made using the connector cables as following:

DHT-22 Sensor connections:

- Feather 3V to DHT22 pin 1
- Feather GND to DHT22 pin 4
- Feather pin 12 to DHT22 pin 2



Fig. 9: Bottom view of Sensor node in TFA Housing

- Resistor between DHT pin 1 and DHT pin 2

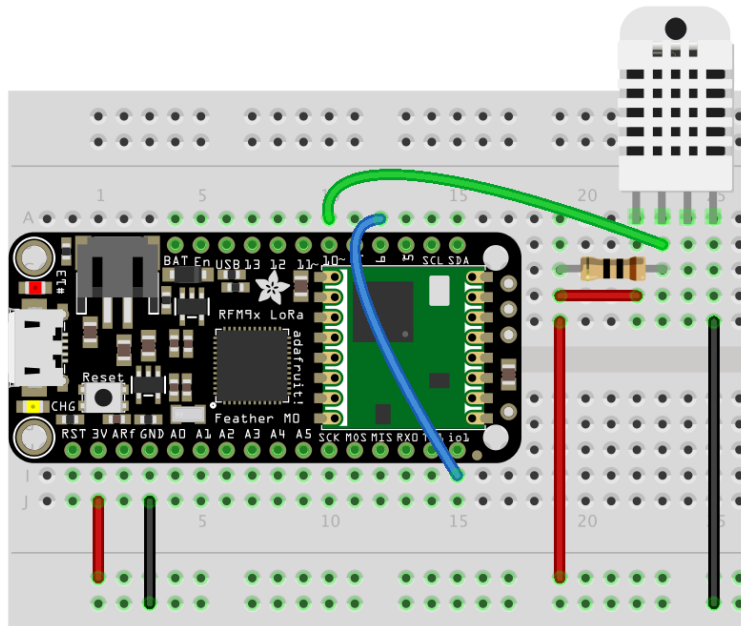


Fig. 10: Wiring with DHT-22 Sensor

Grove-Barometer Sensor connections:

- Feather SCL to Barometer Sensor pin 1 (yellow)
- Feather SDA to Barometer Sensor pin 2 (white)
- Feather 3V to Barometer Sensor pin 3 (red)
- Feather GND to Barometer Sensor pin 4 (black)

Apart from this, Feather pin 6 should be permanently wired with Feather pin io1 as shown in the figure above.

To ensure the durable connections, smaller jumper wires were used on the breadboard instead of longer connecting cables. Sensors and cables were also supported with an insulating duct tape.

Final hardware setup looked as following:

Once all these connection were made, the board is connected with a computer using a USB cable. Further, steps of software part needs to be followed.

2.3.3 Software

To create this node, we use Arduino IDE for setting up the Feather M0 LoRa module. First, install the [Feather M0 LoRa](#) board to your Arduino IDE and select the correct port. Then following libraries needs to be installed before compiling the code:

- [lmic.h](#) for implementing LoRaWAN on Arduino hardware.
- [hal/hal.h](#) bundled with lmic library.
- [Adafruit_SleepyDog.h](#) for controlling low power sleep mode.
- [Wire.h](#) to communicate with I2C devices.

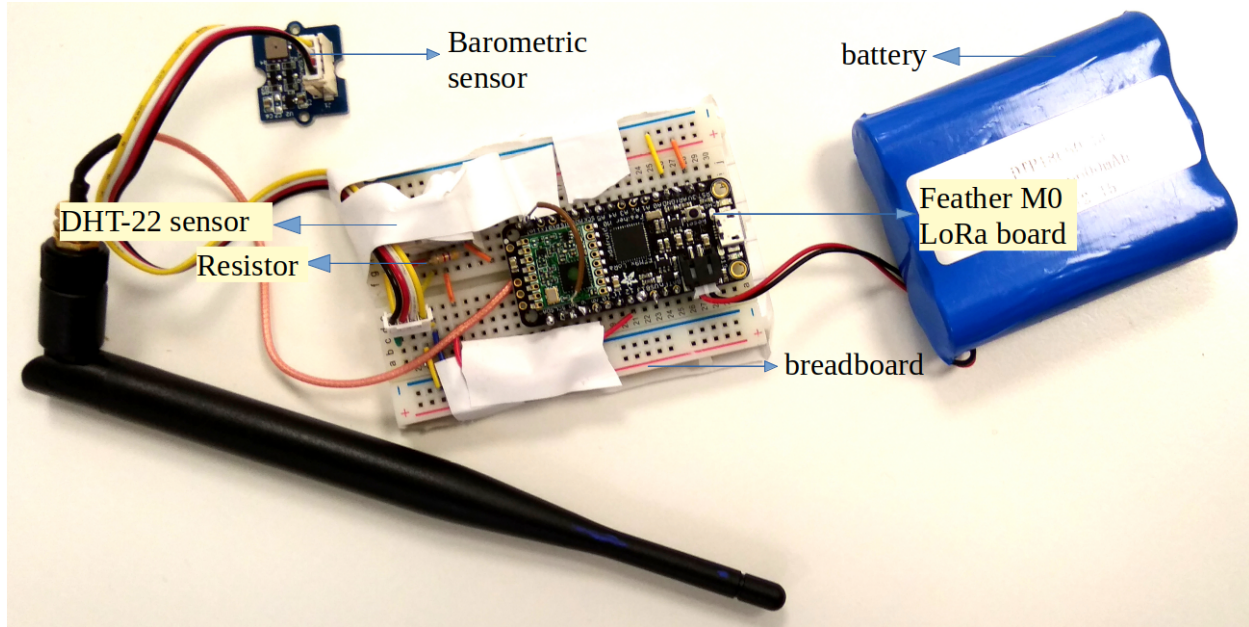


Fig. 11: Final hardware wiring

- [BMP085.h](#) for Barometer sensor.
- [DHT.h](#) for reading DHT-22 sensor.
- [CayenneLPP.h](#) for Cayenne Protocol.

Apart from this, `SPI.h` library is also used for communicating with serial peripheral interface but it is already inbuilt in Arduino IDE and is not required to be separately installed.

Now download and run the *Arduino Sketch for Outdoor Weather Monitoring sensor node* file in the Arduino IDE. This code was created by merging the example code of both the sensors and the `ttn-otaa` example from the `Imic` library. Some required changes were made while merging the example codes. The user should change the network session key, app session key and device address in the code before compiling. These keys can be obtained from the TTN, SWM or other service providers.

Listing 9: Modify the keys in highlighted lines.

```

1 // LoRaWAN NwksKey, network session key
2 // This should be in big-endian (aka msb).
3 static const PROGMEM u1_t NWKSKEY[16] = {NETWORK_SESSION_KEY_HERE_IN_MSB_FORMAT};
4
5 // LoRaWAN AppSKey, application session key
6 // This should also be in big-endian (aka msb).
7 static const u1_t PROGMEM APPSKEY[16] = {APPLICATION_SESSION_KEY_HERE_IN_MSB_FORMAT};
8
9 // LoRaWAN end-device address (DevAddr)
10 // See http://thethingsnetwork.org/wiki/AddressSpace
11 // The library converts the address to network byte order as needed, so this should_
12 // be in big-endian (aka msb) too.
13 static const u4_t DEVADDR = 0x260XXXXX ; // <-- Change this address for every node!

```

The pin mapping configured in the code should also be verified for the board that is being used. Current pin mapping is set as per the Feather M0 LoRa board.

Listing 10: Set the correct pin mapping for the board that is used.

```

1 // Pin mapping
2 const lmic_pinmap lmic_pins = {
3     .nss = 8,
4     .rxtx = LMIC_UNUSED_PIN,
5     .rst = 4,
6     .dio = {3, 6, LMIC_UNUSED_PIN},

```

Following is the example code that can be used to measure the battery voltage of the Feather M0 LoRa board:

Listing 11: Code for measuring the battery voltage

```

1     measuredvbat = analogRead(VBATPIN);
2     measuredvbat *= 2;    // we divided by 2, so multiply back
3     measuredvbat *= 3.3;  // Multiply by 3.3V, our reference voltage
4     measuredvbat /= 1024; // convert to voltage
5
6     SERIALDEBUG_PRINT(" %t");
7     SERIALDEBUG_PRINT("Battery Voltage: ");

```

2.3.4 Services

This node is connected using the TheThingsNetwork service. Further, a node-red work bench is used to forward this collected data from the TTN platform to the OGC Sensor Things API configured on the FROST Server. The node-red workbench that was used for forwarding the data is available at *Node red flow for Outdoor Weather Monitoring sensor node*. To use this node-red-workbench go to the node-red platform <https://iot.gis.bgu.tum.de:1885/>, login with the credentials, go to the options and select Import>Clipboard. Select the downloaded .json file with the given option and click on import. Make necessary changes and deploy the flow.

Datastreams setup for this sensor node on the FROST server can be seen at: [http://iot.gis.bgu.tum.de:8081/FROST-Server-gi3/v1.0/Things\(20\)/Datastreams](http://iot.gis.bgu.tum.de:8081/FROST-Server-gi3/v1.0/Things(20)/Datastreams)

The node-red workbench for this sensor node could be found at: <https://iot.gis.bgu.tum.de:1885/#flow/f6f7a740.c6b338>

The GRAFANA dash-board for visualizing the collected data is available at: <https://iot.gis.bgu.tum.de:3050/d/sMJ3jAAWz/featherm0lora-in-tfa-housing?orgId=1>

2.3.5 Code files

Listing 12: Arduino Sketch for Outdoor Weather Monitoring sensor node

```

1  /*****
2  * Copyright (c) 2015 Thomas Telkamp and Matthijs Kooijman
3  *
4  *
5  *
6  *
7  * Permission is hereby granted, free of charge, to anyone
8  * obtaining a copy of this document and accompanying files,
9  * to do whatever they want with them without any restriction,
10 * including, but not limited to, copying, modification and redistribution.
11 * NO WARRANTY OF ANY KIND IS PROVIDED.

```

(continues on next page)

(continued from previous page)

```

12  *
13  * This example sends a valid LoRaWAN packet with payload "Hello,
14  * world!", using frequency and encryption settings matching those of
15  * the The Things Network.
16  *
17  * This uses ABP (Activation-by-personalisation), where a DevAddr and
18  * Session keys are preconfigured (unlike OTAA, where a DevEUI and
19  * application key is configured, while the DevAddr and session keys are
20  * assigned/generated in the over-the-air-activation procedure).
21  *
22  * Note: LoRaWAN per sub-band duty-cycle limitation is enforced (1% in
23  * g1, 0.1% in g2), but not the TTN fair usage policy (which is probably
24  * violated by this sketch when left running for longer)!
25  *
26  * To use this sketch, first register your application and device with
27  * the things network, to set or generate a DevAddr, NwkSKey and
28  * AppSKey. Each device should have their own unique values for these
29  * fields.
30  *
31  * Do not forget to define the radio type correctly in config.h.
32  *
33  *****/
34  // #define SERIALDEBUG
35
36  #ifndef SERIALDEBUG
37      #define SERIALDEBUG_PRINT(...) Serial.print(__VA_ARGS__)
38      #define SERIALDEBUG_PRINTLN(...) Serial.println(__VA_ARGS__)
39  #else
40      #define SERIALDEBUG_PRINT(...)
41      #define SERIALDEBUG_PRINTLN(...)
42  #endif
43
44
45  #include <lmic.h>
46  #include <hal/hal.h>
47  #include <SPI.h>
48  #include <Adafruit_SleepyDog.h>
49  #include <DHT.h>
50  #include <CayenneLPP.h>
51  #include "BMP085.h"
52  #include <Wire.h>
53
54  CayenneLPP lpp(51);
55
56  #define DHTPIN          12          // Pin which is connected to the DHT sensor.
57  #define DHTTYPE         DHT22      // DHT 22 (AM2302)
58
59  // DHT_Unified dht(DHTPIN, DHTTYPE);
60  DHT dht(DHTPIN, DHTTYPE);
61
62  #define VBATPIN A7
63
64  float temperature2;
65  float pressure;
66  float atm;
67  float altitude;
68  BMP085 myBarometer;

```

(continues on next page)

(continued from previous page)

```

69
70 // LoRaWAN NwkSKey, network session key
71 // This should be in big-endian (aka msb).
72 static const PROGMEM u1_t NWKSKY[16] = {NETWORK_SESSION_KEY_HERE_IN_MSB_FORMAT};
73
74 // LoRaWAN AppSKey, application session key
75 // This should also be in big-endian (aka msb).
76 static const u1_t PROGMEM APPSKY[16] = {APPLICATION_SESSION_KEY_HERE_IN_MSB_FORMAT};
77
78 // LoRaWAN end-device address (DevAddr)
79 // See http://thethingsnetwork.org/wiki/AddressSpace
80 // The library converts the address to network byte order as needed, so this should_
81 // be in big-endian (aka msb) too.
82 static const u4_t DEVADDR = 0x260XXXXX ; // <-- Change this address for every node!
83
84 // These callbacks are only used in over-the-air activation, so they are
85 // left empty here (we cannot leave them out completely unless
86 // DISABLE_JOIN is set in config.h, otherwise the linker will complain).
87 void os_getArtEui (u1_t* buf) { }
88 void os_getDevEui (u1_t* buf) { }
89 void os_getDevKey (u1_t* buf) { }
90
91 static osjob_t sendjob;
92
93 // Schedule TX every this many seconds (might become longer due to duty
94 // cycle limitations).
95 const unsigned TX_INTERVAL = 1; // seconds transmit cycle plus ...
96 const unsigned SLEEP_TIME = 60*9+55; // seconds sleep time plus ...
97 const unsigned MEASURE_TIME = 2; // seconds measuring time should lead to ...
98 // 5 minute(s) total cycle time
99
100 // Pin mapping
101 const lmic_pinmap lmic_pins = {
102     .nss = 8,
103     .rxtx = LMIC_UNUSED_PIN,
104     .rst = 4,
105     .dio = {3, 6, LMIC_UNUSED_PIN},
106 };
107
108 void onEvent (ev_t ev) {
109     // Serial.print(os_getTime());
110     // Serial.print(": ");
111     SERIALDEBUG_PRINT(os_getTime());
112     SERIALDEBUG_PRINT(": ");
113     switch(ev) {
114         case EV_SCAN_TIMEOUT:
115             SERIALDEBUG_PRINTLN(F("EV_SCAN_TIMEOUT"));
116             break;
117         case EV_BEACON_FOUND:
118             SERIALDEBUG_PRINTLN(F("EV_BEACON_FOUND"));
119             break;
120         case EV_BEACON_MISSED:
121             SERIALDEBUG_PRINTLN(F("EV_BEACON_MISSED"));
122             break;
123         case EV_BEACON_TRACKED:
124             SERIALDEBUG_PRINTLN(F("EV_BEACON_TRACKED"));

```

(continues on next page)

(continued from previous page)

```

125         break;
126     case EV_JOINING:
127         SERIALDEBUG_PRINTLN(F("EV_JOINING"));
128         break;
129     case EV_JOINED:
130         SERIALDEBUG_PRINTLN(F("EV_JOINED"));
131         break;
132     case EV_RFU1:
133         SERIALDEBUG_PRINTLN(F("EV_RFU1"));
134         break;
135     case EV_JOIN_FAILED:
136         SERIALDEBUG_PRINTLN(F("EV_JOIN_FAILED"));
137         break;
138     case EV_REJOIN_FAILED:
139         SERIALDEBUG_PRINTLN(F("EV_REJOIN_FAILED"));
140         break;
141     case EV_TXCOMPLETE:
142         digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the_
↪ voltage LOW
143         SERIALDEBUG_PRINTLN(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
144         if (LMIC.txrxFlags & TXRX_ACK)
145             SERIALDEBUG_PRINTLN(F("Received ack"));
146         if (LMIC.dataLen) {
147             SERIALDEBUG_PRINT(F("Received "));
148             SERIALDEBUG_PRINT(LMIC.dataLen);
149             SERIALDEBUG_PRINTLN(F(" bytes of payload"));
150         }
151         // Schedule next transmission
152         os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTERVAL), do_
↪ send);
153
154         SERIALDEBUG_PRINTLN("going to sleep now ... ");
155         // lmic library sleeps automatically after transmission has been completed
156         for(int i= 0; i < SLEEP_TIME / 16; i++) {
157             Watchdog.sleep(16000); // maximum seems to be 16 seconds
158             SERIALDEBUG_PRINT('.');
159         }
160         if (SLEEP_TIME % 16) {
161             Watchdog.sleep((SLEEP_TIME % 16)*1000);
162             SERIALDEBUG_PRINT('*');
163         }
164         SERIALDEBUG_PRINTLN("... woke up again");
165
166         break;
167     case EV_LOST_TSYNC:
168         SERIALDEBUG_PRINTLN(F("EV_LOST_TSYNC"));
169         break;
170     case EV_RESET:
171         SERIALDEBUG_PRINTLN(F("EV_RESET"));
172         break;
173     case EV_RXCOMPLETE:
174         // data received in ping slot
175         SERIALDEBUG_PRINTLN(F("EV_RXCOMPLETE"));
176         break;
177     case EV_LINK_DEAD:
178         SERIALDEBUG_PRINTLN(F("EV_LINK_DEAD"));
179         break;

```

(continues on next page)

(continued from previous page)

```

180     case EV_LINK_ALIVE:
181         SERIALDEBUG_PRINTLN(F("EV_LINK_ALIVE"));
182         break;
183     default:
184         SERIALDEBUG_PRINTLN(F("Unknown event"));
185         break;
186 }
187 }
188
189 void do_send(osjob_t* j){
190     // Check if there is not a current TX/RX job running
191     if (LMIC.opmode & OP_TXRXPEND) {
192         SERIALDEBUG_PRINTLN(F("OP_TXRXPEND, not sending"));
193     } else {
194         // Prepare upstream data transmission at the next possible time.
195
196         float temperature, humidity, measuredvbat;
197         int16_t int16_temperature, int16_humidity, int16_vbat;
198
199         // Start a measurement to update the sensor's internal temperature & humidity
200         SERIALDEBUG_PRINTLN("Start measurement...");
201         temperature = dht.readTemperature();
202         // delay(2000);
203         Watchdog.sleep(2000);
204         // Now read the recently measured temperature (2 secs ago) as Celsius (the
205         temperature = dht.readTemperature();
206         // Read the recently measured humidity (2 secs ago)
207         humidity = dht.readHumidity();
208         SERIALDEBUG_PRINTLN("... finished!");
209
210         // Check if any reads failed and exit early (to try again).
211         if (isnan(humidity) || isnan(temperature)) {
212             SERIALDEBUG_PRINTLN("Failed to read from DHT sensor!");
213             for (int i=0; i<5; i++) {
214                 digitalWrite(LED_BUILTIN, HIGH); // turn the LED on by making the
215                 delay(150);
216                 digitalWrite(LED_BUILTIN, LOW); // turn the LED on by making the
217                 delay(150);
218             }
219             // ok, then wait for another period and try it again
220             os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTERVAL), do_
221             } else {
222                 SERIALDEBUG_PRINT("Humidity: ");
223                 SERIALDEBUG_PRINT(humidity);
224                 SERIALDEBUG_PRINT(" %\t");
225                 SERIALDEBUG_PRINT("Temperature: ");
226                 SERIALDEBUG_PRINT(temperature);
227                 SERIALDEBUG_PRINT(" *C ");
228
229                 measuredvbat = analogRead(VBATPIN);
230                 measuredvbat *= 2; // we divided by 2, so multiply back
231                 measuredvbat *= 3.3; // Multiply by 3.3V, our reference voltage

```

(continues on next page)

(continued from previous page)

```

232     measuredvbat /= 1024; // convert to voltage
233
234     SERIALDEBUG_PRINT(" %\t");
235     SERIALDEBUG_PRINT("Battery Voltage: ");
236     SERIALDEBUG_PRINTLN(measuredvbat);
237
238     temperature2 = myBarometer.bmp085GetTemperature(myBarometer.
↪ bmp085ReadUT()); //Get the temperature, bmp085ReadUT MUST be called first
239     pressure = myBarometer.bmp085GetPressure(myBarometer.bmp085ReadUP()); //
↪ Get the temperature
240
241     /*
242     To specify a more accurate altitude, enter the correct mean sea level
243     pressure level. For example, if the current pressure level is 1019.00 hPa
244     enter 101900 since we include two decimal places in the integer value
245     */
246     altitude = myBarometer.calcAltitude(pressure);
247
248     atm = pressure / 101325;
249
250     lpp.reset();
251     lpp.addTemperature(1, temperature);
252     lpp.addRelativeHumidity(2, humidity);
253     lpp.addAnalogInput(3, measuredvbat);
254     lpp.addTemperature(4, temperature2);
255     lpp.addBarometricPressure(5, pressure/100);
256     lpp.addAnalogInput(6, atm);
257     lpp.addAnalogInput(7, altitude);
258
259     //      LMIC_setTxData2(1, mydata, sizeof(mydata)-1, 0);
260
261     // send the 6 bytes payload to LoRaWAN port 7
262     LMIC_setTxData2(7, lpp.getBuffer(), lpp.getSize(), 0);
263     SERIALDEBUG_PRINTLN(F("Packet queued"));
264     digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on by making the_
↪ voltage HIGH
265     }
266
267     // LMIC_setTxData2(1, mydata, sizeof(mydata)-1, 0);
268     // Serial.println(F("Packet queued"));
269     }
270     // Next TX is scheduled after TX_COMPLETE event.
271 }
272
273 void setup() {
274     delay(5000);
275
276     pinMode(LED_BUILTIN, OUTPUT);
277     digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
278
279     #ifdef SERIALDEBUG
280     Serial.begin(9600);
281     // while (!Serial);
282     #endif
283
284     dht.begin();
285     myBarometer.init();

```

(continues on next page)

(continued from previous page)

```

286 SERIALDEBUG_PRINTLN(F("Starting"));
287
288 #ifdef VCC_ENABLE
289 // For Pinoccio Scout boards
290 pinMode(VCC_ENABLE, OUTPUT);
291 digitalWrite(VCC_ENABLE, HIGH);
292 delay(1000);
293 #endif
294
295 // LMIC init
296 os_init();
297 // Reset the MAC state. Session and pending data transfers will be discarded.
298 LMIC_reset();
299 LMIC_setClockError(MAX_CLOCK_ERROR * 1 / 100);
300
301 // Set static session parameters. Instead of dynamically establishing a session
302 // by joining the network, precomputed session parameters are be provided.
303 #ifdef PROGMEM
304 // On AVR, these values are stored in flash and only copied to RAM
305 // once. Copy them to a temporary buffer here, LMIC_setSession will
306 // copy them into a buffer of its own again.
307 uint8_t appskey[sizeof(APPSKEY)];
308 uint8_t nwkskey[sizeof(NWKSKEY)];
309 memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
310 memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
311 LMIC_setSession(0x1, DEVADDR, nwkskey, appskey);
312 #else
313 // If not running an AVR with PROGMEM, just use the arrays directly
314 LMIC_setSession(0x1, DEVADDR, NWKSKEY, APPSKEY);
315 #endif
316
317 #if defined(CFG_eu868)
318 // Set up the channels used by the Things Network, which corresponds
319 // to the defaults of most gateways. Without this, only three base
320 // channels from the LoRaWAN specification are used, which certainly
321 // works, so it is good for debugging, but can overload those
322 // frequencies, so be sure to configure the full frequency range of
323 // your network here (unless your network autoconfigures them).
324 // Setting up channels should happen after LMIC_setSession, as that
325 // configures the minimal channel set.
326 // NA-US channels 0-71 are configured automatically
327 LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
328 ↪ // g-band
329 LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B), BAND_CENTI);
330 ↪ // g-band
331 LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
332 ↪ // g-band
333 LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
334 ↪ // g-band
335 LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
336 ↪ // g-band
337 LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
338 ↪ // g-band
339 LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
340 ↪ // g-band
341 LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
342 ↪ // g-band

```

(continues on next page)

(continued from previous page)

```

335   LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK, DR_FSK), BAND_MILLI);
↪ // g2-band
336   // TTN defines an additional channel at 869.525Mhz using SF9 for class B
337   // devices' ping slots. LMIC does not have an easy way to define set this
338   // frequency and support for class B is spotty and untested, so this
339   // frequency is not configured here.
340   #elif defined(CFG_us915)
341   // NA-US channels 0-71 are configured automatically
342   // but only one group of 8 should (a subband) should be active
343   // TTN recommends the second sub band, 1 in a zero based count.
344   // https://github.com/TheThingsNetwork/gateway-conf/blob/master/US-global_conf.
↪ json
345   LMIC_selectSubBand(1);
346   #endif
347
348   // Disable link check validation
349   LMIC_setLinkCheckMode(0);
350
351   // TTN uses SF9 for its RX2 window.
352   LMIC.dn2Dr = DR_SF9;
353
354   // Set data rate and transmit power for uplink (note: txpow seems to be ignored_
↪ by the library)
355   LMIC_setDrTxpow(DR_SF7,14);
356
357   // Start job
358   do_send(&sendjob);
359 }
360
361 void loop() {
362   os_runloop_once();
363 }

```

Listing 13: Node red flow for Outdoor Weather Monitoring sensor node

```

1  [
2    {
3      "id": "f6f7a740.c6b338",
4      "type": "tab",
5      "label": "Device 2",
6      "disabled": false,
7      "info": ""
8    },
9    {
10     "id": "fafa9ad3.9659e8",
11     "type": "switch",
12     "z": "f6f7a740.c6b338",
13     "name": "Separate",
14     "property": "key",
15     "propertyType": "msg",
16     "rules": [
17       {
18         "t": "cont",
19         "v": "temperature_1",
20         "vt": "str"
21       },

```

(continues on next page)

(continued from previous page)

```

22     {
23         "t": "cont",
24         "v": "humidity",
25         "vt": "str"
26     },
27     {
28         "t": "cont",
29         "v": "analog_in_3",
30         "vt": "str"
31     },
32     {
33         "t": "cont",
34         "v": "temperature_4",
35         "vt": "str"
36     },
37     {
38         "t": "cont",
39         "v": "barometric",
40         "vt": "str"
41     },
42     {
43         "t": "cont",
44         "v": "analog_in_6",
45         "vt": "str"
46     },
47     {
48         "t": "cont",
49         "v": "analog_in_7",
50         "vt": "str"
51     }
52 ],
53 "checkall": "true",
54 "repair": false,
55 "outputs": 7,
56 "x": 220,
57 "y": 180,
58 "wires": [
59     [
60         "492a1844.49a228"
61     ],
62     [
63         "b5be1839.3121a8"
64     ],
65     [
66         "d7e35050.187eb"
67     ],
68     [
69         "c5363ad1.5d3418"
70     ],
71     [
72         "ee2891fa.0dbbe"
73     ],
74     [
75         "71354cb4.e6af04"
76     ],
77     [
78         "d48c0c97.4eb08"

```

(continues on next page)

(continued from previous page)

```

79         ]
80     ],
81 },
82 {
83     "id": "ccb2fb81.aacd58",
84     "type": "split",
85     "z": "f6f7a740.c6b338",
86     "name": "",
87     "splt": "\\n",
88     "spltType": "str",
89     "arraySplt": 1,
90     "arraySpltType": "len",
91     "stream": false,
92     "addname": "key",
93     "x": 90,
94     "y": 180,
95     "wires": [
96         [
97             "fafe9ad3.9659e8"
98         ]
99     ]
100 },
101 {
102     "id": "657fd8a7.01c5e8",
103     "type": "debug",
104     "z": "f6f7a740.c6b338",
105     "name": "",
106     "active": false,
107     "tosidebar": true,
108     "console": false,
109     "tostatus": false,
110     "complete": "false",
111     "x": 870,
112     "y": 240,
113     "wires": []
114 },
115 {
116     "id": "b5be1839.3121a8",
117     "type": "function",
118     "z": "f6f7a740.c6b338",
119     "name": "Humidity",
120     "func": "var humValue = msg.payload.valueOf();\nvar newMessage = { payload:
↪{ \"result\": humValue, \"Datastream\": {\"@iot.id\": 106} } };\nnewMessage.headers_
↪= {\"Content-type\" : \"application/json\"}\nreturn newMessage;",
121     "outputs": 1,
122     "noerr": 0,
123     "x": 440,
124     "y": 200,
125     "wires": [
126         [
127             "dd5d521b.5c984"
128         ]
129     ]
130 },
131 {
132     "id": "dd5d521b.5c984",
133     "type": "http request",

```

(continues on next page)

(continued from previous page)

```

134     "z": "f6f7a740.c6b338",
135     "name": "POST Observation",
136     "method": "POST",
137     "ret": "obj",
138     "paytoqs": false,
139     "url": "http://iot.gis.bgu.tum.de:8081/FROST-Server-gi3/v1.0/Observations",
140     "tls": "",
141     "proxy": "",
142     "authType": "basic",
143     "x": 690,
144     "y": 240,
145     "wires": [
146       [
147         "657fd8a7.01c5e8"
148       ]
149     ]
150   },
151   {
152     "id": "492a1844.49a228",
153     "type": "function",
154     "z": "f6f7a740.c6b338",
155     "name": "Temperature",
156     "func": "var tempValue = msg.payload.valueOf();\nvar newMessage = { payload:
↪{ \"result\": tempValue, \"Datastream\": {\"@iot.id\": 105}} };\\nnewMessage.
↪headers = {\"Content-type\" : \"application/json\"}\\nreturn newMessage;",
157     "outputs": 1,
158     "noerr": 0,
159     "x": 450,
160     "y": 160,
161     "wires": [
162       [
163         "dd5d521b.5c984"
164       ]
165     ]
166   },
167   {
168     "id": "739d03d0.606a6c",
169     "type": "debug",
170     "z": "f6f7a740.c6b338",
171     "name": "",
172     "active": true,
173     "tosidebar": true,
174     "console": false,
175     "tostatus": false,
176     "complete": "payload",
177     "targetType": "msg",
178     "x": 490,
179     "y": 60,
180     "wires": []
181   },
182   {
183     "id": "cb8ef1e2.a85f6",
184     "type": "ttn uplink",
185     "z": "f6f7a740.c6b338",
186     "name": "TTN Input",
187     "app": "58ceff1f.8576a",
188     "dev_id": "tum-gis-device2",

```

(continues on next page)

(continued from previous page)

```

189     "field": "",
190     "x": 80,
191     "y": 60,
192     "wires": [
193         [
194             "aae507e3.771c18"
195         ]
196     ]
197 },
198 {
199     "id": "aae507e3.771c18",
200     "type": "cayennelp-decoder",
201     "z": "f6f7a740.c6b338",
202     "name": "",
203     "x": 260,
204     "y": 60,
205     "wires": [
206         [
207             "ccb2fb81.aacd58",
208             "739d03d0.606a6c"
209         ]
210     ]
211 },
212 {
213     "id": "d7e35050.187eb",
214     "type": "function",
215     "z": "f6f7a740.c6b338",
216     "name": "Battery Voltage",
217     "func": "var Batteryvolt = msg.payload.valueOf();\nvar newMessage = {
↳ payload: { \"result\": Batteryvolt, \"Datastream\": {\"@iot.id\": 107}} };
↳ \nnewMessage.headers = {\"Content-type\" : \"application/json\"}\nreturn newMessage;
↳ ",
218     "outputs": 1,
219     "noerr": 0,
220     "x": 460,
221     "y": 240,
222     "wires": [
223         [
224             "dd5d521b.5c984"
225         ]
226     ]
227 },
228 {
229     "id": "c5363ad1.5d3418",
230     "type": "function",
231     "z": "f6f7a740.c6b338",
232     "name": "Temperature2",
233     "func": "var tempValue = msg.payload.valueOf();\nvar newMessage = { payload:
↳ { \"result\": tempValue, \"Datastream\": {\"@iot.id\": 108}} }; \nnewMessage.
↳ headers = {\"Content-type\" : \"application/json\"}\nreturn newMessage;",
234     "outputs": 1,
235     "noerr": 0,
236     "x": 460,
237     "y": 280,
238     "wires": [
239         [
240             "dd5d521b.5c984"

```

(continues on next page)

(continued from previous page)

```

241         ]
242     ],
243 },
244 {
245     "id": "ee2891fa.0dbbe",
246     "type": "function",
247     "z": "f6f7a740.c6b338",
248     "name": "Barometric Pressure",
249     "func": "var pressure = msg.payload.valueOf();\nvar newMessage = { payload:
↪{ \"result\": pressure, \"Datastream\": {\"@iot.id\": 109}} }; \nnewMessage.headers
↪= {\"Content-type\" : \"application/json\"}\nreturn newMessage;",
250     "outputs": 1,
251     "noerr": 0,
252     "x": 480,
253     "y": 320,
254     "wires": [
255         [
256             "dd5d521b.5c984"
257         ]
258     ]
259 },
260 {
261     "id": "71354cb4.e6af04",
262     "type": "function",
263     "z": "f6f7a740.c6b338",
264     "name": "Pressure atm",
265     "func": "var atm = msg.payload.valueOf();\nvar newMessage = { payload: { \
↪\"result\": atm, \"Datastream\": {\"@iot.id\": 110}} }; \nnewMessage.headers = { \
↪\"Content-type\" : \"application/json\"}\nreturn newMessage;",
266     "outputs": 1,
267     "noerr": 0,
268     "x": 450,
269     "y": 360,
270     "wires": [
271         [
272             "dd5d521b.5c984"
273         ]
274     ]
275 },
276 {
277     "id": "d48c0c97.4eb08",
278     "type": "function",
279     "z": "f6f7a740.c6b338",
280     "name": "Altitude",
281     "func": "var altitude = msg.payload.valueOf();\nvar newMessage = { payload:
↪{ \"result\": altitude, \"Datastream\": {\"@iot.id\": 111}} }; \nnewMessage.headers
↪= {\"Content-type\" : \"application/json\"}\nreturn newMessage;",
282     "outputs": 1,
283     "noerr": 0,
284     "x": 440,
285     "y": 400,
286     "wires": [
287         [
288             "dd5d521b.5c984"
289         ]
290     ]
291 },

```

(continues on next page)

(continued from previous page)

```

292 {
293   "id": "58ceff1f.8576a",
294   "type": "ttn app",
295   "z": "",
296   "appId": "gis-tum-sensors",
297   "accessKey": "ttn-account-ACCESSKEY_HERE",
298   "discovery": "discovery.thethingsnetwork.org:1900"
299 }
300 ]

```

2.3.6 References

- *Arduino Sketch for Outdoor Weather Monitoring sensor node*
- *Node red flow for Outdoor Weather Monitoring sensor node*
- Feather M0 LoRa Arduino IDE Setup
- Sample Arduino codes of using an Adafruit feather M0 LoRa

2.4 Adafruit 32u4 LoRa

This tutorial is made to showcase the use of Adafruit 32u4 board to create a LoRaWAN enabled sensor node. In the following example, a temperature and humidity sensor was used with the Adafruit 32u4 board.

2.4.1 Hardware

To build this sensor node we have used following hardware components:

- Adafruit Feather 32u4 LoRa module
- Grove - DHT-22 Temperature & Humidity Sensor
- Breadboard
- Battery
- Resistor: 4.7k to 10k Ohm

Microcontroller

The Adafruit Feather 32u4 LoRa module is operated by the 8bit ATmega32u4 microcontroller running at 8MHz. It has 32 KB flash memory (to store the program code) and 2 KB of RAM (to store variables, status information, and buffers). The operating voltage of the board is 3.3V (this is important when attaching sensors and other peripherals; they also must operate on 3.3V). The board offers 20 general purpose digital input/output pins (20 GPIOs) with 10 analog input pins (with 12bit analog digital converters (ADC)), one serial port (programmable Universal Asynchronous Receiver and Transmitter, UART), one I2C port, one SPI port, one USB port. The board comes with an embedded Lithium polymer battery management chip and status indicator led, which allows to directly connect a 3.7V LiPo rechargeable battery that will be automatically recharged when the board is powered over its USB connector. The Adafruit Feather 32u4 LoRa board is available in German shops from around 37 € to 45 €.

The LoRa transmitter and receiver is encapsulated within an RFM95 module from the company HopeRF. This module uses the LoRa chip SX1276 from the company Semtech and is dedicated to the 868 MHz frequency band. The RFM95

module is connected via SPI interface to the microcontroller. Most of the required connections of the LoRa transceiver pins with the microcontroller are already built-in on the Adafruit Feather 32u4 LoRa board. However, Digital Pin 6 of the microcontroller must be connected to DIO1 of the LoRa transceiver module in addition using a simple wire. Since the module only implements the LoRa physical layer, the LoRaWAN protocol stack must be implemented in software on the microcontroller. We are using the Arduino library LMIC for that purpose (see below). The implemented LoRaWAN functionality is compatible with LoRaWAN Class A/C.

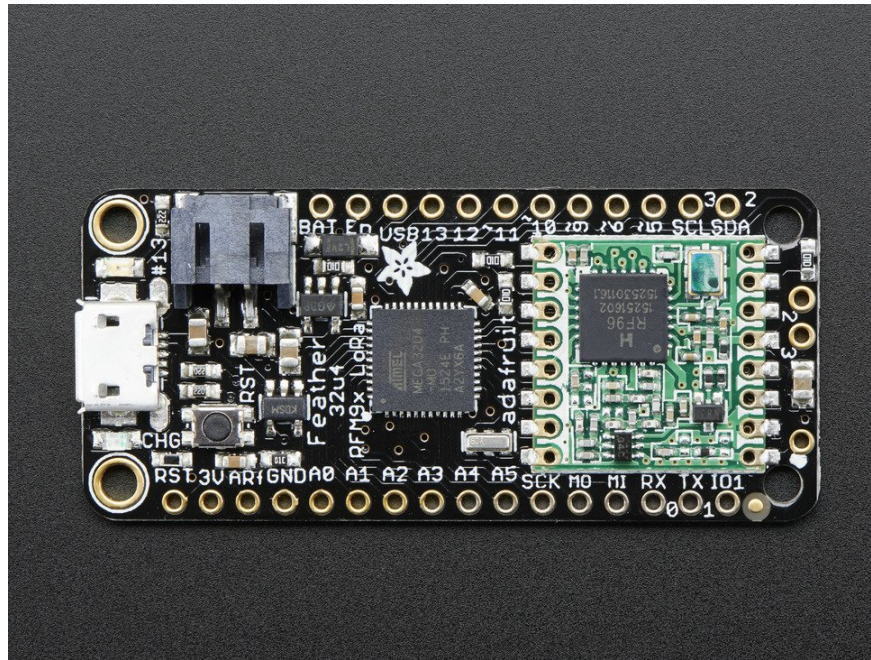


Fig. 12: Feather 32u4 with RFM95 LoRa Radio-868 MHz-RadioFruit from Adafruit. [Feather 32u4 LoRa tutorial](#) with explanations, datasheets, and examples

Sensor

We have attached a DHT22 sensor to the microcontroller board, which measures air temperature and humidity. The minimal time interval between two measurements is 2 seconds. All data transfers between the DHT22 and the microcontroller use a single digital line. The sensor data pin is attached to a GPIO pin (here: Digital Pin 5) of the microcontroller. In addition, a so-called pull-up resistor of 4.7k to 10k Ohm must be connected between the data line and VCC (+3.3V). The [DHT22 datasheet](#) provides more technical details about the DHT22 Sensor. A tutorial on how to use the [DHT22 sensor with Arduino microcontrollers](#) is provided here. The sensor is available in German shops for around 4 € to 10 €.

For more details on the wiring connections, follow [this tutorial](#). Once all these connection are made, the board is connected with a computer using a USB cable. Further, steps of [software part](#) needs to be followed. But, before that we need to [register a new device on the service](#) that we are using.

2.4.2 Software

The sensor node has been programmed using the [Arduino IDE](#). Please note, that in the Arduino framework a program is called a 'Sketch'.

After the sketch has successfully established a connection to The Things Network it reports the air temperature, humidity, and the voltage of a (possibly) attached LiPo battery every 5 minutes. All three values are being encoded in

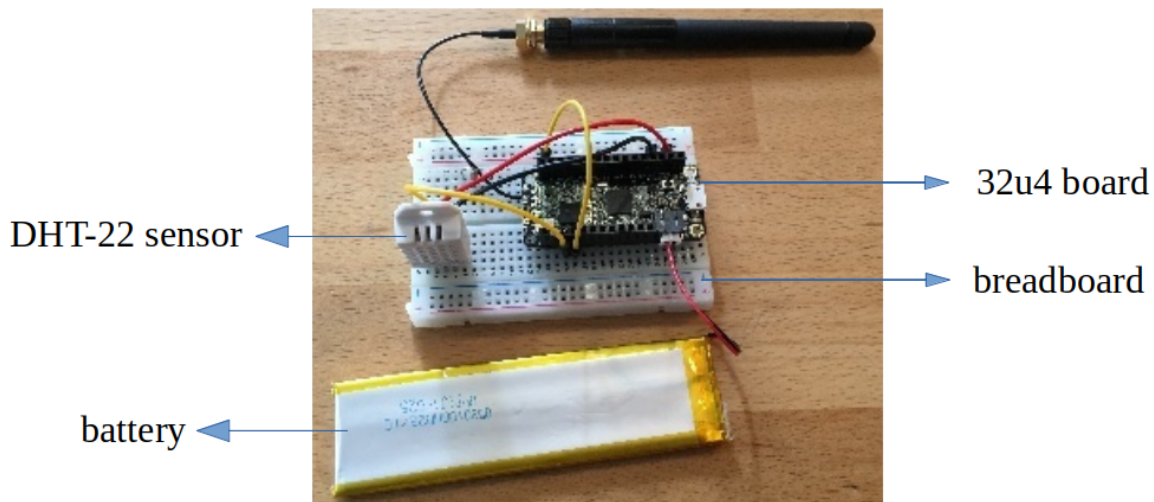


Fig. 13: The Adafruit Feather 32u4 RFM95 LoRa with attached antenna (top), a 1000 mAh lithium polymer (LiPo) battery (bottom), and an attached DHT22 temperature / humidity sensor (white box on the left)

two byte integer values each (in most significant byte order) and then sent as a 6 bytes data packet to the respective TTN application using LoRaWAN port 7. Please note, that LoRaWAN messages can be addressed to ports 1-255 (port 0 is reserved); these ports are similar to port numbers 0-65535 when using the Internet TCP/IP protocol. Voltage and humidity values are always greater or equal to 0, but the temperature value can also become negative. Negative values are represented as a **two's complement**; this must be considered in the Payload Decoding Function used in The Things Network ([see here](#)).

In between two sensor readings the microcontroller is going into deep sleep mode to save battery power. With a 1000 mAh LiPo battery and the current version of the sketch the system can run for at least 5 months. (Further optimizations would be possible, for example, not switching on the LED on the microcontroller board during LoRa data transmissions.)

The employed RFM95 LoRa module does not provide built-in support of the LoRaWAN protocol. Thus, it has to be implemented on the ATmega32u4 microcontroller. We use the **IBM LMIC (LoraMAC-in-C) library** for Arduino. Since the ATmega32u4 microcontroller only has 32 KB of flash memory and the LMIC library is taking most of it, there is only very limited code space left for the application dealing with the sensors (about 2 KB). Nevertheless, this is sufficient to query some sensors like in our example the DHT22.

Now download and run the *Arduino Sketch for Adafruit32u4 LoRa sensor node* file in the Arduino IDE. This code was created by merging the example code of both the sensors and the ttn-otaa example from the Imic library. Some required changes were made while merging the example codes. The user should change the network session key, app session key and device address in the code before compiling. These keys can be obtained from the TTN account as shown in the [services section](#).

Listing 14: Modify the keys in highlighted lines.

```

1 // LoRaWAN NwkSKey, network session key
2 // This should be in big-endian (aka msb).
3 static const PROGMEM u1_t NWKSKEY[16] = {NETWORK_SESSION_KEY_HERE_IN_MSB_FORMAT};
4

```

(continues on next page)

(continued from previous page)

```

5 // LoRaWAN AppSKey, application session key
6 // This should also be in big-endian (aka msb).
7 static const uint_t PROGMEM APPSKEY[16] = {APPLICATION_SESSION_KEY_HERE_IN_MSB_FORMAT};
8
9 // LoRaWAN end-device address (DevAddr)
10 // See http://thethingsnetwork.org/wiki/AddressSpace
11 // The library converts the address to network byte order as needed, so this should
12 // be in big-endian (aka msb) too.
13 static const uint_t DEVADDR = 0x260XXXXX ; // <-- Change this address for every node!

```

Following is the example code that can be used to measure the battery voltage of the sensor node:

Listing 15: Code for measuring the battery voltage

```

1 measuredvbat = analogRead(VBATPIN);
2 measuredvbat *= 2; // we divided by 2, so multiply back
3 measuredvbat *= 3.3; // Multiply by 3.3V, our reference voltage
4 measuredvbat /= 1023; // convert to voltage
5 int16_vbat = round(measuredvbat * 100);
6 mydata[4] = (byte) (int16_vbat >> 8);
7 mydata[5] = (byte) (int16_vbat & 0x00FF);
8 SERIALDEBUG_PRINT(" \t");
9 SERIALDEBUG_PRINT("Battery Voltage: ");
10 SERIALDEBUG_PRINT(measuredvbat);
11 SERIALDEBUG_PRINTLN(" V");

```

2.4.3 Services

The services used for this sensor-node are:

- *TheThingsNetwork* service for LoRaWAN network service.
- *TheThingsNetwork* - *OGC SensorWeb* integration for uploading LoRaWAN sensor data into OGC infrastructure.

Registration of the sensor node with The Things Network (TTN)

The LoRaWAN protocol makes use of a number of different identifiers, addresses, keys, etc. These are required to unambiguously identify devices, applications, as well as to encrypt and decrypt messages. The names and meanings are [nicely explained on a dedicated TTN web page](#).

The sketch given above connects the sensor node with The Things Network (TTN) using the Activation-by-Personalisation (ABP) mode. In this mode, the required keys for data encryption and session management are created manually using the TTN console window and must be pasted into the source code of the sketch below. In order to get this running, you will need to [create a new device in the TTN console window](#). This assumes that you already have a TTN user account (which needs to be created otherwise). In the settings menu of the newly created device the ABP mode must be selected and the settings must be saved. Then copy the DevAddr, the NwkSKey, and the AppSKey from the TTN console web page of the newly registered device and paste them into the proper places in the sketch above. Please make sure that you choose for each of the three keys the correct byte ordering (MSB for all three keys). A detailed explanation of these steps is [given here](#). Then the sketch can be compiled and uploaded to the Adafruit Feather 32u4 LoRa microcontroller.

Important hint: everytime the sensor node is reset or being started again, make sure to reset the frame counter of the registered sensor in the TTN console web page of the registered device. The reason is that in LoRaWAN all transmitted data packets have a frame counter, which is incremented after each data frame being sent. This way a LoRaWAN application can avoid receiving and using the same packet again (replay attack). When TTN receives a

data packet, it checks if the frame number is higher than the last one received before. If not, the received packet is considered to be old or a replay attack and is discarded. When the sensor node is reset or being started again, its frame counter is also reset to 0, hence, the TTN application assumes that all new packages are old, because their frame counter is lower than the last frame received (before the reset). A manual frame counter reset is only necessary when registering the node using ABP mode. In OTAA mode the frame counter is automatically reset in the sensor node and the TTN network server.

TTN Payload Decoding

Everytime a data packet is received by a TTN application a dedicated Javascript function is being called (Payload Decoder Function). This function can be used to decode the received byte string and to create proper Javascript objects or values that can directly be read by humans when looking at the incoming data packet. This is also useful to format the data in a specific way that can then be forwarded to an external application (e.g. a sensor data platform like [MyDevices](#) or [Thingspeak](#)). Such a forwarding can be configured in the TTN console in the “Integrations” tab. *TTN payload decoder for Adafruit32u4 LoRa sensor node* given here checks if a packet was received on LoRaWAN port 7 and then assumes that it consists of the 6 bytes encoded as described above. It creates the three Javascript objects ‘temperature’, ‘humidity’, and ‘vbattery’. Each object has two fields: ‘value’ holds the value and ‘uom’ gives the unit of measure. The source code can simply be copied and pasted into the ‘decoder’ tab in the TTN console after having selected the application. Choose the option ‘Custom’ in the ‘Payload Format’ field. Note that when you also want to handle other sensor nodes sending packets on different LoRaWAN ports, then the Payload Decoder Function can be extended after the end of the `if (port==7) { ... }` statement by adding `else if (port==8) { ... }` `else if (port==9) { ... }` etc.

The Things Network - OGC SensorWeb Integration

The presented Payload Decoder Function works also with the [TTN-OGC SWE Integration](#) for the [52° North Sensor Observation Service \(SOS\)](#). This software component can be downloaded from [this repository](#). It connects a TTN application with a running transactional [Sensor Observation Service 2.0.0 \(SOS\)](#). Data packets received from TTN are imported into the SOS. The SOS persistently stores sensor data from an arbitrary number of sensor nodes and can be queried for the most recent as well as for historic sensor data readings. The 52° North SOS comes with its own REST API and a nice web client allowing to browse the stored sensor data in a convenient way.

We are running an instance of the 52° North SOS and the TTN-OGC SWE Integration. The web client for this LoRaWAN sensor node can be accessed [on this page](#). Here is a screenshot showing the webclient:

Sending a message to the Sensor Node (Downlink)

Using the TTN console we can send a message (i.e. a byte string) to the sensor node. In the [TTN console application page](#) click on the respective application. Then click on the ‘Devices’ tab and choose the proper sensor node (here: [adafruit-feather-32u4-lora3](#)). On the overview page scroll down to the ‘Downlink’ section. In the ‘Payload’ field enter 1 to 4 bytes. In order to show digits or letters on the LED display these must be [ASCII encoded](#) and have to be entered as hexadecimal numbers. When you click on the ‘Send’ button the message will be queued and the next time when the node sends its data packet (uplink) it will receive the message. The first 4 bytes will be shown on the display and the beeper indicates the reception of a new downlink message. In order to blank the display just send a one byte message with the value ‘20’ (hexadecimal for 32, which is the ASCII code for a space). When the node receives just a single blank character it will not produce a beeping sound. There is a nice [web page](#) offering online encoding of text to ASCII numbers in hexadecimal encoding. For example, in order to display the text ‘LoRa’, the four hexadecimal numbers 4C 6F 52 61 have to be entered in the Payload entry field.

2.4.4 Code files



Fig. 14: Web client for data visualization

Listing 16: Arduino Sketch for Adafruit32u4 LoRa sensor node

```

1  /*****
2  * Arduino Sketch for a LoRaWAN sensor node that is registered with
3  * 'The Things Network' (TTN) www.thethingsnetwork.org
4  *
5  * Author: Thomas H. Kolbe, thomas.kolbe@tum.de
6  * Version: 1.0
7  * Last update: 2018-05-21
8  *
9  * The sensor node is based on the Adafruit Feather 32u4 LoRa microcontroller
10 * board https://learn.adafruit.com/adafruit-feather-32u4-radio-with-lora-radio-
11 * module/
12 * The sensor node uses a DHT22 sensor measuring air temperature and humidity.
13 * Also the voltage of an attached LiPo battery is monitored and sent as
14 * an observation. All three values are encoded as 2 byte integer values each.
15 * Hence, the total message payload is 6 bytes. Before the values are converted
16 * to integers they are multiplied by 100 to preserve 2 digits after the decimal
17 * point. Thus, the received values must be divided by 100 to obtain the measured
18 * values. The payload is sent every 300s to LoRaWAN port 7. The following
19 * Javascript function can be used as a payload decoding function in TTN:
20 *
21 * function Decoder(bytes, port) {
22 *   // Decode an uplink message from a buffer
23 *   // (array) of bytes to an object of fields.
24 *   if (port==7) {
25 *     var decoded = {
26 *       "temperature": (bytes[0] << 8 | bytes[1]) / 100.0,

```

(continues on next page)

(continued from previous page)

```

26 *      "humidity": (bytes[2] << 8 | bytes[3]) / 100.0,
27 *      "vbattery": (bytes[4] << 8 | bytes[5]) / 100.0
28 *    };
29 *  } else {
30 *    var decoded = null;
31 *  }
32 *  return decoded;
33 * }
34 *
35 * In between two data transmissions the microcontroller board can go
36 * into sleep mode to reduce energy consumption for extended operation
37 * time when running on battery. Usage of the sleep mode must be
38 * explicitly configured below.
39 *
40 * Important hint: everytime the sensor node is reset or being started again,
41 * make sure to reset the frame counter of the registered sensor in the
42 * TTN console at https://console.thethingsnetwork.org. The reason is that
43 * in LoRaWAN all packets that are transmitted have a frame counter, which
44 * is incremented after each data frame being sent. This way a LoRaWAN application
45 * can avoid receiving and using the same packet again (replay attack). When
46 * TTN receives a data packet, it checks if the frame number is higher than
47 * the last one received before. If not, the received packet is considered
48 * to be old or a replay attack and is discarded. When the sensor node is
49 * reset or being started again, its frame counter is also reset to 0, hence,
50 * the TTN application assumes that all new packages are old, because their
51 * frame counter is lower than the last frame received (before the reset).
52 *
53 * Note, that the DHT22 data pin must be connected to Digital Pin 5 of the
54 * microcontroller board. A resistor of 4.7k - 10k Ohm must be connected to
55 * the data pin and VCC (+3.3V). Digital Pin 6 must be connected to IO1 of the
56 * LoRa transceiver module using a simple wire.
57 *
58 * The code is based on the Open Source library LMIC implementing the LoRaWAN
59 * protocol stack on top of a given LoRa transceiver module (here: RFM95 from
60 * HopeRF, which uses the Semtech SX1276 LoRa chip). The library is originally
61 * being developed by IBM and has been ported to the Arduino platform. See
62 * notes below from the original developers.
63 *
64 * *****
65 * Copyright (c) 2015 Thomas Telkamp and Matthijs Kooijman
66 *
67 * Permission is hereby granted, free of charge, to anyone
68 * obtaining a copy of this document and accompanying files,
69 * to do whatever they want with them without any restriction,
70 * including, but not limited to, copying, modification and redistribution.
71 * NO WARRANTY OF ANY KIND IS PROVIDED.
72 *
73 * This uses ABP (Activation-by-personalisation), where a DevAddr and
74 * Session keys are preconfigured (unlike OTAA, where a DevEUI and
75 * application key is configured, while the DevAddr and session keys are
76 * assigned/generated in the over-the-air-activation procedure).
77 *
78 * Note: LoRaWAN per sub-band duty-cycle limitation is enforced (1% in
79 * g1, 0.1% in g2), but not the TTN fair usage policy (which is probably
80 * violated by this sketch when left running for longer)!
81 *
82 * To use this sketch, first register your application and device with

```

(continues on next page)

(continued from previous page)

```

83  * the things network, to set or generate a DevAddr, NwkSKey and
84  * AppSKey. Each device should have their own unique values for these
85  * fields.
86  *
87  * Do not forget to define the radio type correctly in config.h.
88  *
89  *****/
90
91  // If the following line is uncommented, messages are being printed out to the
92  // serial connection for debugging purposes. When using the Arduino Integrated
93  // Development Environment (Arduino IDE), these messages are displayed in the
94  // Serial Monitor selecting the proper port and a baudrate of 115200.
95
96  //#define SERIALDEBUG
97
98  #ifndef SERIALDEBUG
99      #define SERIALDEBUG_PRINT(...) Serial.print(__VA_ARGS__)
100     #define SERIALDEBUG_PRINTLN(...) Serial.println(__VA_ARGS__)
101 #else
102     #define SERIALDEBUG_PRINT(...)
103     #define SERIALDEBUG_PRINTLN(...)
104 #endif
105
106  // If the following line is uncommented, the sensor node goes into sleep mode
107  // in between two data transmissions. Also the 2secs time between the
108  // initialization of the DHT22 sensor and the reading of the observations
109  // is spent in sleep mode.
110  // Note, that on the Adafruit Feather 32u4 LoRa board the Serial connection
111  // gets lost as soon as the board goes into sleep mode, and it will not be
112  // established again. Thus, the definition of SERIALDEBUG should be commented
113  // out above when using sleep mode.
114
115  #define SLEEPMODE
116
117  #ifndef SLEEPMODE
118      #include <Adafruit_SleepyDog.h>
119  #endif
120
121  #include <lmic.h>
122  #include <hal/hal.h>
123  #include <SPI.h>
124
125  #include <DHT.h>
126
127  #define DHTPIN          5          // Arduino Digital Pin which is connected to the_
128  ↪ DHT sensor.
129  #define DHTTYPE          DHT22    // DHT 22 (AM2302)
130
131  DHT dht (DHTPIN, DHTTYPE);        // create the sensor object
132
133  #define VBATPIN A9              // battery voltage is measured from Analog Input A9
134
135  // The following three constants (NwkSKey, AppSKey, DevAddr) must be changed
136  // for every new sensor node. We are using the LoRaWAN ABP mode (activation by
137  // personalisation) which means that each sensor node must be manually registered
138  // in the TTN console at https://console.thethingsnetwork.org before it can be
139  // started. In the TTN console create a new device and choose ABP mode in the

```

(continues on next page)

(continued from previous page)

```

139 // settings of the newly created device. Then, let TTN generate the NwkSKey and
140 // and the AppSKey and copy them (together with the device address) from the webpage
141 // and paste them below.
142
143 // LoRaWAN NwkSKey, network session key
144 // This should be in big-endian (aka msb).
145 static const PROGMEM ul_t NWKSKY[16] = {NETWORK_SESSION_KEY_HERE_IN_MSB_FORMAT};
146
147 // LoRaWAN AppSKey, application session key
148 // This should also be in big-endian (aka msb).
149 static const ul_t PROGMEM APPSKY[16] = {APPLICATION_SESSION_KEY_HERE_IN_MSB_FORMAT};
150
151 // LoRaWAN end-device address (DevAddr)
152 // See http://thethingsnetwork.org/wiki/AddressSpace
153 // The library converts the address to network byte order as needed, so this should
154 // be in big-endian (aka msb) too.
155 static const u4_t DEVADDR = 0x260XXXXX ; // <-- Change this address for every node!
156
157 // These callbacks are only used in over-the-air activation, so they are
158 // left empty here (we cannot leave them out completely unless
159 // DISABLE_JOIN is set in config.h, otherwise the linker will complain).
160 void os_getArtEui (ul_t* buf) { }
161 void os_getDevEui (ul_t* buf) { }
162 void os_getDevKey (ul_t* buf) { }
163
164 // The following array of bytes is a placeholder to contain the message payload
165 // which is transmitted to the LoRaWAN gateway. We are currently only using 6 bytes.
166 // Please make sure to extend the size of the array, if more sensors should be
167 // attached to the sensor node and the message payload becomes larger than 10 bytes.
168 static uint8_t mydata[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0xA};
169
170 static osjob_t sendjob;
171
172 // Schedule transmission every TX_INTERVAL seconds (might become longer due to duty
173 // cycle limitations). The total interval time is 2secs for the measurement
174 // plus 3secs for the LoRaWAN packet transmission plus TX_INTERVAL_AFTER_SLEEP seconds
175 // plus SLEEP_TIME seconds (microcontroller in sleep mode)
176 const unsigned TX_INTERVAL = 300; // overall cycle time (send one set of
177 // observations every 5mins)
178 const unsigned TX_WAIT_AFTER_SLEEP = 1; // seconds to wait after return from sleep
179 // mode before the next transmit is scheduled
180 const unsigned TX_TIME = 3; // rough estimate of transmission time of a
181 // single packet
182 const unsigned MEASURE_TIME = 2; // seconds measuring time
183 const unsigned SLEEP_TIME = TX_INTERVAL - TX_WAIT_AFTER_SLEEP - TX_TIME - MEASURE_
184 // TIME;
185 const unsigned WAIT_TIME = TX_INTERVAL - TX_TIME - MEASURE_TIME;
186
187 // Pin mapping
188 const lmic_pinmap lmic_pins = {
189     .nss = 8,
190     .rxtx = LMIC_UNUSED_PIN,
191     .rst = 4,
192     .dio = {7, 6, LMIC_UNUSED_PIN},
193 };
194
195 void onEvent (ev_t ev) {

```

(continues on next page)

(continued from previous page)

```

191 SERIALDEBUG_PRINT(os_getTime());
192 SERIALDEBUG_PRINT(": ");
193 switch(ev) {
194     case EV_SCAN_TIMEOUT:
195         SERIALDEBUG_PRINTLN(F("EV_SCAN_TIMEOUT"));
196         break;
197     case EV_BEACON_FOUND:
198         SERIALDEBUG_PRINTLN(F("EV_BEACON_FOUND"));
199         break;
200     case EV_BEACON_MISSED:
201         SERIALDEBUG_PRINTLN(F("EV_BEACON_MISSED"));
202         break;
203     case EV_BEACON_TRACKED:
204         SERIALDEBUG_PRINTLN(F("EV_BEACON_TRACKED"));
205         break;
206     case EV_JOINING:
207         SERIALDEBUG_PRINTLN(F("EV_JOINING"));
208         break;
209     case EV_JOINED:
210         SERIALDEBUG_PRINTLN(F("EV_JOINED"));
211         break;
212     case EV_RFU1:
213         SERIALDEBUG_PRINTLN(F("EV_RFU1"));
214         break;
215     case EV_JOIN_FAILED:
216         SERIALDEBUG_PRINTLN(F("EV_JOIN_FAILED"));
217         break;
218     case EV_REJOIN_FAILED:
219         SERIALDEBUG_PRINTLN(F("EV_REJOIN_FAILED"));
220         break;
221     case EV_TXCOMPLETE:
222         digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the
↳ voltage LOW
223 SERIALDEBUG_PRINTLN(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
224 if (LMIC.txrxFlags & TXRX_ACK)
225     SERIALDEBUG_PRINTLN(F("Received ack"));
226 if (LMIC.dataLen) {
227 #ifdef SERIALDEBUG
228     SERIALDEBUG_PRINT(F("Received "));
229     SERIALDEBUG_PRINT(LMIC.dataLen);
230     SERIALDEBUG_PRINT(F(" bytes of payload: 0x"));
231     for (int i=0; i<LMIC.dataLen; i++) {
232         if (LMIC.frame[LMIC.dataBeg + i] < 0x10) {
233             SERIALDEBUG_PRINT(F("0"));
234         }
235         SERIALDEBUG_PRINT(LMIC.frame[LMIC.dataBeg + i], HEX);
236     }
237     SERIALDEBUG_PRINTLN();
238 #endif
239     // add your code to handle a received downlink data packet here
240 }
241
242 #ifdef SLEEPMODE
243     // Schedule next transmission in 1 second after the board returns from
↳ sleep mode
244     os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_WAIT_AFTER_
↳ SLEEP), do_send);

```

(continues on next page)

(continued from previous page)

```

245     SERIALDEBUG_PRINTLN("going to sleep now ... ");
246     // lmic library sleeps automatically after transmission has been completed
247     for(int i= 0; i < SLEEP_TIME / 8; i++) {
248         Watchdog.sleep(8000); // maximum seems to be 8 seconds
249         SERIALDEBUG_PRINT('.');
250     }
251     if (SLEEP_TIME % 8) {
252         Watchdog.sleep((SLEEP_TIME % 8)*1000);
253         SERIALDEBUG_PRINT('*');
254     }
255     SERIALDEBUG_PRINTLN("... woke up again");
256 #else
257     // Schedule next transmission
258     os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(WAIT_TIME), do_
259     ↪send);
260 #endif
261     break;
262 case EV_LOST_TSYNC:
263     SERIALDEBUG_PRINTLN(F("EV_LOST_TSYNC"));
264     break;
265 case EV_RESET:
266     SERIALDEBUG_PRINTLN(F("EV_RESET"));
267     break;
268 case EV_RXCOMPLETE:
269     // data received in ping slot
270     SERIALDEBUG_PRINTLN(F("EV_RXCOMPLETE"));
271     break;
272 case EV_LINK_DEAD:
273     SERIALDEBUG_PRINTLN(F("EV_LINK_DEAD"));
274     break;
275 case EV_LINK_ALIVE:
276     SERIALDEBUG_PRINTLN(F("EV_LINK_ALIVE"));
277     break;
278 default:
279     SERIALDEBUG_PRINTLN(F("Unknown event"));
280     break;
281 }
282 }
283
284 void do_send(osjob_t* j){
285     // Check if there is not a current TX/RX job running
286     if (LMIC.opmode & OP_TXRXPEND) {
287         SERIALDEBUG_PRINTLN(F("OP_TXRXPEND, not sending"));
288     } else {
289         // Prepare upstream data transmission at the next possible time.
290
291         float temperature, humidity, measuredvbat;
292         int16_t int16_temperature, int16_humidity, int16_vbat;
293
294         // Start a measurement to update the sensor's internal temperature & humidity.
295         ↪reading.
296         // Note, that when fetching measurements from a DHT22 sensor, the reported
297         // values belong to the measurement BEFORE the current measurement.
298         // Therefore, in order to get current observations, we first perform a new
299         ↪measurement
300         // and wait 2 secs (which is the minimum time between two sensor observations.
301         ↪for

```

(continues on next page)

(continued from previous page)

```

299     // the DHT22) and then directly retrieve the observations again.
300     temperature = dht.readTemperature();
301 #ifdef SLEEPMODE
302     Watchdog.sleep(2000);
303 #else
304     delay(2000);
305 #endif
306     // Now read the recently measured temperature (2 secs ago) as Celsius (the_
↪default)
307     temperature = dht.readTemperature();
308     // Read the recently measured humidity (2 secs ago)
309     humidity = dht.readHumidity();
310
311     // Check if any reads failed and exit early (to try again).
312     if (isnan(humidity) || isnan(temperature)) {
313         SERIALDEBUG_PRINTLN("Failed to read from DHT sensor!");
314         // blink the LED five times to indicate that the sensor values could not_
↪be read
315         for (int i=0; i<5; i++) {
316             digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on by making the_
↪voltage HIGH
317             delay(150);
318             digitalWrite(LED_BUILTIN, LOW);    // turn the LED on by making the_
↪voltage HIGH
319             delay(150);
320         }
321         // ok, then wait for another period and try it again
322         os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTERVAL), do_
↪send);
323     } else {
324         SERIALDEBUG_PRINT("Humidity: ");
325         SERIALDEBUG_PRINT(humidity);
326         SERIALDEBUG_PRINT(" %\t");
327         SERIALDEBUG_PRINT("Temperature: ");
328         SERIALDEBUG_PRINT(temperature);
329         SERIALDEBUG_PRINT(" °C ");
330
331         int16_temperature = 100*temperature;
332         int16_humidity = 100*humidity;
333         mydata[0] = (byte) (int16_temperature >> 8);
334         mydata[1] = (byte) (int16_temperature & 0x00FF);
335         mydata[2] = (byte) (int16_humidity >> 8);
336         mydata[3] = (byte) (int16_humidity & 0x00FF);
337
338         measuredvbat = analogRead(VBATPIN);
339         measuredvbat *= 2;    // we divided by 2, so multiply back
340         measuredvbat *= 3.3; // Multiply by 3.3V, our reference voltage
341         measuredvbat /= 1023; // convert to voltage
342         int16_vbat = round(measuredvbat * 100);
343         mydata[4] = (byte) (int16_vbat >> 8);
344         mydata[5] = (byte) (int16_vbat & 0x00FF);
345         SERIALDEBUG_PRINT(" \t");
346         SERIALDEBUG_PRINT("Battery Voltage: ");
347         SERIALDEBUG_PRINT(measuredvbat);
348         SERIALDEBUG_PRINTLN(" V");
349
350         // Send the 6 bytes payload to LoRaWAN port 7 and do not request an_
↪acknowledgement.

```

(continues on next page)

(continued from previous page)

```

351         // The following call does not directly sends the data, but puts a "send_
↪ job"
352         // in the job queue. This job eventually is performed in the call "os_
↪ runloop_once();"
353         // issued repeatedly in the "loop()" method below. After the transmission_
↪ is
354         // complete, the EV_TXCOMPLETE event is signaled, which is handled in the
355         // event handler method "onEvent (ev_t ev)" above. In the EV_TXCOMPLETE_
↪ branch
356         // then a new call to the "do_send(osjob_t* j)" method is being prepared_
↪ for
357         // delayed execution with a waiting time of TX_INTERVAL seconds.
358         LMIC_setTxData2(7, mydata, 6, 0);
359         SERIALDEBUG_PRINTLN(F("Packet queued"));
360         digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on by making the_
↪ voltage HIGH
361
362         // Next TX is scheduled after TX_COMPLETE event.
363     }
364 }
365 }
366
367 void setup() {
368     delay(5000);                // give enough time to open serial monitor (if_
↪ needed)
369
370     pinMode(LED_BUILTIN, OUTPUT);
371     digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
372
373 #ifdef SERIALDEBUG
374     Serial.begin(115200);
375     // while (!Serial);
376 #endif
377
378     dht.begin();                // initialize DHT22 sensor
379
380     SERIALDEBUG_PRINTLN(F("Starting"));
381
382 #ifdef VCC_ENABLE
383     // For Pinoccio Scout boards
384     pinMode(VCC_ENABLE, OUTPUT);
385     digitalWrite(VCC_ENABLE, HIGH);
386     delay(1000);
387 #endif
388
389     // LMIC init
390     os_init();
391     // Reset the MAC state. Session and pending data transfers will be discarded.
392     LMIC_reset();
393     LMIC_setClockError(MAX_CLOCK_ERROR * 1 / 100);
394
395     // Set static session parameters. Instead of dynamically establishing a session
396     // by joining the network, precomputed session parameters are provided.
397 #ifdef PROGMEM
398     // On AVR, these values are stored in flash and only copied to RAM
399     // once. Copy them to a temporary buffer here, LMIC_setSession will
400     // copy them into a buffer of its own again.

```

(continues on next page)

(continued from previous page)

```

401     uint8_t appskey[sizeof(APPSKEY)];
402     uint8_t nwkskey[sizeof(NWKSKEY)];
403     memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
404     memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
405     LMIC_setSession(0x1, DEVADDR, nwkskey, appskey);
406     #else
407     // If not running an AVR with PROGMEM, just use the arrays directly
408     LMIC_setSession(0x1, DEVADDR, NWKSKEY, APPSKEY);
409     #endif
410
411     #if defined(CFG_eu868)
412     // Set up the channels used by the Things Network, which corresponds
413     // to the defaults of most gateways. Without this, only three base
414     // channels from the LoRaWAN specification are used, which certainly
415     // works, so it is good for debugging, but can overload those
416     // frequencies, so be sure to configure the full frequency range of
417     // your network here (unless your network autoconfigures them).
418     // Setting up channels should happen after LMIC_setSession, as that
419     // configures the minimal channel set.
420     // NA-US channels 0-71 are configured automatically
421     LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
422     ↪ // g-band
423     LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B), BAND_CENTI);
424     ↪ // g-band
425     LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
426     ↪ // g-band
427     LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
428     ↪ // g-band
429     LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
430     ↪ // g-band
431     LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
432     ↪ // g-band
433     LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
434     ↪ // g-band
435     LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
436     ↪ // g-band
437     LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK, DR_FSK), BAND_MILLI);
438     ↪ // g2-band
439     // TTN defines an additional channel at 869.525Mhz using SF9 for class B
440     // devices' ping slots. LMIC does not have an easy way to define set this
441     // frequency and support for class B is spotty and untested, so this
442     // frequency is not configured here.
443     #elif defined(CFG_us915)
444     // NA-US channels 0-71 are configured automatically
445     // but only one group of 8 should (a subband) should be active
446     // TTN recommends the second sub band, 1 in a zero based count.
447     // https://github.com/TheThingsNetwork/gateway-conf/blob/master/US-global_conf.
448     ↪ json
449     LMIC_selectSubBand(1);
450     #endif
451
452     // Disable link check validation
453     LMIC_setLinkCheckMode(0);
454
455     // TTN uses SF9 for its RX2 window.
456     LMIC.dn2Dr = DR_SF9;
457

```

(continues on next page)

(continued from previous page)

```

448 // Set data rate and transmit power for uplink (note: txpow seems to be ignored,
↳by the library)
449 LMIC_setDrTxpow(DR_SF7,14);
450
451 // Start job. This will initiate the repetitive sending of data packets,
452 // because after each data transmission, a delayed call to "do_send()"
453 // is being scheduled again.
454 do_send(&sendjob);
455 }
456
457 void loop() {
458     os_runloop_once();
459 }

```

Listing 17: TTN payload decoder for Adafruit32u4 LoRa sensor node

```

1  function Decoder (bytes, port) {
2      var result = {};
3      var transformers = {};
4
5      if (port==7) {
6          transformers = {
7              'temperature': function transform (bytes) {
8                  value=bytes[0]*256 + bytes[1];
9                  if (value>=32768) value=value-65536;
10                 return value/100.0;
11             },
12             'humidity': function transform (bytes) {
13                 return (bytes[0]*256 + bytes[1])/100.0;
14             },
15             'vbattery': function transform (bytes) {
16                 return (bytes[0]*256 + bytes[1])/100.0;
17             },
18         }
19
20         result['temperature'] = {
21             value: transformers['temperature'](bytes.slice(0, 2)),
22             uom: 'Celsius',
23         }
24
25         result['humidity'] = {
26             value: transformers['humidity'](bytes.slice(2, 4)),
27             uom: 'Percent',
28         }
29
30         result['vbattery'] = {
31             value: transformers['vbattery'](bytes.slice(4, 6)),
32             uom: 'Volt',
33         }
34     }
35
36     return result;
37 }

```

2.4.5 References

- [Adafruit Feather 32u4 LoRa microntroller](#)
- [Adafruit Feather 32u4 LoRa tutorial](#)
- [IBM LMIC \(LoraMAC-in-C\) library for Arduino](#)
- [Using Adafruit Feather 32u4 RFM95 as an TTN Node - Stories - Labs](#)
- [TTN LoraWan Atmega32U4 based node – ABP version | Primal Cortex's Weblog](#)
- [node-workshop/lora32u4.md at master · kersing/node-workshop · GitHub](#)
- [Got Adafruit Feather 32u4 LoRa Radio to work and here is how - End Devices \(Nodes\) - The Things Network](#)
- [Adafruit Feather as LoRaWAN node | Wolfgang Klenk](#)
- [LMiC on Adafruit Lora Feather successfully sends message to TTN and then halts with “Packet queued” - End Devices \(Nodes\) - The Things Network](#)
- [GitHub - marcuscbehrens/loralife](#)
- [GPS-Tracker - Stories - Labs](#)

On battery saving / using the deep sleep mode

- [Adafruit Feather 32u4 LoRa - long transmission time after deep sleep - End Devices \(Nodes\) - The Things Network and this](#)
- [Full Arduino Mini LoraWAN and 1.3uA Sleep Mode - End Devices \(Nodes\) - The Things Network](#)
- [Adding Method to Adjust hal_ticks Upon Waking Up from Sleep · Issue #109 · matthijskooijman/arduino-lmic](#)
- [minilora-test/minilora-test.ino at cbe686826bd84fac8381de47b5f5b02dd47c2ca0 · tkerby/minilora-test](#)
- [Arduino-LMIC library with low power mode - Mario Zwiers](#)

2.5 Adafruit 32u4 LoRa with Display

This tutorial is made to showcase the use of Adafruit 32u4 board to create a LoRaWAN enabled sensor node with a display and a case. In the following example, a temperature and humidity sensor was used with the Adafruit 32u4 board to create this tutorial.

2.5.1 Hardware

To build this sensor node we have used following hardware components:

- [Adafruit Feather 32u4 LoRa module](#)
- [Grove - DHT-22 Temperature & Humidity Sensor](#)
- [LED Display](#)
- [Breadboard](#)
- [Battery](#)
- [Resistor: 4.7k to 10k Ohm](#)
- [3d-Printed case](#)

Microcontroller

The Adafruit Feather 32u4 LoRa module is operated by the 8bit ATmega32u4 microcontroller running at 8MHz. It has 32 KB flash memory (to store the program code) and 2 KB of RAM (to store variables, status information, and buffers). The operating voltage of the board is 3.3V (this is important when attaching sensors and other peripherals; they also must operate on 3.3V). The board offers 20 general purpose digital input/output pins (20 GPIOs) with 10 analog input pins (with 12bit analog digital converters (ADC)), one serial port (programmable Universal Asynchronous Receiver and Transmitter, UART), one I2C port, one SPI port, one USB port. The board comes with an embedded Lithium polymer battery management chip and status indicator led, which allows to directly connect a 3.7V LiPo rechargeable battery that will be automatically recharged when the board is powered over its USB connector. The Adafruit Feather 32u4 LoRa board is available in German shops from around 37 € to 45 €.

The LoRa transmitter and receiver is encapsulated within an RFM95 module from the company HopeRF. This module uses the LoRa chip SX1276 from the company Semtech and is dedicated to the 868 MHz frequency band. The RFM95 module is connected via SPI interface to the microcontroller. Most of the required connections of the LoRa transceiver pins with the microcontroller are already built-in on the Adafruit Feather 32u4 LoRa board. However, Digital Pin 6 of the microcontroller must be connected to DIO1 of the LoRa transceiver module in addition using a simple wire. Since the module only implements the LoRa physical layer, the LoRaWAN protocol stack must be implemented in software on the microcontroller. We are using the Arduino library LMIC for that purpose (see below). The implemented LoRaWAN functionality is compatible with LoRaWAN Class A/C.

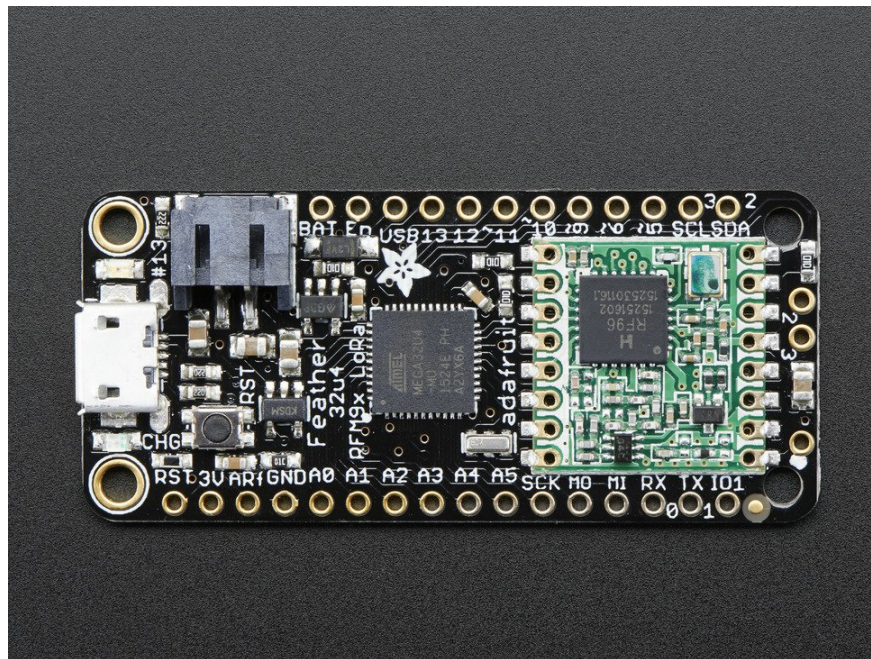


Fig. 15: Feather 32u4 with RFM95 LoRa Radio-868 MHz-RadioFruit from Adafruit. Feather 32u4 LoRa tutorial with explanations, datasheets, and examples.

Sensor

We have attached a DHT22 sensor to the microcontroller board, which measures air temperature and humidity. The minimal time interval between two measurements is 2 seconds. All data transfers between the DHT22 and the microcontroller use a single digital line. The sensor data pin is attached to a GPIO pin (here: Digital Pin 6) of the microcontroller. In addition, a so-called pull-up resistor of 4.7k to 10k Ohm must be connected between the data line and VCC (+3.3V). The [DHT22 datasheet](#) provides more technical details about the DHT22 Sensor. A tutorial on [how](#)

to use the [DHT22 sensor with Arduino](#) microcontrollers is provided here. The sensor is available in German shops for around 4 € to 10 €.

Display / Beeper

On top of the microcontroller board we have attached an [Adafruit Display Wing with a 4 digit 14 segments LED display](#). It can show 0-4 numbers or letters (upper case and lower case). The display controller is using the [I2C protocol](#) and the I2C pins SDA and SCL are directly connected to the Adafruit Feather via the Wing connectors. The Wing is using the default I2C address (0x70). Also a 3.3V beeper is installed that is used to indicate that a new message was received and is now being displayed. The '+' pin of the beeper has to be connected to Digital Pin 12 and the '-' pin to GND. The display and the beeper can be used to notify a user with (very) short messages. The reason why we have included this is mostly to experiment with and to demonstrate the downlink capabilities of LoRaWAN. When a downlink message has been queued it will be transmitted to the node right after it has transmitted the next data packet (uplink data). Hence, it depends on the transmission time period how long it can take unless the node receives and displays a downlink message.

Case

The case was 3D printed using the [design files provided by Adafruit](#). The case consists of three parts. Part 1 is the main enclosure (it does not have a switch holder or tabs, the design file is feather-case.stl). Part 2 is the battery holder (with a slide switch holder, the design file is feather-bat-switch.stl). Part 3 is the case topper (with a cutout for the Adafruit Feather Wing, the design file is feather-top-wing.stl). All design files can be downloaded from [Thingiverse](#).

We have ordered the three parts from an online 3D printing service. The quality of the delivered parts was generally ok, but not good enough for snapping the three parts firmly together. It is not clear yet whether this is a problem of the design files or of the printing service. We used two rubber bands in order to fix the three parts together.

Once all these connections are made, the board is connected with a computer using a USB cable. Further, steps of *software part* need to be followed. But, before that we need to *register a new device on the service* that we are using.

2.5.2 Software

The sensor node has been programmed using the [Arduino IDE](#). Please note, that in the Arduino framework a program is called a 'Sketch'.

After the sketch has successfully established a connection to The Things Network it reports the air temperature, humidity, and the voltage of a (possibly) attached LiPo battery every 5 minutes. All three values are being encoded in two byte integer values each (in most significant byte order) and then sent as a 6 bytes data packet to the respective TTN application using LoRaWAN port 7. Please note, that LoRaWAN messages can be addressed to ports 1-255 (port 0 is reserved); these ports are similar to port numbers 0-65535 when using the Internet TCP/IP protocol. Voltage and humidity values are always greater or equal to 0, but the temperature value can also become negative. Negative values are represented as a *two's complement*; this must be considered in the Payload Decoding Function used in The Things Network (*see here*).

In between two sensor readings the microcontroller is going into deep sleep mode to save battery power. We still have to run some tests to find out for how long the system can run using the 2000 mAh LiPo battery and the current version of the sketch. Showing a received message on the display draws a considerable amount of power and will shorten battery life significantly. Hence, when running on battery it is recommended to clear a displayed message soon by sending a simple space character (0x20). (Further optimizations would be possible, for example, not switching on the LED on the microcontroller board during LoRa data transmissions.)

The employed RFM95 LoRa module does not provide built-in support of the LoRaWAN protocol. Thus, it has to be implemented on the ATmega32u4 microcontroller. We use the [IBM LMIC \(LoraMAC-in-C\) library](#) for Arduino. Since the ATmega32u4 microcontroller only has 32 KB of flash memory and the LMIC library is taking most of it,



2.5. Adafruit 32u4 LoRa with Display

there is only very limited code space left for the application dealing with the sensors (about 2 KB). Nevertheless, this is sufficient to query some sensors like in our example the DHT22.

Now download and run the *Arduino Sketch for Adafruit32u4 LoRa with display sensor node* file in the Arduino IDE. This code was created by merging the example code of both the sensors and the ttn-otaa example from the Imic library. Some required changes were made while merging the example codes. The user should change the network session key, app session key and device address in the code before compiling. These keys can be obtained from the TTN account as shown in the *services section*.

Listing 18: Modify the keys in highlighted lines.

```

1 // This EUI must be in little-endian format, so least-significant-byte
2 // first. When copying an EUI from ttnctl output, this means to reverse
3 // the bytes. For TTN issued EUIs the last bytes should be 0xD5, 0xB3, 0x70.
4 static const uint_t PROGMEM APPEUI[8]={ 0x55, 0xC1, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF }
5 ↪;
6 void os_getArtEui (uint_t* buf) { memcpy_P(buf, APPEUI, 8);}
7
8 // This should also be in little endian format, see above.
9 static const uint_t PROGMEM DEVEUI[8]={ 0xF6, 0xE2, 0x10, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF }
10 ↪;
11 void os_getDevEui (uint_t* buf) { memcpy_P(buf, DEVEUI, 8);}
12
13 // This key should be in big endian format (or, since it is not really a
14 // number but a block of memory, endianness does not really apply). In
15 // practice, a key taken from ttnctl can be copied as-is.
16 // The key shown here is the semtech default key.
17 static const uint_t PROGMEM APPKEY[16] = { 0xC2, 0x21, 0x2E, 0x7A, 0xFF, 0xFF, 0xFF,
18 ↪0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
19 void os_getDevKey (uint_t* buf) { memcpy_P(buf, APPKEY, 16);}

```

Following is the example code that can be used to measure the battery voltage of the sensor node:

Listing 19: Code for measuring the battery voltage

```

1 measuredvbat = analogRead(VBATPIN);
2 measuredvbat *= 2.0; // we divided by 2, so multiply back
3 measuredvbat *= 3.3; // Multiply by 3.3V, our reference voltage
4 measuredvbat /= 1023.0; // convert to voltage
5 int16_vbat = round(measuredvbat * 100.0);
6 mydata[4] = (byte) (int16_vbat >> 8);
7 mydata[5] = (byte) (int16_vbat & 0x00FF);
8 SERIALDEBUG_PRINT(" \t");
9 SERIALDEBUG_PRINT("Battery Voltage: ");
10 SERIALDEBUG_PRINT(measuredvbat);
11 SERIALDEBUG_PRINTLN(" V");

```

2.5.3 Services

The services used for this sensor-node are:

- *TheThingsNetwork* service for LoRaWAN network service.
- *TheThingsNetwork - OGC SensorWeb* integration for uploading LoRaWAN sensor data into OGC infrastructure.

Registration of the sensor node with The Things Network (TTN)

The LoRaWAN protocol makes use of a number of different identifiers, addresses, keys, etc. These are required to unambiguously identify devices, applications, as well as to encrypt and decrypt messages. The names and meanings are [nicely explained on a dedicated TTN web page](#).

The sketch given above connects the sensor node with The Things Network (TTN) using the Over-the-Air-Activation (OTAA) mode. In this mode, we use the three keys AppEUI, DevEUI, AppKey. The DevEUI should be delivered with the sensor node by the manufacturer, the other two keys are created using the TTN console. Each sensor node must be manually registered in the [TTN console](#) before it can be started. This assumes that you already have a TTN user account (which needs to be created otherwise). [In the TTN console create a new device](#) and enter the DevEUI number that was shipped with the Adafruit Feather LoRa board. Note that the shipped number only consists of 6 bytes while LoRaWAN requires an 8 bytes DevEUI. We simply add 0x00 0x00 in the middle of the 6 bytes provided. If you have lost the provided DevEUI you can also let the TTN console create a new one. After the registration of the device the respective keys (AppEUI, DevEUI, AppKey) can be copied from the TTN console and must be pasted into the proper places in the source code of the sketch above. Please make sure that you choose for each of the three keys the correct byte ordering (DevEUI, AppEUI in LSB; AppKey in MSB). A detailed explanation of these steps is [given here](#). Then the sketch can be compiled and uploaded to the Adafruit Feather 32u4 LoRa microcontroller. Note that the three constants (AppEUI, DevEUI, AppKey) must be changed in the source code for every new sensor node.

Using the OTAA mode has the advantage over the ABP (activation by personalization) mode that during connection the session keys are newly created which improves security. Another advantage is that the packet counter is automatically reset to 0 both in the node and in the TTN application.

TTN Payload Decoding

Everytime a data packet is received by a TTN application a dedicated Javascript function is being called (Payload Decoder Function). This function can be used to decode the received byte string and to create proper Javascript objects or values that can directly be read by humans when looking at the incoming data packet. This is also useful to format the data in a specific way that can then be forwarded to an external application (e.g. a sensor data platform like [MyDevices](#) or [Thingspeak](#)). Such a forwarding can be configured in the TTN console in the “Integrations” tab. [TTN payload decoder for Adafruit32u4 LoRa with display sensor node](#) given here checks if a packet was received on LoRaWAN port 7 and then assumes that it consists of the 6 bytes encoded as described above. It creates the three Javascript objects ‘temperature’, ‘humidity’, and ‘vbattery’. Each object has two fields: ‘value’ holds the value and ‘uom’ gives the unit of measure. The source code can simply be copied and pasted into the ‘decoder’ tab in the TTN console after having selected the application. Choose the option ‘Custom’ in the ‘Payload Format’ field. Note that when you also want to handle other sensor nodes sending packets on different LoRaWAN ports, then the Payload Decoder Function can be extended after the end of the if (port==7) { ... } statement by adding else if (port==8) { ... } else if (port==9) { ... } etc.

The Things Network - OGC SensorWeb Integration

The presented Payload Decoder Function works also with the [TTN-OGC SWE Integration](#) for the [52° North Sensor Observation Service \(SOS\)](#). This software component can be downloaded from [this repository](#). It connects a TTN application with a running transactional [Sensor Observation Service 2.0.0 \(SOS\)](#). Data packets received from TTN are imported into the SOS. The SOS persistently stores sensor data from an arbitrary number of sensor nodes and can be queried for the most recent as well as for historic sensor data readings. The 52° North SOS comes with its own REST API and a nice web client allowing to browse the stored sensor data in a convenient way.

We are running an instance of the 52° North SOS and the TTN-OGC SWE Integration. The web client for this LoRaWAN sensor node can be accessed [on this page](#). Here is a screenshot showing the webclient:



Fig. 17: Web client for data visualization

Sending a message to the Sensor Node (Downlink)

Using the TTN console we can send a message (i.e. a byte string) to the sensor node. In the [TTN console application page](#) click on the respective application. Then click on the ‘Devices’ tab and choose the proper sensor node (here: adafruit-feather-32u4-lora3). On the overview page scroll down to the ‘Downlink’ section. In the ‘Payload’ field enter 1 to 4 bytes. In order to show digits or letters on the LED display these must be [ASCII encoded](#) and have to be entered as hexadecimal numbers. When you click on the ‘Send’ button the message will be queued and the next time when the node sends its data packet (uplink) it will receive the message. The first 4 bytes will be shown on the display and the beeper indicates the reception of a new downlink message. In order to blank the display just send a one byte message with the value ‘20’ (hexadecimal for 32, which is the ASCII code for a space). When the node receives just a single blank character it will not produce a beeping sound. There is a nice [web page](#) offering online encoding of text to ASCII numbers in hexadecimal encoding. For example, in order to display the text ‘LoRa’, the four hexadecimal numbers 4C 6F 52 61 have to be entered in the Payload entry field.

2.5.4 Code files

Listing 20: Arduino Sketch for Adafruit32u4 LoRa with display sensor node

```

1  /*****
2  * Arduino Sketch for a LoRaWAN sensor node that is registered with
3  * 'The Things Network' (TTN) www.thethingsnetwork.org
4  *
5  * Author: Thomas H. Kolbe, thomas.kolbe@tum.de
6  * Version: 1.0.0
7  * Last update: 2018-12-09

```

(continues on next page)

(continued from previous page)

```

8  *
9  * The sensor node is based on the Adafruit Feather LoRa microcontroller board
10 * with either the AVR ATmega32u4 or the ATSAM21G18 ARM Cortex M0 microcontroller.
11 * See https://learn.adafruit.com/adafruit-feather-32u4-radio-with-lora-radio-module/
12 * or https://learn.adafruit.com/adafruit-feather-m0-radio-with-lora-radio-module/
13 * The sensor node uses a DHT22 sensor measuring air temperature and humidity.
14 * Also the voltage of an attached LiPo battery is monitored and sent as
15 * an observation. All three values are encoded as 2 byte integer values each.
16 * Hence, the total message payload is 6 bytes. Before the values are converted
17 * to integers they are multiplied by 100 to preserve 2 digits after the decimal
18 * point. Thus, the received values must be divided by 100 to obtain the measured
19 * values. The payload is sent every 300s to LoRaWAN port 7. The following
20 * Javascript function can be used as a payload decoding function in TTN:
21 *
22 * function Decoder(bytes, port) {
23 *   // Decode an uplink message from a buffer
24 *   // (array) of bytes to an object of fields.
25 *   if (port==7) {
26 *     var decoded = {
27 *       "temperature": (bytes[0] << 8 | bytes[1]) / 100.0,
28 *       "humidity": (bytes[2] << 8 | bytes[3]) / 100.0,
29 *       "vbbattery": (bytes[4] << 8 | bytes[5]) / 100.0
30 *     };
31 *   } else {
32 *     var decoded = null;
33 *   }
34 *   return decoded;
35 * }
36 *
37 * In between two data transmissions the microcontroller board can go
38 * into sleep mode to reduce energy consumption for extended operation
39 * time when running on battery. Usage of the sleep mode must be
40 * explicitly configured below.
41 *
42 * Note, that the DHT22 data pin must be connected to Digital Pin 6 of the
43 * microcontroller board (for the Feather 32u4) or Digital Pin 12 (for
44 * the Feather M0). A resistor of 4.7k - 10k Ohm must be connected to
45 * the data pin and VCC (+3.3V).
46 *
47 * Digital Pin 5 (for the Feather 32u4) must be connected to DIO1 of the
48 * LoRa transceiver module using a simple wire.
49 *
50 * For this node we also attach an Adafruit Feather Wing with a four digit
51 * 14-segments LED display. The display controller is using I2C and the
52 * I2C pins SDA and SCL are directly connected to the Adafruit Feather
53 * via the Wing connectors. The wing is using the default I2C address
54 * (0x70). Any LoRaWAN downlink message sent to this node is shown on
55 * the display (only the first 4 characters). We treat each byte of the
56 * received payload as a character in ASCII code. Besides numbers and
57 * letters in upper and lower case also some special characters are
58 * supported. For further details on the Feather Display Wing see here:
59 * https://learn.adafruit.com/14-segment-alpha-numeric-led-featherwing
60 *
61 * In order to notify persons standing nearby that a new text was received
62 * we let the node beep a couple of times. Therefore, Digital Pin 12 (for
63 * the Feather 32u4) should be connected to the '+' port of a 3.3V buzzer
64 * module. The '-' port of the buzzer must be connected to GND.

```

(continues on next page)

(continued from previous page)

```

65  * If a payload containing just a single space (character code 0x20) is
66  * received, the display will be blanked without emitting beeps.
67  *
68  * Note that if the LED display shows some text this will draw a
69  * significant amount of power. This will certainly reduce the operational
70  * duration when running on battery.
71  *
72  * The code is based on the Open Source library LMIC implementing the LoRaWAN
73  * protocol stack on top of a given LoRa transceiver module (here: RFM95 from
74  * HopeRF, which uses the Semtech SX1276 LoRa chip). The library is originally
75  * being developed by IBM and has been ported to the Arduino platform. See
76  * notes below from the original developers.
77  *
78  *****
79  * Copyright (c) 2015 Thomas Telkamp and Matthijs Kooijman
80  *
81  * Permission is hereby granted, free of charge, to anyone
82  * obtaining a copy of this document and accompanying files,
83  * to do whatever they want with them without any restriction,
84  * including, but not limited to, copying, modification and redistribution.
85  * NO WARRANTY OF ANY KIND IS PROVIDED.
86  *
87  * This uses OTAA (Over-the-air activation), where a DevEUI and
88  * application key is configured, which are used in an over-the-air
89  * activation procedure where a DevAddr and session keys are
90  * assigned/generated for use with all further communication.
91  *
92  * Note: LoRaWAN per sub-band duty-cycle limitation is enforced (1% in
93  * g1, 0.1% in g2), but not the TTN fair usage policy (which is probably
94  * violated by this sketch when left running for longer)!
95  *
96  * To use this sketch, first register your application and device with
97  * the things network, to set or generate an AppEUI, DevEUI and AppKey.
98  * Multiple devices can use the same AppEUI, but each device has its own
99  * DevEUI and AppKey.
100  *
101  * Do not forget to define the radio type correctly in config.h.
102  *
103  *****/
104
105  // If the following line is uncommented, messages are being printed out to the
106  // serial connection for debugging purposes. When using the Arduino Integrated
107  // Development Environment (Arduino IDE), these messages are displayed in the
108  // Serial Monitor selecting the proper port and a baudrate of 115200.
109
110  // #define SERIALDEBUG
111
112  #ifndef SERIALDEBUG
113      #define SERIALDEBUG_PRINT(...) Serial.print(__VA_ARGS__)
114      #define SERIALDEBUG_PRINTLN(...) Serial.println(__VA_ARGS__)
115  #else
116      #define SERIALDEBUG_PRINT(...)
117      #define SERIALDEBUG_PRINTLN(...)
118  #endif
119
120  // If the following line is uncommented, the sensor node goes into sleep mode
121  // in between two data transmissions. Also the 2secs time between the

```

(continues on next page)

(continued from previous page)

```

122 // initialization of the DHT22 sensor and the reading of the observations
123 // is spent in sleep mode.
124 // Note, that on the Adafruit Feather 32u4 LoRa board the Serial connection
125 // gets lost as soon as the board goes into sleep mode, and it will not be
126 // established again. Thus, the definition of SERIALDEBUG should be commented
127 // out above when using sleep mode with this board.
128
129 #define SLEEPMODE
130
131 #ifndef SLEEPMODE
132     #include <Adafruit_SleepyDog.h>
133 #endif
134
135 #include <lmic.h>
136 #include <hal/hal.h>
137 #include <SPI.h>
138
139 #include <util/atomic.h>
140 #include <avr/power.h>
141
142 #include <DHT.h>
143
144 #ifdef __AVR_ATmega32U4__
145     #define DHTPIN 6 // Arduino Digital Pin which is connected to the
146     ↪ DHT sensor for Feather 32u4.
147 #endif
148 #ifdef ARDUINO_SAMD_FEATHER_M0
149     #define DHTPIN 12 // Arduino Digital Pin which is connected to the
150     ↪ DHT sensor for Feather M0.
151 #endif
152 #define DHTTYPE DHT22 // DHT 22 (AM2302)
153
154 DHT dht(DHTPIN, DHTTYPE); // create the sensor object
155
156 #ifdef __AVR_ATmega32U4__
157     #define VBATPIN A9 // battery voltage is measured from Analog Input A9
158     ↪ for Feather 32u4
159 #endif
160 #ifdef ARDUINO_SAMD_FEATHER_M0
161     #define VBATPIN A7 // battery voltage is measured from Analog Input A7
162     ↪ for Feather M0
163 #endif
164
165 #ifdef __AVR_ATmega32U4__
166     extern volatile unsigned long timer0_overflow_count;
167 #endif
168
169 #include <Wire.h>
170 #include <Adafruit_GFX.h>
171 #include "Adafruit_LEDBackpack.h"
172
173 Adafruit_AlphaNum4 alpha4 = Adafruit_AlphaNum4();
174
175 #define BUZZERPIN 12 // Arduino Digital Pin which is connected to the
176 ↪ buzzer module

```

(continues on next page)

(continued from previous page)

```

174
175 // The following three constants (AppEUI, DevEUI, AppKey) must be changed
176 // for every new sensor node. We are using the LoRaWAN OTAA mode (over the
177 // air activation). Each sensor node must be manually registered in the
178 // TTN console at https://console.thethingsnetwork.org before it can be
179 // started. In the TTN console create a new device and enter the DevEUI
180 // number that was shipped with the Adafruit Feather LoRa board. Note that
181 // the shipped number only consists of 6 bytes while LoRaWAN requires
182 // an 8 bytes DevEUI. We simply add 0x00 0x00 in the middle of the 6 bytes
183 // provided. If you have lost the provided DevEUI you can also let the
184 // TTN console create a new one. After the registration of the device the
185 // three values can be copied from the TTN console. A detailed explanation
186 // of these steps is given in
187 // https://learn.adafruit.com/the-things-network-for-feather?view=all
188
189 // This EUI must be in little-endian format, so least-significant-byte
190 // first. When copying an EUI from ttnctl output, this means to reverse
191 // the bytes. For TTN issued EUIs the last bytes should be 0xD5, 0xB3, 0x70.
192 static const ul_t PROGMEM APPEUI[8]={ 0x55, 0xC1, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF }
193 ↪;
194 void os_getArtEui (ul_t* buf) { memcpy_P(buf, APPEUI, 8);}
195
196 // This should also be in little endian format, see above.
197 static const ul_t PROGMEM DEVEUI[8]={ 0xF6, 0xE2, 0x10, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF }
198 ↪;
199 void os_getDevEui (ul_t* buf) { memcpy_P(buf, DEVEUI, 8);}
200
201 // This key should be in big endian format (or, since it is not really a
202 // number but a block of memory, endianness does not really apply). In
203 // practice, a key taken from ttnctl can be copied as-is.
204 // The key shown here is the semtech default key.
205 static const ul_t PROGMEM APPKEY[16] = { 0xC2, 0x21, 0x2E, 0x7A, 0xFF, 0xFF, 0xFF,
206 ↪0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
207 void os_getDevKey (ul_t* buf) { memcpy_P(buf, APPKEY, 16);}
208
209 // The following array of bytes is a placeholder to contain the message payload
210 // which is transmitted to the LoRaWAN gateway. We are currently only using 6 bytes.
211 // Please make sure to extend the size of the array, if more sensors should be
212 // attached to the sensor node and the message payload becomes larger than 10 bytes.
213 static uint8_t mydata[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0xA};
214
215 static osjob_t sendjob;
216
217 // Schedule transmission every TX_INTERVAL seconds (might become longer due to duty
218 // cycle limitations). The total interval time is 2secs for the measurement
219 // plus 3secs for the LoRaWAN packet transmission plus SLEEP_TIME seconds
220 // plus SLEEP_TIME seconds (microcontroller in sleep mode)
221 const unsigned int TX_INTERVAL = 300; // overall cycle time (send one set of
222 ↪observations every 5mins)
223 const unsigned int TX_TIME = 22; // rough estimate of transmission time of
224 ↪a single packet
225 const unsigned int MEASURE_TIME = 2; // seconds measuring time
226 const unsigned int SLEEP_TIME = TX_INTERVAL - TX_TIME - MEASURE_TIME;
227 const unsigned int WAIT_TIME = TX_INTERVAL - TX_TIME - MEASURE_TIME;
228
229 // Pin mapping of the LoRa transceiver. Please make sure that DIO1 is connected

```

(continues on next page)

(continued from previous page)

```

226 // to Arduino Digital Pin 6 using an external wire. DIO2 is left unconnected
227 // (it is only required, if FSK modulation instead of LoRa would be used).
228 #ifndef __AVR_ATmega32U4__
229     const lmic_pinmap lmic_pins = {
230         .nss = 8,
231         .rxtx = LMIC_UNUSED_PIN,
232         .rst = 4,
233         .dio = {7, 5, LMIC_UNUSED_PIN},    // in the Feather 32u4 DIO0 is connected to_
        ↪ Arduino Digital Pin 7
234     };
235 #endif
236 #ifdef ARDUINO_SAMD_FEATHER_M0
237     const lmic_pinmap lmic_pins = {
238         .nss = 8,
239         .rxtx = LMIC_UNUSED_PIN,
240         .rst = 4,
241         .dio = {3, 6, LMIC_UNUSED_PIN},    // in the Feather M0 DIO0 is connected to_
        ↪ Arduino Digital Pin 3
242     };
243 #endif
244
245 void onEvent (ev_t ev) {
246     SERIALDEBUG_PRINT(os_getTime());
247     SERIALDEBUG_PRINT(": ");
248     switch(ev) {
249         case EV_SCAN_TIMEOUT:
250             SERIALDEBUG_PRINTLN(F("EV_SCAN_TIMEOUT"));
251             break;
252         case EV_BEACON_FOUND:
253             SERIALDEBUG_PRINTLN(F("EV_BEACON_FOUND"));
254             break;
255         case EV_BEACON_MISSED:
256             SERIALDEBUG_PRINTLN(F("EV_BEACON_MISSED"));
257             break;
258         case EV_BEACON_TRACKED:
259             SERIALDEBUG_PRINTLN(F("EV_BEACON_TRACKED"));
260             break;
261         case EV_JOINING:
262             SERIALDEBUG_PRINTLN(F("EV_JOINING"));
263             break;
264         case EV_JOINED:
265             SERIALDEBUG_PRINTLN(F("EV_JOINED"));
266
267             // Disable link check validation (automatically enabled
268             // during join, but not supported by TTN at this time).
269             LMIC_setLinkCheckMode(0);
270             break;
271         case EV_RFU1:
272             SERIALDEBUG_PRINTLN(F("EV_RFU1"));
273             break;
274         case EV_JOIN_FAILED:
275             SERIALDEBUG_PRINTLN(F("EV_JOIN_FAILED"));
276             break;
277         case EV_REJOIN_FAILED:
278             SERIALDEBUG_PRINTLN(F("EV_REJOIN_FAILED"));
279             break;
280         case EV_TXCOMPLETE:

```

(continues on next page)

(continued from previous page)

```

281     digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the
↳voltage LOW
282     SERIALDEBUG_PRINTLN(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
283     if (LMIC.txrxFlags & TXRX_ACK)
284         SERIALDEBUG_PRINTLN(F("Received ack"));
285     if (LMIC.dataLen) {
286 #ifdef SERIALDEBUG
287         SERIALDEBUG_PRINT(F("Received "));
288         SERIALDEBUG_PRINT(LMIC.dataLen);
289         SERIALDEBUG_PRINT(F(" bytes of payload: 0x"));
290         for (int i=0; i<LMIC.dataLen; i++) {
291             if (LMIC.frame[LMIC.dataBeg + i] < 0x10) {
292                 SERIALDEBUG_PRINT(F("0"));
293             }
294             SERIALDEBUG_PRINT(LMIC.frame[LMIC.dataBeg + i], HEX);
295         }
296         SERIALDEBUG_PRINTLN();
297 #endif
298         // add your code to handle a received downlink data packet here
↳
299         alpha4.clear();
300         for (int i=0; i<LMIC.dataLen && i<4; i++) {
301             alpha4.writeDigitAscii(i, LMIC.frame[LMIC.dataBeg + i]);
302         }
303         alpha4.writeDisplay();
304         if (!(LMIC.frame[LMIC.dataBeg]==' ' && LMIC.dataLen==1))
305             messagebeep();
306     }
307
308 #ifdef SLEEPMODE
309     // Schedule next transmission in 1ms second after the board returns from
↳sleep mode
310     os_setTimedCallback(&sendjob, os_getTime()+ms2osticks(1), do_send);
311
312     SERIALDEBUG_PRINTLN("going to sleep now ... ");
313     // lmic library sleeps automatically after transmission has been completed
314
315     doSleep((uint32_t) SLEEP_TIME*1000);
316 /*
317     int sleepcycles = (int)SLEEP_TIME / 8;
318     int restsleep = (int)SLEEP_TIME % 8;
319     for(int i=0; i < sleepcycles; i++) {
320         Watchdog.sleep(8000); // maximum seems to be 8 seconds
321         SERIALDEBUG_PRINT('.');
322     }
323     if (restsleep) {
324         Watchdog.sleep(restsleep*1000);
325         SERIALDEBUG_PRINT('*');
326     }
327     SERIALDEBUG_PRINTLN("... woke up again");
328
329 #ifdef __AVR_ATmega32U4__
330     // The following statement is required to prevent that LMIC spends another
331     // couple of seconds busy waiting for some RX packets. This is only
↳required
332     // when using SLEEPMODE, because during sleep mode the Arduino timer
↳variables

```

(continues on next page)

(continued from previous page)

```

333 // are not being incremented and LMIC job scheduling is based on this.
334 // timer0_overflow_count += 3E6;
335
336 ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
337     extern volatile unsigned long timer0_millis;
338     extern volatile unsigned long timer0_overflow_count;
339     timer0_millis += SLEEP_TIME*1000;
340     // timer0 uses a /64 prescaler and overflows every 256 timer ticks
341     timer0_overflow_count += microsecondsToClockCycles((uint32_t)SLEEP_
↪ TIME * 1000000) / (64 * 256);
342 }
343 #endif
344 */
345 // We need to reset the duty cycle limits within the LMIC library.
346 // The reason is that in sleep mode the Arduino system timers millis and
↪ micros
347 // do not get incremented. However, LMIC monitors the adherence to the
348 // LoRaWAN duty cycle limitations using the system timers millis and
↪ micros.
349 // Since LMIC does not know that we have slept for a long time and duty
350 // cycle requirements in fact are met, we must reset the respective LMIC
↪ timers
351 // in order to prevent the library to wait for some extra time (which
↪ would
352 // not use sleep mode and, thus, would waste battery energy).
353 LMIC.bands[BAND_MILLI].avail = os_getTime();
354 LMIC.bands[BAND_CENTI].avail = os_getTime();
355 LMIC.bands[BAND_DECI].avail = os_getTime();
356 #else
357 // Schedule next transmission
358 os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(WAIT_TIME), do_
↪ send);
359 #endif
360 break;
361 case EV_LOST_TSYNC:
362     SERIALDEBUG_PRINTLN(F("EV_LOST_TSYNC"));
363     break;
364 case EV_RESET:
365     SERIALDEBUG_PRINTLN(F("EV_RESET"));
366     break;
367 case EV_RXCOMPLETE:
368     // data received in ping slot
369     SERIALDEBUG_PRINTLN(F("EV_RXCOMPLETE"));
370     break;
371 case EV_LINK_DEAD:
372     SERIALDEBUG_PRINTLN(F("EV_LINK_DEAD"));
373     break;
374 case EV_LINK_ALIVE:
375     SERIALDEBUG_PRINTLN(F("EV_LINK_ALIVE"));
376     break;
377 default:
378     SERIALDEBUG_PRINTLN(F("Unknown event"));
379     break;
380 }
381 }
382
383 void do_send(osjob_t* j){

```

(continues on next page)

(continued from previous page)

```

384 // Check if there is not a current TX/RX job running
385 if (LMIC.opmode & OP_TXRXPEND) {
386     SERIALDEBUG_PRINTLN(F("OP_TXRXPEND, not sending"));
387 } else {
388     // Prepare upstream data transmission at the next possible time.
389
390     float temperature, humidity, measuredvbat;
391     int16_t int16_temperature, int16_humidity, int16_vbat;
392
393     // Start a measurement to update the sensor's internal temperature & humidity.
394     ↪reading.
395     // Note, that when fetching measurements from a DHT22 sensor, the reported
396     // values belong to the measurement BEFORE the current measurement.
397     // Therefore, in order to get current observations, we first perform a new
398     ↪measurement
399     // and wait 2 secs (which is the minimum time between two sensor observations.
400     ↪for
401     // the DHT22) and then directly retrieve the observations again.
402
403     temperature = dht.readTemperature();
404 #ifdef SLEEPMODE
405     // Watchdog.sleep(MEASURE_TIME * 1000UL);
406     doSleep(MEASURE_TIME * 1000UL);
407 #else
408     delay(MEASURE_TIME * 1000UL);
409 #endif
410
411     // Now read the recently measured temperature (2 secs ago) as Celsius (the
412     ↪default)
413     temperature = dht.readTemperature();
414
415     // Read the recently measured humidity (2 secs ago)
416     humidity = dht.readHumidity();
417
418     // Check if any reads failed and exit early (to try again).
419     if (isnan(humidity) || isnan(temperature)) {
420         SERIALDEBUG_PRINTLN("Failed to read from DHT sensor!");
421         // blink the LED five times to indicate that the sensor values could not
422         ↪be read
423         for (int i=0; i<5; i++) {
424             digitalWrite(LED_BUILTIN, HIGH); // turn the LED on by making the
425             ↪voltage HIGH
426             delay(150);
427             digitalWrite(LED_BUILTIN, LOW); // turn the LED on by making the
428             ↪voltage HIGH
429             delay(150);
430         }
431         // ok, then wait for another period and try it again
432         os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTERVAL), do_
433         ↪send);
434     } else {
435         SERIALDEBUG_PRINT("Humidity: ");
436         SERIALDEBUG_PRINT(humidity);
437         SERIALDEBUG_PRINT(" %\t");
438         SERIALDEBUG_PRINT("Temperature: ");
439         SERIALDEBUG_PRINT(temperature);
440         SERIALDEBUG_PRINT(" °C ");

```

(continues on next page)

(continued from previous page)

```

433     int16_temperature = round(100.0*temperature);
434     int16_humidity = round(100.0*humidity);
435     mydata[0] = (byte) (int16_temperature >> 8);
436     mydata[1] = (byte) (int16_temperature & 0x00FF);
437     mydata[2] = (byte) (int16_humidity >> 8);
438     mydata[3] = (byte) (int16_humidity & 0x00FF);
439
440
441     measuredvbat = analogRead(VBATPIN);
442     measuredvbat *= 2.0;           // we divided by 2, so multiply back
443     measuredvbat *= 3.3;           // Multiply by 3.3V, our reference voltage
444     measuredvbat /= 1023.0;        // convert to voltage
445     int16_vbat = round(measuredvbat * 100.0);
446     mydata[4] = (byte) (int16_vbat >> 8);
447     mydata[5] = (byte) (int16_vbat & 0x00FF);
448     SERIALDEBUG_PRINT(" \t");
449     SERIALDEBUG_PRINT("Battery Voltage: ");
450     SERIALDEBUG_PRINT(measuredvbat);
451     SERIALDEBUG_PRINTLN(" V");
452
453     // Send the 6 bytes payload to LoRaWAN port 7 and do not request an
↪acknowledgement.
454     // The following call does not directly sends the data, but puts a "send
↪job"
455     // in the job queue. This job eventually is performed in the call "os_
↪runloop_once();"
456     // issued repeatedly in the "loop()" method below. After the transmission
↪is
457     // complete, the EV_TXCOMPLETE event is signaled, which is handled in the
458     // event handler method "onEvent (ev_t ev)" above. In the EV_TXCOMPLETE
↪branch
459     // then a new call to the "do_send(osjob_t* j)" method is being prepared
↪for
460     // delayed execution with a waiting time of TX_INTERVAL seconds.
461     LMIC_setTxData2(7, mydata, 6, 0);
462     SERIALDEBUG_PRINTLN(F("Packet queued"));
463     digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on by making the
↪voltage HIGH
464
465     // Next TX is scheduled after TX_COMPLETE event.
466 }
467 }
468 }
469
470 void doSleep(uint32_t time) {
471     ADCSRA &= ~(1 << ADEN);
472     power_adc_disable();
473
474     while (time > 0) {
475         uint16_t slept;
476         if (time < 8000)
477             slept = Watchdog.sleep(time);
478         else
479             slept = Watchdog.sleep(8000);
480
481         // Update the millis() and micros() counters, so duty cycle
482         // calculations remain correct. This is a hack, fiddling with

```

(continues on next page)

(continued from previous page)

```

483 // Arduino's internal variables, which is needed until
484 // https://github.com/arduino/Arduino/issues/5087 is fixed.
485 ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
486     extern volatile unsigned long timer0_millis;
487     extern volatile unsigned long timer0_overflow_count;
488     timer0_millis += slept;
489     // timer0 uses a /64 prescaler and overflows every 256 timer ticks
490     timer0_overflow_count += microsecondsToClockCycles((uint32_t)slept * 1000) /
↪ (64 * 256);
491 }
492
493 if (slept >= time)
494     break;
495 time -= slept;
496 }
497
498 power_adc_enable();
499 ADCSRA |= (1 << ADEN);
500 }
501
502 void beep(bool longbeep) {
503     digitalWrite(BUZZERPIN, HIGH); // turn the BUZZER off by making the voltage LOW
504     if (longbeep)
505         delay(250);
506     else
507         delay(100);
508     digitalWrite(BUZZERPIN, LOW); // turn the BUZZER off by making the voltage LOW
509     delay(100);
510 }
511
512 void messagebeep() {
513     beep(false);
514     beep(true);
515     beep(false);
516     beep(false);
517     delay(200);
518     beep(false);
519     beep(true);
520     beep(false);
521     beep(false);
522 }
523
524 void setup() {
525     pinMode(LED_BUILTIN, OUTPUT);
526     digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
527
528     pinMode(BUZZERPIN, OUTPUT);
529     digitalWrite(BUZZERPIN, LOW); // turn the BUZZER off by making the voltage LOW
530
531     alpha4.begin(0x70); // pass in the I2C address of the display
532     alpha4.clear();
533     alpha4.writeDisplay();
534     alpha4.writeDigitAscii(0, 'T');
535     alpha4.writeDigitAscii(1, 'e');
536     alpha4.writeDigitAscii(2, 's');
537     alpha4.writeDigitAscii(3, 't');
538     alpha4.writeDisplay();

```

(continues on next page)

(continued from previous page)

```

539     messagebeep();
540
541
542     delay(10000);                // give enough time to open serial monitor (if_
↪needed) or to start uploading of a new sketch
543
544     alpha4.clear();
545     alpha4.writeDisplay();
546
547 #ifndef SERIALDEBUG
548     Serial.begin(115200);
549     // while (!Serial);
550 #endif
551
552     dht.begin();                // initialize DHT22 sensor
553
554     SERIALDEBUG_PRINTLN(F("Starting"));
555
556 #ifdef VCC_ENABLE
557     // For Pinoccio Scout boards
558     pinMode(VCC_ENABLE, OUTPUT);
559     digitalWrite(VCC_ENABLE, HIGH);
560     delay(1000);
561 #endif
562
563     // LMIC init
564     os_init();
565     // Reset the MAC state. Session and pending data transfers will be discarded.
566     LMIC_reset();
567     LMIC_setClockError(MAX_CLOCK_ERROR * 1 / 100);
568
569     #if defined(CFG_eu868)
570     // Set up the channels used by the Things Network, which corresponds
571     // to the defaults of most gateways. Without this, only three base
572     // channels from the LoRaWAN specification are used, which certainly
573     // works, so it is good for debugging, but can overload those
574     // frequencies, so be sure to configure the full frequency range of
575     // your network here (unless your network autoconfigures them).
576     // Setting up channels should happen after LMIC_setSession, as that
577     // configures the minimal channel set.
578     // NA-US channels 0-71 are configured automatically
579     LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
↪ // g-band
580     LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B), BAND_CENTI);
↪ // g-band
581     LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
↪ // g-band
582     LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
↪ // g-band
583     LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
↪ // g-band
584     LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
↪ // g-band
585     LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
↪ // g-band
586     LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
↪ // g-band

```

(continues on next page)

(continued from previous page)

```

587     LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK, DR_FSK), BAND_MILLI);
588     ↪ // g2-band
589     // TTN defines an additional channel at 869.525Mhz using SF9 for class B
590     // devices' ping slots. LMIC does not have an easy way to define set this
591     // frequency and support for class B is spotty and untested, so this
592     // frequency is not configured here.
593     #elif defined(CFG_us915)
594     // NA-US channels 0-71 are configured automatically
595     // but only one group of 8 should (a subband) should be active
596     // TTN recommends the second sub band, 1 in a zero based count.
597     // https://github.com/TheThingsNetwork/gateway-conf/blob/master/US-global_conf.
598     ↪ json
599     LMIC_selectSubBand(1);
600     #endif
601     // Disable link check validation
602     // LMIC_setLinkCheckMode(0);
603     // TTN uses SF9 for its RX2 window.
604     LMIC.dn2Dr = DR_SF9;
605     // Set data rate and transmit power for uplink (note: txpow seems to be ignored_
606     ↪ by the library)
607     LMIC_setDrTxpow(DR_SF9,14);
608     // Start job. This will initiate the repetitive sending of data packets,
609     // because after each data transmission, a delayed call to "do_send()"
610     // is being scheduled again.
611     do_send(&sendjob);
612     // The following settings should further reduce energy consumption. I have not
613     // tested them yet, they are taken from a post in the TTN forum. See
614     // https://www.thethingsnetwork.org/forum/t/adafruit-lora-feather-gateway/2440/50
615     /*
616     power_adc_disable();
617     power_usart0_disable();
618     power_twi_disable();
619     power_timer1_disable();
620     power_timer2_disable();
621     power_timer3_disable();
622     power_usart1_disable();
623     power_usb_disable();
624     USBCON |= (1 << FRZCLK);
625     PLLCSR &= ~(1 << PLLE);
626     USBCON &= ~(1 << USBE );
627     clock_prescale_set(clock_div_2);
628     */
629 }
630
631 void loop() {
632     os_runloop_once();
633 }

```


Listing 21: TTN payload decoder for Adafruit32u4 LoRa with display sensor node

```

1 function Decoder (bytes, port) {
2   var result = {};
3   var transformers = {};
4
5   if (port==7) {
6     transformers = {
7       'temperature': function transform (bytes) {
8         value=bytes[0]*256 + bytes[1];
9         if (value>=32768) value=value-65536;
10        return value/100.0;
11      },
12      'humidity': function transform (bytes) {
13        return (bytes[0]*256 + bytes[1])/100.0;
14      },
15      'vbattery': function transform (bytes) {
16        return (bytes[0]*256 + bytes[1])/100.0;
17      },
18    }
19
20    result['temperature'] = {
21      value: transformers['temperature'](bytes.slice(0, 2)),
22      uom: 'Celsius',
23    }
24
25    result['humidity'] = {
26      value: transformers['humidity'](bytes.slice(2, 4)),
27      uom: 'Percent',
28    }
29
30    result['vbattery'] = {
31      value: transformers['vbattery'](bytes.slice(4, 6)),
32      uom: 'Volt',
33    }
34  }
35
36  return result;
37 }

```

2.5.5 References

- [Adafruit Feather 32u4 LoRa microcontroller](#)
- [Adafruit Feather 32u4 LoRa tutorial](#)
- [IBM LMIC \(LoRaMAC-in-C\) library for Arduino](#)
- [Using Adafruit Feather 32u4 RFM95 as an TTN Node - Stories - Labs](#)
- [TTN LoraWan Atmega32U4 based node – ABP version | Primal Cortex's Weblog](#)
- [node-workshop/lora32u4.md at master · kersing/node-workshop · GitHub](#)
- [Got Adafruit Feather 32u4 LoRa Radio to work and here is how - End Devices \(Nodes\) - The Things Network](#)
- [Adafruit Feather as LoRaWAN node | Wolfgang Klenk](#)

- LMIC on Adafruit Lora Feather successfully sends message to TTN and then halts with “Packet queued” - End Devices (Nodes) - The Things Network
- GitHub - marcusbehrens/loralife
- GPS-Tracker - Stories - Labs

On battery saving / using the deep sleep mode

- Adafruit Feather 32u4 LoRa - long transmission time after deep sleep - End Devices (Nodes) - The Things Network and this
- Full Arduino Mini LoraWAN and 1.3uA Sleep Mode - End Devices (Nodes) - The Things Network
- Adding Method to Adjust hal_ticks Upon Waking Up from Sleep · Issue #109 · matthijskooijman/arduino-lmic
- minilora-test/minilora-test.ino at cbe686826bd84fac8381de47b5f5b02dd47c2ca0 · tkerby/minilora-test
- Arduino-LMIC library with low power mode - Mario Zwiers

2.6 Adafruit M0 LoRa

2.6.1 Hardware

Microcontroller

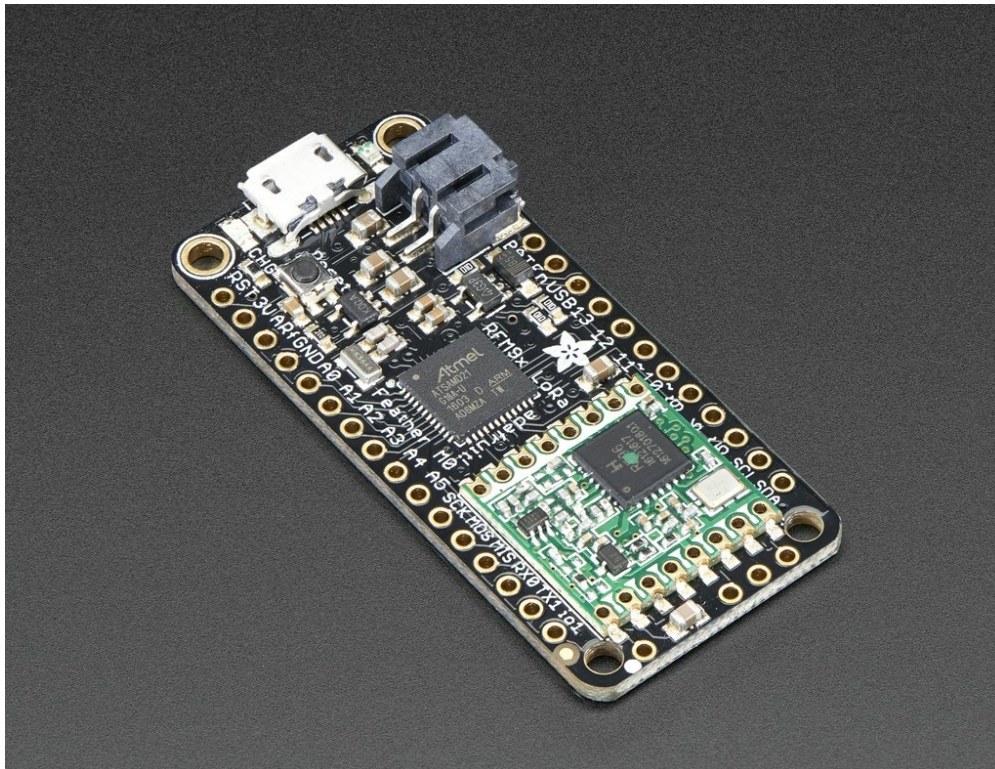


Fig. 18: Feather M0 with RFM95 LoRa Radio - 900 MHz - RadioFruit from Adafruit. Feather M0 LoRa tutorial with explanations, datasheets, and examples.

The Adafruit Feather M0 LoRa board is operated by the 32bit ATSAMD21G18 ARM Cortex M0 microcontroller running at 48MHz. It has 256 KB flash memory (to store the program code) and 32 KB of RAM (to store variables, status information, and buffers). The operating voltage of the board is 3.3V (this is important when attaching sensors and other peripherals; they also must operate on 3.3V). The board offers 20 general purpose digital input/output pins (20 GPIOs) with 10 analog input pins (with 12bit analog digital converters (ADC)), one analog output pin, one serial port (programmable Universal Asynchronous Receiver and Transmitter, UART), one I2C port, one SPI port, one USB port. The board comes with an embedded Lithium polymer battery management chip and status indicator led, which allows to directly connect a 3.7V LiPo rechargeable battery that will be automatically recharged when the board is powered over its USB connector. The Adafruit Feather M0 LoRa board is available in German shops from around 37 € to 45 €.

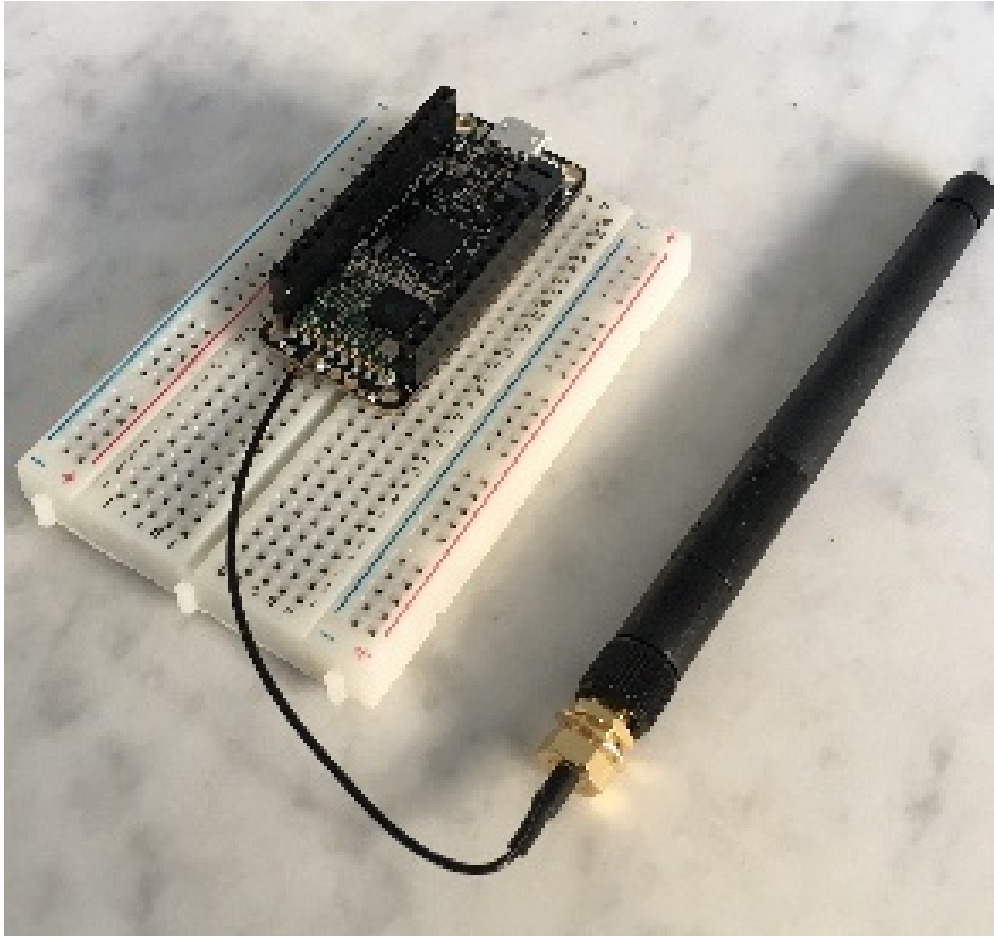


Fig. 19: The Adafruit Feather M0 RFM95 LoRa with attached antenna placed onto a prototyping breadboard. (On this photo the DHT22 sensor and the LiPo battery are missing; we will upload a new photo in the future)

The LoRa transmitter and receiver is encapsulated within an RFM95 module from the company HopeRF. This module uses the LoRa chip SX1276 from the company Semtech and is dedicated to the 868 MHz frequency band. The RFM95 module is connected via SPI interface to the microcontroller. Most of the required connections of the LoRa transceiver pins with the microcontroller are already built-in on the Adafruit Feather M0 LoRa board. However, Digital Pin 6 of the microcontroller must be connected to DIO1 of the LoRa transceiver module in addition using a simple wire. Since the module only implements the LoRa physical layer, the LoRaWAN protocol stack must be implemented in software on the microcontroller. We are using the Arduino library LMIC for that purpose (see below). The implemented LoRaWAN functionality is compatible with LoRaWAN Class A/C.

Sensor

We have attached a DHT22 sensor to the microcontroller board, which measures air temperature and humidity. The minimal time interval between two measurements is 2 seconds. All data transfers between the DHT22 and the microcontroller use a single digital line. The sensor data pin is attached to a GPIO pin (here: Digital Pin 12) of the microcontroller. In addition, a so-called pull-up resistor of 4.7k to 10k Ohm must be connected between the data line and VCC (+3.3V). The [DHT22 datasheet](#) can be accessed here. A tutorial on how to use the DHT22 sensor with Arduino microcontrollers is provided on [this page](#). The sensor is available in German shops for around 4 € to 10 €.

2.6.2 Software

The sensor node has been programmed using the [Arduino IDE](#). Please note, that in the Arduino framework a program is called a ‘Sketch’.

Now download and run the *Arduino Sketch for Adafruit M0 LoRa sensor node* file in the Arduino IDE. After the sketch has successfully established a connection to The Things Network it reports the air temperature, humidity, and the voltage of a (possibly) attached LiPo battery every 5 minutes. All three values are being encoded in two byte integer values each and then sent as a 6 bytes data packet to the respective TTN application using LoRaWAN port 7. Please note, that LoRaWAN messages can be addressed to ports 1-255 (port 0 is reserved); these ports are similar to port numbers 0-65535 when using the Internet TCP/IP protocol. Voltage and humidity values are always greater or equal to 0, but the temperature value can also become negative. Negative values are represented as a two’s complement; this must be considered in the Payload Decoding Function used in The Things Network (see [below](#)).

In between two sensor readings the microcontroller is going into deep sleep mode to save battery power. With a 2000 mAh LiPo battery and the current version of the sketch the system can run for at least 3 months. (Further optimizations would be possible, for example, not switching on the LED on the microcontroller board during LoRa data transmissions.)

The employed RFM95 LoRa module does not provide built-in support of the LoRaWAN protocol. Thus, it has to be implemented on the ARM Cortex M0 microcontroller. We use the IBM LMIC (LoraMAC-in-C) library for Arduino, which can be downloaded from [this repository](#). The ARM Cortex M0 microcontroller has 256 KB of flash memory, which is plenty enough for the LMIC library, the code dealing with the sensors, and even some sophisticated analysis tasks (if required). The source code is given in the following listing:

Note, that the source code is very similar to the source code for the Adafruit Feather 32u4 LoRa board given on the Wiki page [LoRaWAN Node - Adafruit 32u4 LoRa](#). The source code for the Adafruit Feather 32u4 LoRa board has also more detailed comments. It is planned to merge them into a single source code that can be used and compiled for both types of microcontrollers (ATmega32u4 and ARM Cortex M0). The merged source code is already available from [LoRaWAN Node - Adafruit 32u4 LoRa](#), but was not tested with the M0 microcontroller board yet.

Note also, that there is an open issue regarding the deep sleep mode on the ARM Cortex M0 microcontroller in the source code above. During deep sleep mode the (software) timers of the LMIC library are not incremented and after wake-up the library does not recognize that enough time has passed to allow sending another data packet. This built-in mechanism of the LMIC library should ensure that the sensor node does not exceed the maximum duty cycle for LoRaWAN of 1%. This somehow also affects the waiting time for a possible downlink data packet coming from the gateway. As a consequence, the sensor node is not only active for around 2.5 seconds (0.5 seconds to submit the most recent datapacket to the gateway (uplink) and 2 seconds to wait for possible downlink data packets), but sometimes for about 5-6 seconds before it goes back into deep sleep mode (this can be seen from the duration the red LED is activated on the board). These extra seconds awake (with the LED and the LoRa transceiver module switched on) reduce battery lifetime significantly. The ATmega32u4 microcontroller does not have these problems and can go faster back to deep sleep mode. As a result the Adafruit Feather 32u4 LoRa board can run with a 1000 mAh LiPo battery for 5 months and the Adafruit Feather M0 LoRa board with a 2000 mAh LiPo battery for only 3 months.

2.6.3 Services

The services used for this sensor-node are:

- *TheThingsNetwork* service for LoRaWAN network service.
- *TheThingsNetwork - OGC SensorWeb* integration for uploading LoRaWAN sensor data into OGC infrastructure.

Registration of the sensor node with The Things Network (TTN)

The LoRaWAN protocol makes use of a number of different identifiers, addresses, keys, etc. These are required to unambiguously identify devices, applications, as well as to encrypt and decrypt messages. The names and meanings are nicely explained on a [dedicated TTN web page](#).

The sketch given above connects the sensor node with The Things Network (TTN) using the Activation-by-Personalisation (ABP) mode. In this mode, the required keys for data encryption and session management are created manually using the [TTN console](#) window and must be pasted into the source code of the sketch provided in [software section](#). In order to get this running, you will need to create a new device [<https://www.thethingsnetwork.org/docs/devices/registration.html>](https://www.thethingsnetwork.org/docs/devices/registration.html) in the TTN console window. This assumes that you already have a TTN user account (which needs to be created otherwise). In the settings menu of the newly created device the ABP mode must be selected and the settings must be saved. Then copy the DevAddr, the NwkSKey, and the AppSKey from the TTN console web page of the newly registered device and paste them into the proper places in the sketch above. Please make sure that you choose for each of the three keys the correct byte ordering (MSB for all three keys). A detailed explanation of these steps is [given here](#). Then the sketch can be compiled and uploaded to the Adafruit Feather M0 LoRa microcontroller.

Important hint: everytime the sensor node is reset or being started again, make sure to reset the frame counter of the registered sensor in the TTN console web page of the registered device. The reason is that in LoRaWAN all transmitted data packets have a frame counter, which is incremented after each data frame being sent. This way a LoRaWAN application can avoid receiving and using the same packet again (replay attack). When TTN receives a data packet, it checks if the frame number is higher than the last one received before. If not, the received packet is considered to be old or a replay attack and is discarded. When the sensor node is reset or being started again, its frame counter is also reset to 0, hence, the TTN application assumes that all new packages are old, because their frame counter is lower than the last frame received (before the reset). A manual frame counter reset is only necessary when registering the node using ABP mode. In OTAA mode the frame counter is automatically reset in the sensor node and the TTN network server.

TTN Payload Decoding

Everytime a data packet is received by a TTN application a dedicated Javascript function is being called (Payload Decoder Function). This function can be used to decode the received byte string and to create proper Javascript objects or values that can directly be read by humans when looking at the incoming data packet. This is also useful to format the data in a specific way that can then be forwarded to an external application (e.g. a sensor data platform like [MyDevices](#) or [Thingspeak](#)). Such a forwarding can be configured in the TTN console in the “Integrations” tab. [TTN payload decoder for Adafruit M0 LoRa sensor node](#) given here checks if a packet was received on LoRaWAN port 7 and then assumes that it consists of the 6 bytes encoded as described above. It creates the three Javascript objects ‘temperature’, ‘humidity’, and ‘vbattery’. Each object has two fields: ‘value’ holds the value and ‘uom’ gives the unit of measure. The source code can simply be copied and pasted into the ‘decoder’ tab in the TTN console after having selected the application. Choose the option ‘Custom’ in the ‘Payload Format’ field. Note that when you also want to handle other sensor nodes sending packets on different LoRaWAN ports, then the Payload Decoder Function can be extended after the end of the if (port==7) {...} statement by adding else if (port==8) {...} else if (port==9) {...} etc.

The Things Network - OGC SensorWeb Integration

The presented Payload Decoder Function works also with the TTN-OGC SWE Integration for the **52° North Sensor Observation Service (SOS)**. This software component can be downloaded from this [repository](#). It connects a TTN application with a running transactional **Sensor Observation Service 2.0.0 (SOS)**. Data packets received from TTN are imported into the SOS. The SOS persistently stores sensor data from an arbitrary number of sensor nodes and can be queried for the most recent as well as for historic sensor data readings. The 52° North SOS comes with its own REST API and a nice web client allowing to browse the stored sensor data in a convenient way.

We are running an instance of the 52° North SOS and the TTN-OGC SWE Integration. The web client for this LoRaWAN sensor node can be accessed [on this page](#). Here is a screenshot showing the webclient: (Note that the sensor node was wrongly registered with TTN using the name `adafruit-feather-32u4-lora` - it should have been `adafruit-feather-m0-lora`. Hence, while the legend says it is a 32u4 microcontroller in fact it is the M0)



Fig. 20: Web client for data visualization

2.6.4 Code files

Listing 22: Arduino Sketch for Adafruit M0 LoRa sensor node

```

1  /*****
2   * Copyright (c) 2015 Thomas Telkamp and Matthijs Kooijman
3   *
4   * Permission is hereby granted, free of charge, to anyone
5   * obtaining a copy of this document and accompanying files,
6   * to do whatever they want with them without any restriction,
7   * including, but not limited to, copying, modification and redistribution.
8   * NO WARRANTY OF ANY KIND IS PROVIDED.
9   */

```

(continues on next page)

(continued from previous page)

```

10  * This example sends a valid LoRaWAN packet with payload "Hello,
11  * world!", using frequency and encryption settings matching those of
12  * the The Things Network.
13  *
14  * This uses ABP (Activation-by-personalisation), where a DevAddr and
15  * Session keys are preconfigured (unlike OTAA, where a DevEUI and
16  * application key is configured, while the DevAddr and session keys are
17  * assigned/generated in the over-the-air-activation procedure).
18  *
19  * Note: LoRaWAN per sub-band duty-cycle limitation is enforced (1% in
20  * g1, 0.1% in g2), but not the TTN fair usage policy (which is probably
21  * violated by this sketch when left running for longer)!
22  *
23  * To use this sketch, first register your application and device with
24  * the things network, to set or generate a DevAddr, NwkSKey and
25  * AppSKey. Each device should have their own unique values for these
26  * fields.
27  *
28  * Do not forget to define the radio type correctly in config.h.
29  *
30  *****/
31
32 // #define SERIALDEBUG
33
34 #ifndef SERIALDEBUG
35     #define SERIALDEBUG_PRINT(...) Serial.print(__VA_ARGS__)
36     #define SERIALDEBUG_PRINTLN(...) Serial.println(__VA_ARGS__)
37 #else
38     #define SERIALDEBUG_PRINT(...)
39     #define SERIALDEBUG_PRINTLN(...)
40 #endif
41
42
43 #include <lmic.h>
44 #include <hal/hal.h>
45 #include <SPI.h>
46
47 #include <Adafruit_SleepyDog.h>
48
49 // #include <Adafruit_Sensor.h>
50 #include <DHT.h>
51 // #include <DHT_U.h>
52
53 #define DHTPIN          12          // Pin which is connected to the DHT sensor.
54 #define DHTTYPE          DHT22      // DHT 22 (AM2302)
55
56 // DHT_Unified dht(DHTPIN, DHTTYPE);
57 DHT dht(DHTPIN, DHTTYPE);
58
59 #define VBATPIN A7
60
61 // LoRaWAN NwkSKey, network session key
62 // This should be in big-endian (aka msb).
63 static const PROGMEM u1_t NWKSKEY[16] = {NETWORK_SESSION_KEY_HERE_IN_MSB_FORMAT};
64
65 // LoRaWAN AppSKey, application session key
66 // This should also be in big-endian (aka msb).

```

(continues on next page)

(continued from previous page)

```

67 static const ul_t PROGMEM APPSKEY[16] = {APPLICATION_SESSION_KEY_HERE_IN_MSB_FORMAT};
68
69 // LoRaWAN end-device address (DevAddr)
70 // See http://thethingsnetwork.org/wiki/AddressSpace
71 // The library converts the address to network byte order as needed, so this should_
  ↳ be in big-endian (aka msb) too.
72 static const u4_t DEVADDR = 0x260XXXXX ; // <-- Change this address for every node!
73
74 // These callbacks are only used in over-the-air activation, so they are
75 // left empty here (we cannot leave them out completely unless
76 // DISABLE_JOIN is set in config.h, otherwise the linker will complain).
77 void os_getArtEui (ul_t* buf) { }
78 void os_getDevEui (ul_t* buf) { }
79 void os_getDevKey (ul_t* buf) { }
80
81 static uint8_t mydata[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0xA};
82 static osjob_t sendjob;
83
84 // Schedule TX every this many seconds (might become longer due to duty
85 // cycle limitations).
86 const unsigned TX_INTERVAL = 1;           // seconds transmit cycle plus ...
87 const unsigned SLEEP_TIME = 60*4+55;      // seconds sleep time plus ...
88 const unsigned MEASURE_TIME = 2;          // seconds measuring time should lead to ...
89                                           // 5 minute(s) total cycle time
90
91 // Pin mapping
92 const lmic_pinmap lmic_pins = {
93     .nss = 8,
94     .rxtx = LMIC_UNUSED_PIN,
95     .rst = 4,
96     .dio = {3, 6, LMIC_UNUSED_PIN},
97 };
98
99
100 void onEvent (ev_t ev) {
101     // Serial.print(os_getTime());
102     // Serial.print(": ");
103     SERIALDEBUG_PRINT(os_getTime());
104     SERIALDEBUG_PRINT(": ");
105     switch(ev) {
106         case EV_SCAN_TIMEOUT:
107             SERIALDEBUG_PRINTLN(F("EV_SCAN_TIMEOUT"));
108             break;
109         case EV_BEACON_FOUND:
110             SERIALDEBUG_PRINTLN(F("EV_BEACON_FOUND"));
111             break;
112         case EV_BEACON_MISSED:
113             SERIALDEBUG_PRINTLN(F("EV_BEACON_MISSED"));
114             break;
115         case EV_BEACON_TRACKED:
116             SERIALDEBUG_PRINTLN(F("EV_BEACON_TRACKED"));
117             break;
118         case EV_JOINING:
119             SERIALDEBUG_PRINTLN(F("EV_JOINING"));
120             break;
121         case EV_JOINED:
122             SERIALDEBUG_PRINTLN(F("EV_JOINED"));

```

(continues on next page)

(continued from previous page)

```

123         break;
124     case EV_RFU1:
125         SERIALDEBUG_PRINTLN(F("EV_RFU1"));
126         break;
127     case EV_JOIN_FAILED:
128         SERIALDEBUG_PRINTLN(F("EV_JOIN_FAILED"));
129         break;
130     case EV_REJOIN_FAILED:
131         SERIALDEBUG_PRINTLN(F("EV_REJOIN_FAILED"));
132         break;
133     case EV_TXCOMPLETE:
134         digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the_
↪ voltage LOW
135         SERIALDEBUG_PRINTLN(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
136         if (LMIC.txrxFlags & TXRX_ACK)
137             SERIALDEBUG_PRINTLN(F("Received ack"));
138         if (LMIC.dataLen) {
139             SERIALDEBUG_PRINT(F("Received "));
140             SERIALDEBUG_PRINT(LMIC.dataLen);
141             SERIALDEBUG_PRINTLN(F(" bytes of payload"));
142         }
143         // Schedule next transmission
144         os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTERVAL), do_
↪ send);
145
146         SERIALDEBUG_PRINTLN("going to sleep now ... ");
147         // lmic library sleeps automatically after transmission has been completed
148         for(int i= 0; i < SLEEP_TIME / 16; i++) {
149             Watchdog.sleep(16000); // maximum seems to be 16 seconds
150             SERIALDEBUG_PRINT('.');
151         }
152         if (SLEEP_TIME % 16) {
153             Watchdog.sleep((SLEEP_TIME % 16)*1000);
154             SERIALDEBUG_PRINT('*');
155         }
156         SERIALDEBUG_PRINTLN("... woke up again");
157
158         break;
159     case EV_LOST_TSYNC:
160         SERIALDEBUG_PRINTLN(F("EV_LOST_TSYNC"));
161         break;
162     case EV_RESET:
163         SERIALDEBUG_PRINTLN(F("EV_RESET"));
164         break;
165     case EV_RXCOMPLETE:
166         // data received in ping slot
167         SERIALDEBUG_PRINTLN(F("EV_RXCOMPLETE"));
168         break;
169     case EV_LINK_DEAD:
170         SERIALDEBUG_PRINTLN(F("EV_LINK_DEAD"));
171         break;
172     case EV_LINK_ALIVE:
173         SERIALDEBUG_PRINTLN(F("EV_LINK_ALIVE"));
174         break;
175     default:
176         SERIALDEBUG_PRINTLN(F("Unknown event"));
177         break;

```

(continues on next page)

(continued from previous page)

```

178     }
179 }
180
181 void do_send(osjob_t* j){
182     // Check if there is not a current TX/RX job running
183     if (LMIC.opmode & OP_TXRXPEND) {
184         SERIALDEBUG_PRINTLN(F("OP_TXRXPEND, not sending"));
185     } else {
186         // Prepare upstream data transmission at the next possible time.
187
188         float temperature, humidity, measuredvbat;
189         int16_t int16_temperature, int16_humidity, int16_vbat;
190
191         // Start a measurement to update the sensor's internal temperature & humidity
192         SERIALDEBUG_PRINTLN("Start measurement...");
193         temperature = dht.readTemperature();
194         // delay(2000);
195         Watchdog.sleep(2000);
196         // Now read the recently measured temperature (2 secs ago) as Celsius (the
197         temperature = dht.readTemperature();
198         // Read the recently measured humidity (2 secs ago)
199         humidity = dht.readHumidity();
200         SERIALDEBUG_PRINTLN("... finished!");
201
202         // Check if any reads failed and exit early (to try again).
203         if (isnan(humidity) || isnan(temperature)) {
204             SERIALDEBUG_PRINTLN("Failed to read from DHT sensor!");
205             for (int i=0; i<5; i++) {
206                 digitalWrite(LED_BUILTIN, HIGH); // turn the LED on by making the
207                 delay(150);
208                 digitalWrite(LED_BUILTIN, LOW); // turn the LED on by making the
209                 delay(150);
210             }
211             // ok, then wait for another period and try it again
212             os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTERVAL), do_
213         } else {
214             SERIALDEBUG_PRINT("Humidity: ");
215             SERIALDEBUG_PRINT(humidity);
216             SERIALDEBUG_PRINT(" %\t");
217             SERIALDEBUG_PRINT("Temperature: ");
218             SERIALDEBUG_PRINT(temperature);
219             SERIALDEBUG_PRINT(" *C ");
220
221             int16_temperature = 100*temperature;
222             int16_humidity = 100*humidity;
223             mydata[0] = (byte) (int16_temperature >> 8);
224             mydata[1] = (byte) (int16_temperature & 0x00FF);
225             mydata[2] = (byte) (int16_humidity >> 8);
226             mydata[3] = (byte) (int16_humidity & 0x00FF);
227
228             measuredvbat = analogRead(VBATPIN);
229             measuredvbat *= 2; // we divided by 2, so multiply back

```

(continues on next page)

(continued from previous page)

```

230     measuredvbat *= 3.3;    // Multiply by 3.3V, our reference voltage
231     measuredvbat /= 1023;   // convert to voltage
232     int16_vbat = round(measuredvbat * 100);
233     mydata[4] = (byte) (int16_vbat >> 8);
234     mydata[5] = (byte) (int16_vbat & 0x00FF);
235     SERIALDEBUG_PRINT(" %\t");
236     SERIALDEBUG_PRINT("Battery Voltage: ");
237     SERIALDEBUG_PRINTLN(measuredvbat);
238
239     //          LMIC_setTxData2(1, mydata, sizeof(mydata)-1, 0);
240
241     // send the 6 bytes payload to LoRaWAN port 7
242     LMIC_setTxData2(7, mydata, 6, 0);
243     SERIALDEBUG_PRINTLN(F("Packet queued"));
244     digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on by making the
↪ voltage HIGH
245     }
246
247     // LMIC_setTxData2(1, mydata, sizeof(mydata)-1, 0);
248     // Serial.println(F("Packet queued"));
249     }
250     // Next TX is scheduled after TX_COMPLETE event.
251 }
252
253 void setup() {
254     delay(5000);
255
256     pinMode(LED_BUILTIN, OUTPUT);
257     digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
258
259     #ifndef SERIALDEBUG
260         Serial.begin(115200);
261         // while (!Serial);
262     #endif
263
264     dht.begin();
265
266     SERIALDEBUG_PRINTLN(F("Starting"));
267
268     #ifndef VCC_ENABLE
269         // For Pinoccio Scout boards
270         pinMode(VCC_ENABLE, OUTPUT);
271         digitalWrite(VCC_ENABLE, HIGH);
272         delay(1000);
273     #endif
274
275     // LMIC init
276     os_init();
277     // Reset the MAC state. Session and pending data transfers will be discarded.
278     LMIC_reset();
279     LMIC_setClockError(MAX_CLOCK_ERROR * 1 / 100);
280
281     // Set static session parameters. Instead of dynamically establishing a session
282     // by joining the network, precomputed session parameters are be provided.
283     #ifndef PROGMEM
284         // On AVR, these values are stored in flash and only copied to RAM
285         // once. Copy them to a temporary buffer here, LMIC_setSession will

```

(continues on next page)

(continued from previous page)

```

286 // copy them into a buffer of its own again.
287 uint8_t appskey[sizeof(APPSKEY)];
288 uint8_t nwkskey[sizeof(NWKSKEY)];
289 memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
290 memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
291 LMIC_setSession (0x1, DEVADDR, nwkskey, appskey);
292 #else
293 // If not running an AVR with PROGMEM, just use the arrays directly
294 LMIC_setSession (0x1, DEVADDR, NWKSKEY, APPSKEY);
295 #endif
296
297 #if defined(CFG_eu868)
298 // Set up the channels used by the Things Network, which corresponds
299 // to the defaults of most gateways. Without this, only three base
300 // channels from the LoRaWAN specification are used, which certainly
301 // works, so it is good for debugging, but can overload those
302 // frequencies, so be sure to configure the full frequency range of
303 // your network here (unless your network autoconfigures them).
304 // Setting up channels should happen after LMIC_setSession, as that
305 // configures the minimal channel set.
306 // NA-US channels 0-71 are configured automatically
307 LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
↪ // g-band
308 LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B), BAND_CENTI);
↪ // g-band
309 LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
↪ // g-band
310 LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
↪ // g-band
311 LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
↪ // g-band
312 LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
↪ // g-band
313 LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
↪ // g-band
314 LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
↪ // g-band
315 LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK, DR_FSK), BAND_MILLI);
↪ // g2-band
316 // TTN defines an additional channel at 869.525Mhz using SF9 for class B
317 // devices' ping slots. LMIC does not have an easy way to define set this
318 // frequency and support for class B is spotty and untested, so this
319 // frequency is not configured here.
320 #elif defined(CFG_us915)
321 // NA-US channels 0-71 are configured automatically
322 // but only one group of 8 should (a subband) should be active
323 // TTN recommends the second sub band, 1 in a zero based count.
324 // https://github.com/TheThingsNetwork/gateway-conf/blob/master/US-global_conf.
↪ json
325 LMIC_selectSubBand(1);
326 #endif
327
328 // Disable link check validation
329 LMIC_setLinkCheckMode(0);
330
331 // TTN uses SF9 for its RX2 window.
332 LMIC.dn2Dr = DR_SF9;

```

(continues on next page)

(continued from previous page)

```

333
334 // Set data rate and transmit power for uplink (note: txpow seems to be ignored_
↪by the library)
335 LMIC_setDrTxpow(DR_SF7,14);
336
337 // Start job
338 do_send(&sendjob);
339 }
340
341 void loop() {
342     os_runloop_once();
343 }

```

Listing 23: TTN payload decoder for Adafruit M0 LoRa sensor node

```

1  function Decoder (bytes, port) {
2      var result = {};
3      var transformers = {};
4
5      if (port==7) {
6          transformers = {
7              'temperature': function transform (bytes) {
8                  value=bytes[0]*256 + bytes[1];
9                  if (value>=32768) value=value-65536;
10                 return value/100.0;
11             },
12             'humidity': function transform (bytes) {
13                 return (bytes[0]*256 + bytes[1])/100.0;
14             },
15             'vbattery': function transform (bytes) {
16                 return (bytes[0]*256 + bytes[1])/100.0;
17             },
18         }
19
20         result['temperature'] = {
21             value: transformers['temperature'](bytes.slice(0, 2)),
22             uom: 'Celsius',
23         }
24
25         result['humidity'] = {
26             value: transformers['humidity'](bytes.slice(2, 4)),
27             uom: 'Percent',
28         }
29
30         result['vbattery'] = {
31             value: transformers['vbattery'](bytes.slice(4, 6)),
32             uom: 'Volt',
33         }
34     }
35
36     return result;
37 }

```

2.6.5 References

- [Adafruit Feather M0 LoRa microcontroller](#)
- [Adafruit Feather M0 LoRa tutorial](#)
- [IBM LMIC \(LoraMAC-in-C\) library for Arduino](#)
- [Adafruit feather m0 lora 900 end-to-end instructions - End Devices \(Nodes\) - The Things Network](#)
- [Getting Started with AdaFruit Feather M0 LoRa - Telenor Start IoT](#)
- [GitHub - mcci-catena/arduino-lorawan: User-friendly library for using Feather M0 LoRa with The Things Network and LoRaWAN™](#)
- [GitHub - marcuschbehrens/loralife: source code associated with https://www.meetup.com/Internet-of-Things-IoT-LoRaWan-Infrastruktur-4-RheinNeckar/](#)
- [Workshop — LoRaTAS](#)
- [mikenz/Feather_M0_LoRa: Example Arduino code of using an Adafruit Feather M0 LoRa module to send sensor data](#)
- [TTN Ulm - LoRaWAN und LoRa in Ulm | Verkehrszählung mit LoRaWAN und TTN](#)

On battery saving / using the deep sleep mode

- [Full Arduino Mini LoraWAN and 1.3uA Sleep Mode - End Devices \(Nodes\) - The Things Network](#)
- [Adding Method to Adjust hal_ticks Upon Waking Up from Sleep · Issue #109 · matthijskooijman/arduino-lmic](#)
- [minilora-test/minilora-test.ino at cbe686826bd84fac8381de47b5f5b02dd47c2ca0 · tkerby/minilora-test](#)

2.7 Dragino LoRa Arduino Shield

This tutorial is made to showcase the use of Dragino LoRa Arduino board to create a LoRaWAN enabled sensor node. In the following example, a temperature and humidity sensor was used with the Dragino LoRa board.

2.7.1 Hardware

Microcontroller

The employed microcontroller board is an [Arduino Uno R3](#) variant (i.e. it is a cheap clone of the Arduino Uno R3). It is operated by the 8bit ATmega328 microcontroller running at 16MHz. It has 32 KB flash memory (to store the program code), 1 KB EEPROM (to store configuration data), and 2 KB of RAM (to store variables, status information, and buffers). The operating voltage of the board is 5V (this is important when attaching sensors and other peripherals; they also must operate on 5V). The board offers 20 general purpose digital input/output pins (20 GPIOs) of which 6 can be used as analog input pins (with 10bit analog digital converters (ADC)) and 6 as PWM outputs, one serial port (programmable Universal Asynchronous Receiver and Transmitter, UART), one I2C port, one SPI port, one USB port (which is attached to a USB/Serial converter that is connected to the hardware serial port). Arduino Uno R3 compatible boards are available in German shops from around 5 € to 10 €. The original Arduino Uno R3 board costs around 22 €.

The [Dragino LoRa/GPS Shield](#) runs on 5V and is directly attached to the connectors of the Arduino Uno R3 microcontroller board. It comes with a built-in LoRa transmitter and receiver chip SX1276 from the company Semtech that is dedicated to the 868 MHz frequency band. The SX1276 module is connected via SPI interface to the microcontroller. For that purpose, Lora CLK, Lora D0, and Lora DI must be jumpered to SCK, MISO, and MOSI respectively (on the left side of the Dragino shield when looking on the top side of the shield with the Antenna connectors showing to the

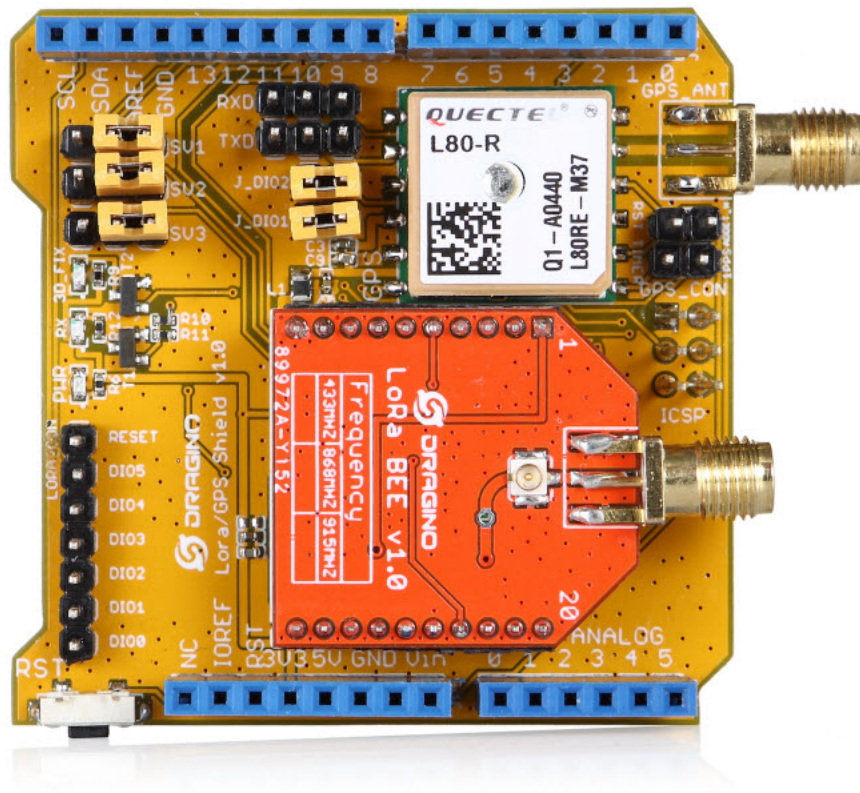


Fig. 21: LoRa/GPS Shield from Dragino. [LoRa/GPS Shield Wiki](#) with explanations, datasheets, and examples.

right). Lora DIO1 and Lora DIO2 must be jumpered to Arduino Digital Pin 6 and Pin 7 respectively. Since the module only implements the LoRa physical layer, the LoRaWAN protocol stack must be implemented in software on the microcontroller. We are using the Arduino library LMIC for that purpose (see below). The implemented LoRaWAN functionality is compatible with LoRaWAN Class A/C.

The board also contains a [Quectel L80 GPS module](#) (based on the MTK MT3339 GPS receiver) with a built-in antenna. According to the Dragino Wiki “this GPS module can calculate and predict orbits automatically using the ephemeris data (up to 3 days) stored in internal flash memory, so the shield can fix position quickly even at indoor signal levels with low power consumption”. The GPS module has a serial UART interface that can be connected in different ways to the Arduino microcontroller. The default data transmission rate is 9600 baud, the default position reporting rate is 1s (1 Hz). The module is capable to report up to 10 positions per second (10 Hz). Supported protocols are [NMEA 0183](#) and [MediaTek PMTK](#). Note that the ATmega328 microcontroller has only one hardware serial UART interface and this is already connected via a USB/Serial converter to the USB port of the Arduino board. In order to attach the serial interface of the GPS module to the microcontroller two general purpose IO lines (GPIOs) are being used and the serial protocol is implemented in software. The GPS_RXD pin on the Dragino Shield must be connected to Arduino Digital Pin 4 and the GPS_TXD pin to Digital Pin 3 using two wires. No jumpers must be present for GPS_RXD and GPS_TXD (besides the two wires mentioned above to Digital Pins 4 and 3). The Dragino LoRa/GPS Shield is available in German shops for around 34 € to 40 €.



Fig. 22: Solar Charger Shield V2.2 from Seestudio.

Since the Arduino Uno R3 board normally has to be powered externally via the USB port or the power connector, we have added the [Solar Charger Shield V2.2](#) from the company Seestudio. This shield is directly attached to the connectors of the Arduino Uno R3 microcontroller board and sits in-between the Arduino board (bottom) and the LoRa/GPS Shield (top). A lithium polymer LiPo battery with 3.7V can be attached to the shield. The 3.7V of the battery is transformed to 5V as required by the Arduino microcontroller board. The battery is automatically recharged when the Arduino board is powered externally (over USB or the power connector). Also a photovoltaic panel with 4.8-6V can be attached to the shield to recharge the battery. The Solar Charger Shield V2.2 can report the current battery voltage level. For that purpose we had to solder a bridge on the shield at the connector marked as ‘R7’. Over

a voltage divider the battery anode is connected to Analog Pin A0 and can be queried using the built-in analog/digital converter. The Solar Charger Shield V2.2 is available in German shops for around 12 € to 18 €.

Sensor

We have attached a DHT22 sensor to the microcontroller board, which measures air temperature and humidity. The minimal time interval between two measurements is 2 seconds. All data transfers between the DHT22 and the microcontroller use a single digital line. The sensor data pin is attached to a GPIO pin (here: Digital Pin 5) of the microcontroller. In addition, a so-called pull-up resistor of 4.7k to 10k Ohm must be connected between the data line and VCC (+3.3V). The [DHT22 datasheet](#) provides more technical details about the DHT22 Sensor. A tutorial on how to use the [DHT22 sensor with Arduino microcontrollers](#) is provided here. The sensor is available in German shops for around 4 € to 10 €.

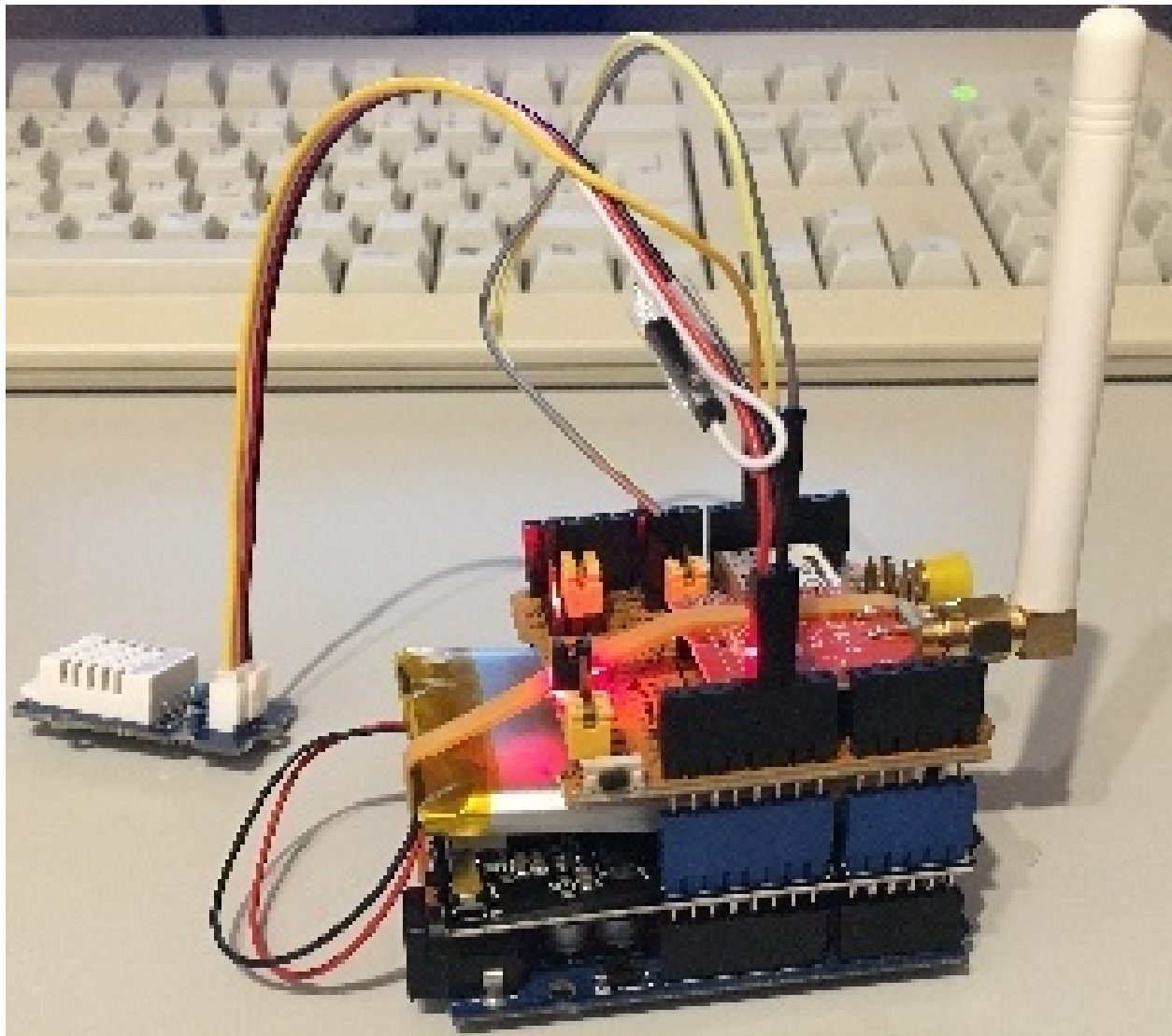


Fig. 23: The Arduino Uno R3 (bottom) with attached Solar Charger Shield and a 2000 mAh lithium polymer LiPo battery (middle), the Dragino LoRa/GPS Shield with attached antenna (top), and an attached DHT22 temperature / humidity sensor (white box on the left).

2.7.2 Software

The sensor node has been programmed using the [Arduino IDE](#). Please note, that in the Arduino framework a program is called a ‘Sketch’.

After the sketch has successfully established a connection to The Things Network it reports the air temperature, humidity, and the voltage of a (possibly) attached LiPo battery every 5 minutes. All three values are being encoded in two byte integer values each (in most significant byte order) and then sent as a 6 bytes data packet to the respective TTN application using LoRaWAN port 7. Please note, that LoRaWAN messages can be addressed to ports 1-255 (port 0 is reserved); these ports are similar to port numbers 0-65535 when using the Internet TCP/IP protocol. Voltage and humidity values are always greater or equal to 0, but the temperature value can also become negative. Negative values are represented as a [two’s complement](#); this must be considered in the Payload Decoding Function used in The Things Network (see [this section](#)).

The next eight bytes contain two 32 bit integer values (MSB) for latitude and longitude . In order to a) provide enough precision and b) avoid negative values, the original angles (given as decimal fractions) are first added with an offset (90.0 degrees for the latitude and 180.0 degrees for the longitude) and then multiplied by 1,000,000. These transformations have to be reverted in the Payload Decoding Function. The next two bytes represent a 16 bit integer value for the altitude (MSB). The next byte contains the current number of satellites seen by the GPS receiver. Note that only when this number is greater or equal to 4 the provided GPS position is a current one. Finally, the last two bytes contain a 16 bit integer value (MSB) for the battery voltage in centivolts (this value will be divided by 100 in the Payload Decoding Function to provide volts). The entire data packet is sent to the respective TTN application using LoRaWAN port 9. Please note, that LoRaWAN messages can be addressed to ports 1-255 (port 0 is reserved); these ports are similar to port numbers 0-65535 when using the Internet TCP/IP protocol.

Currently we are not making use of the sleep mode, because we have to find out how to deal with the GPS receiver in conjunction with deep sleep mode. This means that the board is constantly drawing a significant amount of power reducing battery life considerably. Using the current sketch the sensor node can operate roughly 6 hours on battery power before it has to be recharged. Besides software improvements there are also other possibilities to reduce power consumption: the Arduino board and the Dragino LoRa/GPS Shield have power LEDs which are constantly lit during operation. Furthermore, the Dragino LoRa/GPS Shield has an indicator LED that blinks when the GPS module is successfully receiving position fixes. These LEDs could be desoldered to reduce the energy consumption of the sensor node.

The employed SX1276 LoRa module on the Dragino LoRa/GPS shield does not provide built-in support of the LoRaWAN protocol. Thus, it has to be implemented on the ATmega328 microcontroller. We use the IBM LMIC (LoraMAC-in-C) library for Arduino, which can be downloaded from [this repository](#). Since the ATmega328 microcontroller only has 32 KB of flash memory and the LMIC library is taking most of it, there is only very limited code space left for the application dealing with the sensors (about 2 KB). Nevertheless, this is sufficient to query some sensors like in our example the DHT22 and to decode the GPS data. The source code is given in the following section: *Arduino Sketch for Dragino LoRa sensor node*

2.7.3 Services

The services used for this sensor-node are:

- *TheThingsNetwork* service for LoRaWAN network service.
- *TheThingNetwork- OGC SensorWeb Integration*

Registration of the sensor node with The Things Network (TTN)

The LoRaWAN protocol makes use of a number of different identifiers, addresses, keys, etc. These are required to unambiguously identify devices, applications, as well as to encrypt and decrypt messages. The names and meanings are [nicely explained on a dedicated TTN web page](#).

The sketch given above connects the sensor node with The Things Network (TTN) using the Activation-by-Personalisation (ABP) mode. In this mode, the required keys for data encryption and session management are created manually using the TTN console window and must be pasted into the source code of the sketch below. In order to get this running, you will need to [create a new device in the TTN console window](#). This assumes that you already have a TTN user account (which needs to be created otherwise). In the settings menu of the newly created device the ABP mode must be selected and the settings must be saved. Then copy the DevAddr, the NwkSKey, and in the AppSKey from the TTN console web page of the newly registered device and paste them into the proper places in the sketch above. Please make sure that you choose for each of the three keys the correct byte ordering (MSB for all three keys). A detailed explanation of these steps is given [here](#). Then the sketch can be compiled and uploaded to the Arduino Uno R3 microcontroller.

Important hint: everytime the sensor node is reset or being started again, make sure to reset the frame counter of the registered sensor in the TTN console web page of the registered device. The reason is that in LoRaWAN all transmitted data packets have a frame counter, which is incremented after each data frame being sent. This way a LoRaWAN application can avoid receiving and using the same packet again (replay attack). When TTN receives a data packet, it checks if the frame number is higher than the last one received before. If not, the received packet is considered to be old or a replay attack and is discarded. When the sensor node is reset or being started again, its frame counter is also reset to 0, hence, the TTN application assumes that all new packages are old, because their frame counter is lower than the last frame received (before the reset). A manual frame counter reset is only necessary when registering the node using ABP mode. In OTAA mode the frame counter is automatically reset in the sensor node and the TTN network server.

TTN Payload Decoding

Everytime a data packet is received by a TTN application a dedicated Javascript function is being called (*TTN payload decoder for Dragino LoRa sensor node*). This function can be used to decode the received byte string and to create proper Javascript objects or values that can directly be read by humans when looking at the incoming data packet. This is also useful to format the data in a specific way that can then be forwarded to an external application (e.g. a sensor data platform like [MyDevices](#) or [Thingspeak](#)). Such a forwarding can be configured in the TTN console in the “Integrations” tab. The Payload Decoder Function given below checks if a packet was received on LoRaWAN port 9 and then assumes that it consists of the 17 bytes encoded as described above. It creates the seven Javascript objects ‘temperature’, ‘humidity’, ‘lat’, ‘lon’, ‘altitude’, ‘sat’, and ‘vbattery’. Each object has two fields: ‘value’ holds the value and ‘uom’ gives the unit of measure. The source code can simply be copied and pasted into the ‘decoder’ tab in the TTN console after having selected the application. Choose the option ‘Custom’ in the ‘Payload Format’ field. Note that when you also want to handle other sensor nodes sending packets on different LoRaWAN ports, then the Payload Decoder Function can be extended after the end of the `if (port==9) { ... }` statement by adding `else if (port==7) { ... }` `else if (port==8) { ... }` etc.

The Things Network - OGC SensorWeb Integration

The presented Payload Decoder Function works also with the TTN-OGC SWE Integration for the [52° North Sensor Observation Service \(SOS\)](#). This software component can be downloaded from this [repository](#). It connects a TTN application with a running transactional [Sensor Observation Service 2.0.0 \(SOS\)](#). Data packets received from TTN are imported into the SOS. The SOS persistently stores sensor data from an arbitrary number of sensor nodes and can be queried for the most recent as well as for historic sensor data readings. The 52° North SOS comes with its own REST API and a nice web client allowing to browse the stored sensor data in a convenient way.

We are running an instance of the 52° North SOS and the TTN-OGC SWE Integration. The web client for this LoRaWAN sensor node can be accessed [on this page](#). Here is a screenshot showing the webclient:

2.7.4 Code files



Fig. 24: Web client for data visualization

Listing 24: Arduino Sketch for Dragino LoRa sensor node

```

1  /*****
2  * Arduino Sketch for a LoRaWAN sensor node that is registered with
3  * 'The Things Network' (TTN) www.thethingsnetwork.org
4  *
5  * Author: Thomas H. Kolbe, thomas.kolbe@tum.de
6  * Version: 0.4
7  * Last update: 2018-11-28
8  *
9  * The sensor node is based on the Arduino Uno3 microcontroller board
10 * and the Dragino Lora Shield with GPS receiver. Also a Seeedstudio
11 * Solar Charger Shield V2.2 is connected to provide a battery power
12 * supply with the possibility to use a small PV panel for recharging.
13 * See https://wiki.dragino.com/index.php?title=Lora/GPS_Shield
14 * and http://wiki.seeedstudio.com/Solar_Charger_Shield_V2.2/
15 *
16 * The sensor node uses a DHT22 sensor measuring air temperature and humidity.
17 * The GPS receiver of the Dragino Lora Shield is used to locate the node.
18 * The voltage of an attached LiPo battery is monitored and sent as an
19 * additional observation.
20 *
21 * All three values are encoded as 2 byte integer values each.
22 * Hence, the total message payload is 6 bytes. Before the values are converted
23 * to integers they are multiplied by 100 to preserve 2 digits after the decimal
24 * point. Thus, the received values must be divided by 100 to obtain the measured
25 * values. The payload is sent every 60s to LoRaWAN port 9. The following
26 * Javascript function can be used as a payload decoding function in TTN:

```

(continues on next page)

(continued from previous page)

```

27 *
28 * function Decoder(bytes, port) {
29 *   // Decode an uplink message from a buffer
30 *   // (array) of bytes to an object of fields.
31 *   if (port==7) {
32 *     var decoded = {
33 *       "temperature": (bytes[0] << 8 | bytes[1]) / 100.0,
34 *       "humidity": (bytes[2] << 8 | bytes[3]) / 100.0,
35 *       "vbattery": (bytes[4] << 8 | bytes[5]) / 100.0
36 *     };
37 *   } else {
38 *     var decoded = null;
39 *   }
40 *   return decoded;
41 * }
42 *
43 * In between two data transmissions the microcontroller board can go
44 * into sleep mode to reduce energy consumption for extended operation
45 * time when running on battery. Usage of the sleep mode must be
46 * explicitly configured below.
47 *
48 * Important hint: everytime the sensor node is reset or being started again,
49 * make sure to reset the frame counter of the registered sensor in the
50 * TTN console at https://console.thethingsnetwork.org. The reason is that
51 * in LoRaWAN all transmitted packets have a frame counter, which is
52 * incremented after each data frame being sent. This way a LoRaWAN application
53 * can avoid receiving and using the same packet again (replay attack). When
54 * TTN receives a data packet, it checks if the frame number is higher than
55 * the last one received before. If not, the received packet is considered
56 * to be old or a replay attack and is discarded. When the sensor node is
57 * reset or being started again, its frame counter is also reset to 0, hence,
58 * the TTN application assumes that all new packages are old, because their
59 * frame counter is lower than the last frame received (before the reset).
60 *
61 * Note, that the DHT22 data pin must be connected to Digital Pin 5 of the
62 * Arduino board. A resistor of 4.7k - 10k Ohm must be connected to
63 * the data pin and VCC (+5V). The GPS_RXD pin on the Dragino Shield must
64 * be connected to Arduino Digital Pin 4 and the GPS_TXD pin to Digital Pin 3.
65 * Lora CLK, Lora D0, and Lora DI must be jumpered to SCK, MISO, and MOSI
66 * respectively (on the left side of the Dragino shield when looking on the
67 * top side of the shield with the Antenna connectors shwoing to the right).
68 * Lora DIO1 and Lora DIO2 must be jumpered to Arduino Digital Pin 6 and
69 * Pin 7 respectively. No jumpers must be present for GPS_RXD and GPS_TXD
70 * (besides the two wires mentioned above to Digital Pins 4 and 3).
71 *
72 * The code is based on the Open Source library LMIC implementing the LoRaWAN
73 * protocol stack on top of a given LoRa transceiver module (here: RFM95 from
74 * HopeRF, which uses the Semtech SX1276 LoRa chip). The library is originally
75 * being developed by IBM and has been ported to the Arduino platform. See
76 * notes below from the original developers.
77 *
78 * *****
79 * Copyright (c) 2015 Thomas Telkamp and Matthijs Kooijman
80 *
81 * Permission is hereby granted, free of charge, to anyone
82 * obtaining a copy of this document and accompanying files,
83 * to do whatever they want with them without any restriction,

```

(continues on next page)

(continued from previous page)

```

84  * including, but not limited to, copying, modification and redistribution.
85  * NO WARRANTY OF ANY KIND IS PROVIDED.
86  *
87  * This uses ABP (Activation-by-personalisation), where a DevAddr and
88  * Session keys are preconfigured (unlike OTAA, where a DevEUI and
89  * application key is configured, while the DevAddr and session keys are
90  * assigned/generated in the over-the-air-activation procedure).
91  *
92  * Note: LoRaWAN per sub-band duty-cycle limitation is enforced (1% in
93  * g1, 0.1% in g2), but not the TTN fair usage policy (which is probably
94  * violated by this sketch when left running for longer)!
95  *
96  * To use this sketch, first register your application and device with
97  * the things network, to set or generate a DevAddr, NwkSKey and
98  * AppSKey. Each device should have their own unique values for these
99  * fields.
100  *
101  * Do not forget to define the radio type correctly in config.h.
102  *
103  *****/
104
105  // If the following line is uncommented, messages are being printed out to the
106  // serial connection for debugging purposes. When using the Arduino Integrated
107  // Development Environment (Arduino IDE), these messages are displayed in the
108  // Serial Monitor selecting the proper port and a baudrate of 115200.
109
110  // #define SERIALDEBUG
111
112  #ifndef SERIALDEBUG
113      #define SERIALDEBUG_PRINT(...) Serial.print(__VA_ARGS__)
114      #define SERIALDEBUG_PRINTLN(...) Serial.println(__VA_ARGS__)
115  #else
116      #define SERIALDEBUG_PRINT(...)
117      #define SERIALDEBUG_PRINTLN(...)
118  #endif
119
120  // If the following line is uncommented, the sensor node goes into sleep mode
121  // in between two data transmissions. Also the 2secs time between the
122  // initialization of the DHT22 sensor and the reading of the observations
123  // is spent in sleep mode.
124  // Note, that on the Adafruit Feather 32u4 LoRa board the Serial connection
125  // gets lost as soon as the board goes into sleep mode, and it will not be
126  // established again. Thus, the definition of SERIALDEBUG should be commented
127  // out above when using sleep mode with this board.
128
129  // #define SLEEPMODE
130
131  #ifndef SLEEPMODE
132      #include <Adafruit_SleepyDog.h>
133  #endif
134
135  #include <lmic.h>
136  #include <hal/hal.h>
137  #include <SPI.h>
138
139  #include <DHT.h>
140  #define DHTPIN          5          // Arduino Digital Pin which is connected to the
    ↪ DHT sensor for Arduino.

```

(continues on next page)

(continued from previous page)

```

141 #define DHTTYPE          DHT22      // DHT 22 (AM2302)
142 DHT dht (DHTPIN, DHTTYPE);          // create the sensor object
143
144 #include <TinyGPS.h>
145 TinyGPS gps;
146 bool newGPSdata = false;
147
148 #include <SoftwareSerial.h>
149 SoftwareSerial SWSerial(3, 4);
150
151 #define VBATPIN A0                // battery voltage is measured from Analog Input A0_
152 ↪for Sseed Solar Shield V2.2
153
154 // The following three constants (NwksKey, AppSKey, DevAddr) must be changed
155 // for every new sensor node. We are using the LoRaWAN ABP mode (activation by
156 // personalisation) which means that each sensor node must be manually registered
157 // in the TTN console at https://console.thethingsnetwork.org before it can be
158 // started. In the TTN console create a new device and choose ABP mode in the
159 // settings of the newly created device. Then, let TTN generate the NwksKey and
160 // and the AppSKey and copy them (together with the device address) from the webpage
161 // and paste them below.
162
163 // LoRaWAN NwksKey, network session key
164 // This should be in big-endian (aka msb).
165 static const PROGMEM u1_t NWKSKEY[16] = {NETWORK_SESSION_KEY_HERE_IN_MSB_FORMAT};
166
167 // LoRaWAN AppSKey, application session key
168 // This should also be in big-endian (aka msb).
169 static const u1_t PROGMEM APPSKEY[16] = {APPLICATION_SESSION_KEY_HERE_IN_MSB_FORMAT};
170
171 // LoRaWAN end-device address (DevAddr)
172 // See http://thethingsnetwork.org/wiki/AddressSpace
173 // The library converts the address to network byte order as needed, so this should_
174 ↪be in big-endian (aka msb) too.
175 static const u4_t DEVADDR = 0x260XXXXX ; // <-- Change this address for every node!
176
177 // These callbacks are only used in over-the-air activation, so they are
178 // left empty here (we cannot leave them out completely unless
179 // DISABLE_JOIN is set in config.h, otherwise the linker will complain).
180 void os_getArtEui (u1_t* buf) { }
181 void os_getDevEui (u1_t* buf) { }
182 void os_getDevKey (u1_t* buf) { }
183
184 // The following array of bytes is a placeholder to contain the message payload
185 // which is transmitted to the LoRaWAN gateway. We are currently only using 6 bytes.
186 // Please make sure to extend the size of the array, if more sensors should be
187 // attached to the sensor node and the message payload becomes larger than 10 bytes.
188 static uint8_t mydata[17] = {0, 1, 2, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0xE, 0xF, 0x10}
189 ↪;
190
191 static osjob_t sendjob;
192
193 // Schedule transmission every TX_INTERVAL seconds (might become longer due to duty
194 // cycle limitations). The total interval time is 2secs for the measurement
195 // plus 3secs for the LoRaWAN packet transmission plus TX_INTERVAL_AFTER_SLEEP seconds
196 // plus SLEEP_TIME seconds (microcontroller in sleep mode)
197 const unsigned TX_INTERVAL = 300;          // overall cycle time (send one set of_
198 ↪observations every 5 mins)

```

(continues on next page)

(continued from previous page)

```

195 // const unsigned TX_INTERVAL = 30;          // overall cycle time (send one set of_
    ↳ observations every 30 secs)
196 const unsigned TX_TIME = 3;                  // rough estimate of transmission time of a_
    ↳ single packet
197 const unsigned MEASURE_TIME = 2;             // seconds measuring time
198 const unsigned SLEEP_TIME = TX_INTERVAL - TX_TIME - MEASURE_TIME;
199 const unsigned WAIT_TIME = TX_INTERVAL - TX_TIME - MEASURE_TIME;
200
201 // Pin mapping of the LoRa transceiver. Please make sure that DIO1 is connected
202 // to Arduino Digital Pin 6 using an external wire. DIO2 is left unconnected
203 // (it is only required, if FSK modulation instead of LoRa would be used).
204 const lmic_pinmap lmic_pins = {
205     .nss = 10,
206     .rxtx = LMIC_UNUSED_PIN,
207     .rst = 9,
208     .dio = {2, 6, 7},
209 };
210
211 void onEvent (ev_t ev) {
212     SERIALDEBUG_PRINT(os_getTime());
213     SERIALDEBUG_PRINT(": ");
214     switch(ev) {
215         case EV_SCAN_TIMEOUT:
216             SERIALDEBUG_PRINTLN(F("EV_SCAN_TIMEOUT"));
217             break;
218         case EV_BEACON_FOUND:
219             SERIALDEBUG_PRINTLN(F("EV_BEACON_FOUND"));
220             break;
221         case EV_BEACON_MISSED:
222             SERIALDEBUG_PRINTLN(F("EV_BEACON_MISSED"));
223             break;
224         case EV_BEACON_TRACKED:
225             SERIALDEBUG_PRINTLN(F("EV_BEACON_TRACKED"));
226             break;
227         case EV_JOINING:
228             SERIALDEBUG_PRINTLN(F("EV_JOINING"));
229             break;
230         case EV_JOINED:
231             SERIALDEBUG_PRINTLN(F("EV_JOINED"));
232             break;
233         case EV_RFU1:
234             SERIALDEBUG_PRINTLN(F("EV_RFU1"));
235             break;
236         case EV_JOIN_FAILED:
237             SERIALDEBUG_PRINTLN(F("EV_JOIN_FAILED"));
238             break;
239         case EV_REJOIN_FAILED:
240             SERIALDEBUG_PRINTLN(F("EV_REJOIN_FAILED"));
241             break;
242         case EV_TXCOMPLETE:
243             digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the_
    ↳ voltage LOW
244             SERIALDEBUG_PRINTLN(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
245             if (LMIC.txrxFlags & TXRX_ACK)
246                 SERIALDEBUG_PRINTLN(F("Received ack"));
247             if (LMIC.dataLen) {
248 #ifdef SERIALDEBUG

```

(continues on next page)

(continued from previous page)

```

249     SERIALDEBUG_PRINT(F("Received "));
250     SERIALDEBUG_PRINT(LMIC.dataLen);
251     SERIALDEBUG_PRINT(F(" bytes of payload: 0x"));
252     for (int i=0; i<LMIC.dataLen; i++) {
253         if (LMIC.frame[LMIC.dataBeg + i] < 0x10) {
254             SERIALDEBUG_PRINT(F("0"));
255         }
256         SERIALDEBUG_PRINT(LMIC.frame[LMIC.dataBeg + i], HEX);
257     }
258     SERIALDEBUG_PRINTLN();
259 #endif
260     // add your code to handle a received downlink data packet here
261 }
262
263 #ifndef SLEEPMODE
264     // Schedule next transmission in 1ms second after the board returns from_
↪ sleep mode
265     os_setTimedCallback(&sendjob, os_getTime()+ms2osticks(1), do_send);
266
267     SERIALDEBUG_PRINTLN("going to sleep now ... ");
268     // lmric library sleeps automatically after transmission has been completed
269     for(int i= 0; i < SLEEP_TIME / 8; i++) {
270         Watchdog.sleep(8000); // maximum seems to be 8 seconds
271         SERIALDEBUG_PRINT('.');
272     }
273     if (SLEEP_TIME % 8) {
274         Watchdog.sleep((SLEEP_TIME % 8)*1000);
275         SERIALDEBUG_PRINT('*');
276     }
277     SERIALDEBUG_PRINTLN("... woke up again");
278
279     // We need to reset the duty cycle limits within the LMIC library.
280     // The reason is that in sleep mode the Arduino system timers millis and_
↪ micros
281     // do not get incremented. However, LMIC monitors the adherence to the
282     // LoRaWAN duty cycle limitations using the system timers millis and_
↪ micros.
283     // Since LMIC does not know that we have slept for a long time and duty
284     // cycle requirements in fact are met, we must reset the respective LMIC_
↪ timers
285     // in order to prevent the library to wait for some extra time (which_
↪ would
286     // not use sleep mode and, thus, would waste battery energy).
287     LMIC.bands[BAND_MILLI].avail = os_getTime();
288     LMIC.bands[BAND_CENTI].avail = os_getTime();
289     LMIC.bands[BAND_DECI].avail = os_getTime();
290 #else
291     // Schedule next transmission
292     os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(WAIT_TIME), do_
↪ send);
293 #endif
294     break;
295     case EV_LOST_TSYNC:
296         SERIALDEBUG_PRINTLN(F("EV_LOST_TSYNC"));
297         break;
298     case EV_RESET:
299         SERIALDEBUG_PRINTLN(F("EV_RESET"));

```

(continues on next page)

(continued from previous page)

```

300         break;
301     case EV_RXCOMPLETE:
302         // data received in ping slot
303         SERIALDEBUG_PRINTLN(F("EV_RXCOMPLETE"));
304         break;
305     case EV_LINK_DEAD:
306         SERIALDEBUG_PRINTLN(F("EV_LINK_DEAD"));
307         break;
308     case EV_LINK_ALIVE:
309         SERIALDEBUG_PRINTLN(F("EV_LINK_ALIVE"));
310         break;
311     default:
312         SERIALDEBUG_PRINTLN(F("Unknown event"));
313         break;
314 }
315 }
316
317 void do_send(osjob_t* j){
318     // Check if there is not a current TX/RX job running
319     if (LMIC.opmode & OP_TXRXPEND) {
320         SERIALDEBUG_PRINTLN(F("OP_TXRXPEND, not sending"));
321     } else {
322         // Prepare upstream data transmission at the next possible time.
323
324         float temperature, humidity, measuredvbat, lat, lon, alt;
325         int16_t int16_temperature, int16_humidity, int16_vbat, int16_alt;
326         int32_t int32_lat, int32_lon;
327         unsigned long age;
328         byte sat=0;
329
330         // Start a measurement to update the sensor's internal temperature & humidity
331         // Note, that when fetching measurements from a DHT22 sensor, the reported
332         // values belong to the measurement BEFORE the current measurement.
333         // Therefore, in order to get current observations, we first perform a new
334         // and wait 2 secs (which is the minimum time between two sensor observations)
335         // the DHT22) and then directly retrieve the observations again.
336         temperature = dht.readTemperature();
337         // temperature = 23;
338         #ifndef SLEEPMODE
339             Watchdog.sleep(2000);
340         #else
341             delay(2000);
342         #endif
343         // Now read the recently measured temperature (2 secs ago) as Celsius (the
344         // default)
345         temperature = dht.readTemperature();
346         // temperature = 23;
347         // Read the recently measured humidity (2 secs ago)
348         humidity = dht.readHumidity();
349         // humidity = 66;
350
351         // Check if any reads failed and exit early (to try again).
352         if (isnan(humidity) || isnan(temperature)) {
353             SERIALDEBUG_PRINTLN("Failed to read from DHT sensor!");

```

(continues on next page)

(continued from previous page)

```

353 // blink the LED five times to indicate that the sensor values could not
↪be read
354 for (int i=0; i<5; i++) {
355     digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on by making the
↪voltage HIGH
356     delay(150);
357     digitalWrite(LED_BUILTIN, LOW);    // turn the LED on by making the
↪voltage HIGH
358     delay(150);
359 }
360 // ok, then wait for another period and try it again
361 os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTERVAL), do_
↪send);
362 } else {
363     SERIALDEBUG_PRINT("Humidity: ");
364     SERIALDEBUG_PRINT(humidity);
365     SERIALDEBUG_PRINT(" %\t");
366     SERIALDEBUG_PRINT("Temperature: ");
367     SERIALDEBUG_PRINT(temperature);
368     SERIALDEBUG_PRINT(" °C ");
369
370     int16_temperature = round(100.0*temperature);
371     int16_humidity = round(100.0*humidity);
372     mydata[0] = (byte) (int16_temperature >> 8);
373     mydata[1] = (byte) (int16_temperature & 0x00FF);
374     mydata[2] = (byte) (int16_humidity >> 8);
375     mydata[3] = (byte) (int16_humidity & 0x00FF);
376
377     if (newGPSdata) {
378         gps.f_get_position(&lat, &lon, &age);
379         int32_lat = round(1000000.0*(lat+90.0));
380         int32_lon = round(1000000.0*(lon+180.0));
381         alt = gps.f_altitude();
382         int16_alt = round(alt);
383         sat = gps.satellites();
384         mydata[4] = (byte) (int32_lat >> 24);
385         mydata[5] = (byte) ((int32_lat >> 16) & 0x00FF);
386         mydata[6] = (byte) ((int32_lat >> 8) & 0x0000FF);
387         mydata[7] = (byte) (int32_lat & 0x000000FF);
388         mydata[8] = (byte) (int32_lon >> 24);
389         mydata[9] = (byte) ((int32_lon >> 16) & 0x00FF);
390         mydata[10] = (byte) ((int32_lon >> 8) & 0x0000FF);
391         mydata[11] = (byte) (int32_lon & 0x000000FF);
392         mydata[12] = (byte) (int16_alt >> 8);
393         mydata[13] = (byte) (int16_alt & 0x00FF);
394         mydata[14] = sat;
395     } else {
396         mydata[14] = 0;
397     }
398
399 #ifdef VBATPIN
400     measuredvbat = analogRead(VBATPIN);
401     measuredvbat *= 2.0;    // we divided by 2, so multiply back
402     measuredvbat *= 5.0;    // Multiply by 5V, our reference voltage
403     measuredvbat /= 1023.0; // convert to voltage
404 #else
405     measuredvbat = 0.0;

```

(continues on next page)

(continued from previous page)

```

406 #endif
407     int16_vbat = round(measuredvbat * 100.0);
408     mydata[15] = (byte) (int16_vbat >> 8);
409     mydata[16] = (byte) (int16_vbat & 0x00FF);
410     SERIALDEBUG_PRINT(" \t");
411     SERIALDEBUG_PRINT("Battery Voltage: ");
412     SERIALDEBUG_PRINT(measuredvbat);
413     SERIALDEBUG_PRINTLN(" V");
414
415     // Send the 17 bytes payload to LoRaWAN port 9 and do not request an_
↪acknowledgement.
416     // The following call does not directly sends the data, but puts a "send_
↪job"
417     // in the job queue. This job eventually is performed in the call "os_
↪runloop_once();"
418     // issued repeatedly in the "loop()" method below. After the transmission_
↪is
419     // complete, the EV_TXCOMPLETE event is signaled, which is handled in the
420     // event handler method "onEvent (ev_t ev)" above. In the EV_TXCOMPLETE_
↪branch
421     // then a new call to the "do_send(osjob_t* j)" method is being prepared_
↪for
422     // delayed execution with a waiting time of TX_INTERVAL seconds.
423     LMIC_setTxData2(9, mydata, 17, 0);
424     SERIALDEBUG_PRINTLN(F("Packet queued"));
425     newGPSdata=false;
426     digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on by making the_
↪voltage HIGH
427
428     // Next TX is scheduled after TX_COMPLETE event.
429 }
430 }
431 }
432
433 void setup() {
434     delay(5000);                // give enough time to open serial monitor (if_
↪needed)
435
436     pinMode(LED_BUILTIN, OUTPUT);
437     digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
438
439 #ifndef SERIALDEBUG
440     Serial.begin(115200);
441     // while (!Serial);
442 #endif
443
444     dht.begin();                // initialize DHT22 sensor
445
446     SERIALDEBUG_PRINTLN(F("Starting"));
447
448 #ifdef VCC_ENABLE
449     // For Pinoccio Scout boards
450     pinMode(VCC_ENABLE, OUTPUT);
451     digitalWrite(VCC_ENABLE, HIGH);
452     delay(1000);
453 #endif
454

```

(continues on next page)

(continued from previous page)

```

455     SWSerial.begin(9600);
456
457     // LMIC init
458     os_init();
459     // Reset the MAC state. Session and pending data transfers will be discarded.
460     LMIC_reset();
461     LMIC_setClockError(MAX_CLOCK_ERROR * 1 / 100);
462
463     // Set static session parameters. Instead of dynamically establishing a session
464     // by joining the network, precomputed session parameters are be provided.
465     #ifdef PROGMEM
466     // On AVR, these values are stored in flash and only copied to RAM
467     // once. Copy them to a temporary buffer here, LMIC_setSession will
468     // copy them into a buffer of its own again.
469     uint8_t appskey[sizeof(APPSKEY)];
470     uint8_t nwkskey[sizeof(NWKSKEY)];
471     memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
472     memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
473     LMIC_setSession(0x1, DEVADDR, nwkskey, appskey);
474     #else
475     // If not running an AVR with PROGMEM, just use the arrays directly
476     LMIC_setSession(0x1, DEVADDR, NWKSKEY, APPSKEY);
477     #endif
478
479     #if defined(CFG_eu868)
480     // Set up the channels used by the Things Network, which corresponds
481     // to the defaults of most gateways. Without this, only three base
482     // channels from the LoRaWAN specification are used, which certainly
483     // works, so it is good for debugging, but can overload those
484     // frequencies, so be sure to configure the full frequency range of
485     // your network here (unless your network autoconfigures them).
486     // Setting up channels should happen after LMIC_setSession, as that
487     // configures the minimal channel set.
488     // NA-US channels 0-71 are configured automatically
489     LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
490     ↪ // g-band
491     LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B), BAND_CENTI);
492     ↪ // g-band
493     LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
494     ↪ // g-band
495     LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
496     ↪ // g-band
497     LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
498     ↪ // g-band
499     LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
500     ↪ // g-band
501     LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
502     ↪ // g-band
503     LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);
504     ↪ // g-band
505     LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK, DR_FSK), BAND_MILLI);
506     ↪ // g2-band
507     // TTN defines an additional channel at 869.525Mhz using SF9 for class B
508     // devices' ping slots. LMIC does not have an easy way to define set this
509     // frequency and support for class B is spotty and untested, so this
510     // frequency is not configured here.
511     #elif defined(CFG_us915)

```

(continues on next page)

(continued from previous page)

```

503 // NA-US channels 0-71 are configured automatically
504 // but only one group of 8 should (a subband) should be active
505 // TTN recommends the second sub band, 1 in a zero based count.
506 // https://github.com/TheThingsNetwork/gateway-conf/blob/master/US-global_conf.
↪ json
507 LMIC_selectSubBand(1);
508 #endif
509
510 // Disable link check validation
511 LMIC_setLinkCheckMode(0);
512
513 // TTN uses SF9 for its RX2 window.
514 LMIC.dn2Dr = DR_SF9;
515
516 // Set data rate and transmit power for uplink (note: txpow seems to be ignored_
↪ by the library)
517 // LMIC_setDrTxpow(DR_SF7,14);
518 LMIC_setDrTxpow(DR_SF9,14);
519
520 // Start job. This will initiate the repetitive sending of data packets,
521 // because after each data transmission, a delayed call to "do_send()"
522 // is being scheduled again.
523 do_send(&sendjob);
524 }
525
526 void loop() {
527 /*
528 // read from port 1, send to port 0:
529 if (Serial.available()) {
530     int inByte = Serial.read();
531     SWSerial.write(inByte);
532 }
533
534 // read from port 0, send to port 1:
535 if (SWSerial.available()) {
536     int inByte = SWSerial.read();
537     Serial.write(inByte);
538 }
539 */
540 unsigned long chars;
541 unsigned short sentences, failed;
542
543 // For one second we parse GPS data and report some key values
544 // for (unsigned long start = millis(); millis() - start < 1000;)
545 // {
546     while (SWSerial.available())
547     {
548         char c = SWSerial.read();
549         // Serial.write(c); // uncomment this line if you want to see the GPS data_
↪ flowing
550         if (gps.encode(c)) // Did a new valid sentence come in?
551             newGPSdata = true;
552     }
553 // }
554
555 /*
556     if (newGPSdata)

```

(continues on next page)

(continued from previous page)

```

557 {
558     float flat, flon;
559     unsigned long age;
560     gps.f_get_position(&flat, &flon, &age);
561     Serial.print("LAT=");
562     Serial.print(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat, 6);
563     Serial.print(" LON=");
564     Serial.print(flou == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flon, 6);
565     Serial.print(" ALT=");
566     Serial.print(gps.f_altitude() == TinyGPS::GPS_INVALID_F_ALTITUDE ? 0.0 : gps.f_
↪altitude(), 6);
567     Serial.print(" SAT=");
568     Serial.print(gps.satellites() == TinyGPS::GPS_INVALID_SATELLITES ? 0 : gps.
↪satellites());
569     Serial.print(" PREC=");
570     Serial.print(gps.hdop() == TinyGPS::GPS_INVALID_HDOP ? 0 : gps.hdop());
571 }
572
573 gps.stats(&chars, &sentences, &failed);
574 Serial.print(" CHARS=");
575 Serial.print(chars);
576 Serial.print(" SENTENCES=");
577 Serial.print(sentences);
578 Serial.print(" CSUM ERR=");
579 Serial.println(failed);
580 if (chars == 0)
581     Serial.println("** No characters received from GPS: check wiring **");
582 */
583 os_runloop_once();
584 }

```

Listing 25: TTN payload decoder for Dragino LoRa sensor node

```

1  function Decoder(bytes, port) {
2      var result = {};
3      var transformers = {};
4
5      if (port == 9) {
6          transformers = {
7              temperature: function transform(bytes) {
8                  value = bytes[0] * 256 + bytes[1];
9                  if (value >= 32768) value = value - 65536;
10                 return value / 100.0;
11             },
12             humidity: function transform(bytes) {
13                 return (bytes[0] * 256 + bytes[1]) / 100.0;
14             },
15             lat: function transform(bytes) {
16                 return (
17                     (bytes[0] * 16777216 + bytes[1] * 65536 + bytes[2] * 256 + bytes[3]) /
18                     1000000.0 -
19                     90.0
20                 );
21             },
22             lon: function transform(bytes) {
23                 return (

```

(continues on next page)

(continued from previous page)

```

24         (bytes[0] * 16777216 + bytes[1] * 65536 + bytes[2] * 256 + bytes[3]) /
25         1000000.0 -
26         180.0
27     );
28 },
29     altitude: function transform(bytes) {
30         return bytes[0] * 256 + bytes[1];
31     },
32     sat: function transform(bytes) {
33         return bytes[0];
34     },
35     vbattery: function transform(bytes) {
36         return (bytes[0] * 256 + bytes[1]) / 100.0;
37     }
38 };
39
40 result["temperature"] = {
41     value: transformers["temperature"](bytes.slice(0, 2)),
42     uom: "Celsius"
43 };
44
45 result["humidity"] = {
46     value: transformers["humidity"](bytes.slice(2, 4)),
47     uom: "Percent"
48 };
49
50 result["lat"] = {
51     value: transformers["lat"](bytes.slice(4, 8)),
52     uom: "Degree"
53 };
54
55 result["lon"] = {
56     value: transformers["lon"](bytes.slice(8, 12)),
57     uom: "Degree"
58 };
59
60 result["altitude"] = {
61     value: transformers["altitude"](bytes.slice(12, 14)),
62     uom: "Meter"
63 };
64
65 result["sat"] = {
66     value: transformers["sat"](bytes.slice(14, 15)),
67     uom: "Count"
68 };
69
70 result["vbattery"] = {
71     value: transformers["vbattery"](bytes.slice(15, 17)),
72     uom: "Volt"
73 };
74
75 return result;
76 }
77 }

```


2.7.5 References

- [Arduino Uno R3 microcontroller](#)
- [FAQ on Arduino microcontrollers from Adafruit](#)
- [Dragino Lora/GPS Shield Wiki](#)
- [Dragino Lora/GPS Shield github](#)
- [Seeedstudio Solar Charger Shield V2.2](#)
- [IBM LMIC \(LoraMAC-in-C\) library for Arduino](#)
- [Connect to TTN - Wiki for Dragino Project](#)
- [dragino/Arduino-Profile-Examples/Arduino_LMIC.ino GitHub](#)
- [dragino/Arduino-Profile-Examples/tinygps_example.ino GitHub](#)
- [goodcheney/ttn_mapper/gps_shield at master GitHub](#)

On battery saving / using the deep sleep mode (these are written for other microcontroller boards, but do apply for the Arduino Uno R3 and the Dragino Lora/GPS Shield, too):

- [Adafruit Feather 32u4 LoRa - long transmission time after deep sleep - End Devices \(Nodes\) - The Things Network](#)
- [Full Arduino Mini LoraWAN and 1.3uA Sleep Mode - End Devices \(Nodes\) - The Things Network](#)
- [Adding Method to Adjust hal_ticks Upon Waking Up from Sleep · Issue #109 · matthijskooijman/arduino-lmic](#)
- [minilora-test/minilora-test.ino at cbe686826bd84fac8381de47b5f5b02dd47c2ca0 · tkerby/minilora-test](#)
- [Arduino-LMIC library with low power mode - Mario Zwiers](#)

2.8 Pycom LoPy4

This tutorial is made to showcase the use of Pycom LoPy4 board to create a LoRaWAN enabled sensor node. In the following example, a temperature and humidity sensor was used with the Pycom LoPy4 board.

2.8.1 Hardware

Microcontroller

The [Pycom LoPy4](#) is a microcontroller board offering many radio frequency (RF) connection options, namely LoRa (and LoRaWAN), SIGFOX, Bluetooth (Classic and Low Energy, BLE), and WiFi. In contrast to most other microcontroller boards the LoPy4 is programmed in [MicroPython](#), which is a special subset of the Python 3 programming language and libraries for microcontrollers. The module is operated by the [Espressif ESP32 microcontroller](#) board, which contains a dual-core Xtensa 32bit LX6 processor running with up to 240MHz, 8 MB of flash memory (to store the program code and some files within a file system), and 520 KB of RAM (to store variables, status information, and buffers). The ESP32 module also has built-in WiFi and Bluetooth LE connectivity. In addition, the LoPy4 has 4 MB of PSRAM (pseudo static RAM) that is used as a memory extension for the ESP32. The operating voltage of the board is 3.3V (this is important when attaching sensors and other peripherals; they also must operate on 3.3V). The board offers 18 general purpose input/output pins (18 GPIOs), from which up to 12 can be used as analog input pins (with 12bit analog digital converters (ADC)) and two as analog output pins (8bit digital analog converter (DAC)). Most GPIO pins can be configured for specific hardware protocols. In total 3 serial ports (programmable Universal Asynchronous Receiver and Transmitter, UART), 2 I2C ports, 3 SPI ports, 1 CAN bus, 1 PWM channel, and an I2S

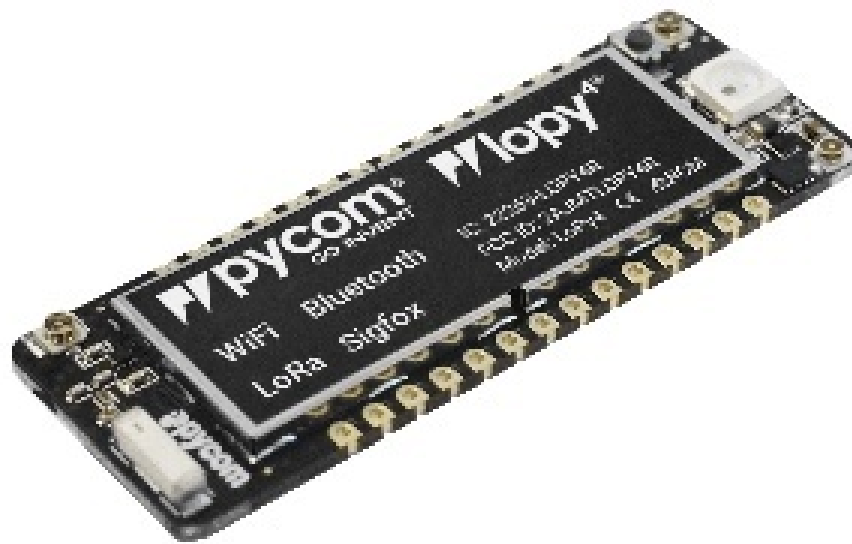


Fig. 25: LoPy4 from Pycom. [LoPy4 pinout](#), [documentation](#), [example code](#).

port can be utilized. The board has a built-in RGB LED that can be programmed by the user. The LoPy4 is available from the manufacturer for around 35 €.

The LoPy4 needs to be operated on 3.5V – 5.5V put to the VIN pin. The onboard regulator brings it down to 3.3V. The 3.3V pin can only be used as an output. Do not feed 3.3V into this pin as this could damage the regulator. The board can be programmed over the serial interface, or via WiFi using a telnet or FTP connection. By default, the LoPy4 acts as a WiFi access point (SSID: lopy4-wlan-XXXX, Password: www.pycom.io) and a user can connect to the module after joining the WiFi network in order to upload user programs and data files.

A WiFi and Bluetooth antenna is mounted on the LoPy board, but also an external antenna can be connected via an SMA-type connector. The LoRa or SIGFOX antenna has to be connected via an SMA-type connector. The LoRa transmitter and receiver is encapsulated within a LoRa module. It uses the LoRa chip SX1276 from the company Semtech and can be configured to work either in the 433 MHz, 868 MHz, or 915 MHz frequency band. The LoRa module is connected via SPI interface to the microcontroller and all of the required connections of the LoRa transceiver pins with the microcontroller are already built-in on the LoPy4 board. Since the module only implements the LoRa physical layer, the LoRaWAN protocol stack is implemented in software on the microcontroller. The implemented LoRaWAN functionality is compatible with LoRaWAN Class A/C.

Expansion board 3.0

The LoPy4 can be attached to the [Pycom Expansion Board 3.0](#). The board offers a USB port that is connected internally via a USB/Serial converter to the default serial port (UART) of the LoPy4. This allows to easily upload programs and data files to the LoPy4 from a developer computer over USB connection. The expansion board also comes with a connector for a 3.7V lithium polymer (LiPo) battery with an additional battery charger circuit. When the expansion board is connected via USB to a developer computer or to an USB charger, an attached battery will be automatically charged. The battery voltage can be monitored via the LoPy4 analog to digital converter (ADC). The board also comes with a user LED, a user switch, and a MicroSD card slot to read and write files from the LoPy4, for example, to log recorded data. The LoPy Expansion Board 3.0 is available from the manufacturer for 16 €.

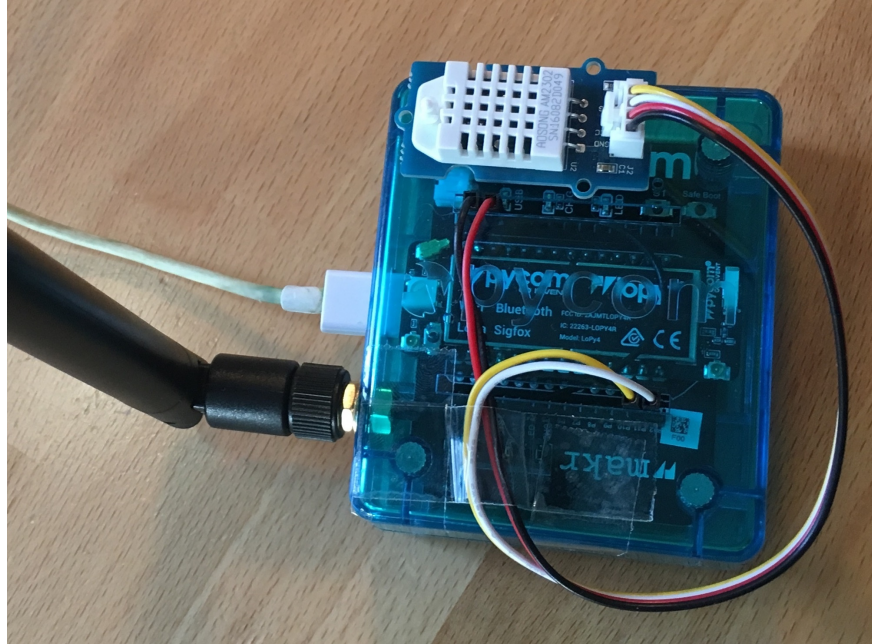


Fig. 26: The Pycom LoPy4 with the Expansion Board 3 (inside the blue case) and an externally attached DHT22 temperature and humidity sensor.

Sensor

We have attached a [Grove DHT22 sensor](#) module to the expansion board, which measures air temperature and humidity. The minimal time interval between two measurements is 2 seconds. All data transfers between the DHT22 and the microcontroller use a single digital line. The sensor data pin is attached to a GPIO pin (here: GPIO22) of the expansion board. The 5V pin of the Grove module is connected to 3V3 of the expansion board, and the GND of the Grove module to GND of the expansion board. The [DHT22 datasheet](#) can be accessed here. The sensor is available in German shops for around 4 € to 10 €.

2.8.2 Software

The sensor node has been programmed in the [MicroPython](#) language. We use [Microsoft's Visual Studio Code](#) platform with the [Pymakr plugin](#) to edit and upload the program. The Pymakr plugin is developed by Pycom and can be used with either Visual Studio Code or the [Atom Text Editor](#). Both IDEs can be downloaded free of charge; Atom is also Open Source software. Note that MicroPython programs do not need to be compiled (like Java or C/C++ programs). The source code is interpreted by the Microcontroller instead.

The source code consists of the following two files, [main.py](#) and [boot.py](#). They must be copied into the base folder on the LoPy4. We use a library for the DHT22 written by Erik de Lange. It can be downloaded from [the following link](#) but is also provided here [dht22.py](#). The library has to be copied into the subdirectory “lib” on the LoPy4.

After the program has successfully established a connection to The Things Network it reports the air temperature, humidity, and the voltage of an attached LiPo battery every 5 minutes. Since we are running the device on an USB charger, the program does not check the battery level and the transferred value is always set to 0 V. All three values are being encoded in two byte integer values each (in most significant byte order) and then sent as a 6 bytes data packet to the respective TTN application using LoRaWAN port 7. Please note, that LoRaWAN messages can be addressed to ports 1-255 (port 0 is reserved); these ports are similar to port numbers 0-65535 when using the Internet TCP/IP protocol. Voltage and humidity values are always greater or equal to 0, but the temperature value can also become negative. Negative values are represented as a [two's complement](#); this must be considered in the Payload Decoding

Function used in The Things Network (see [below](#)).

The program as given above does not make use of the deep sleep mode or any other power saving method. In between two sensor readings the microcontroller is busy ‘doing nothing’ until the waiting time before the next measurement is over. When the LoPy4 should be operated on battery, the [power saving modes](#) of the LoPy4 should be investigated. Note that this will require to restructure the [main.py](#) program significantly.

2.8.3 Services

The services used for this sensor-node are:

- [TheThingsNetwork](#) service for LoRaWAN network service.
- [TheThingsNetwork - OGC SensorWeb](#) integration for uploading LoRaWAN sensor data into OGC infrastructure.

Registration of the sensor node with The Things Network (TTN)

The LoRaWAN protocol makes use of a number of different identifiers, addresses, keys, etc. These are required to unambiguously identify devices, applications, as well as to encrypt and decrypt messages. The names and meanings are nicely explained on a [dedicated TTN web page](#).

The program given above connects the sensor node with The Things Network (TTN) using the Over-the-Air-Activation (OTAA) mode. In this mode, we use the three keys AppEUI, DevEUI, AppKey. The DevEUI is pre-programmed into the LoPy4. In order to register the device with TTN, you first need to fetch the DevEUI from the LoPy4 board. This is [explained in the LoPy4 documentation](#). Each sensor node must be manually registered in the [TTN console](#) before it can be started. This assumes that you already have a TTN user account and have created an application in the user account (both need to be created otherwise). [In the TTN console create a new device](#) using the DevEUI value that was previously determined. After the registration of the device the two generated keys (AppEUI, AppKey) can be copied from the TTN console and must be pasted into the proper places in the source code of the program above. Please make sure that you choose for both keys the correct byte ordering (all are in MSB, i.e. in the same ordering as given in the TTN console). A detailed explanation of these steps is given [here](#). Then the program can be uploaded to the LoPy4 microcontroller. Note that the two constants (AppEUI, AppKey) must be changed in the source code for every new sensor node (the DevEUI is different for each node anyway).

Using the OTAA mode has the advantage over the ABP (activation by personalization) mode that during connection the session keys are newly created which improves security. Another advantage is that the packet counter is automatically reset to 0 both in the node and in the TTN application.

TTN Payload Decoding

Everytime a data packet is received by a TTN application a dedicated Javascript function is being called (Payload Decoder Function). This function can be used to decode the received byte string and to create proper Javascript objects or values that can directly be read by humans when looking at the incoming data packet. This is also useful to format the data in a specific way that can then be forwarded to an external application (e.g. a sensor data platform like [MyDevices](#) or [Thingspeak](#)). Such a forwarding can be configured in the TTN console in the “Integrations” tab. [TTN payload decoder](#) given here checks if a packet was received on LoRaWAN port 7 and then assumes that it consists of the 6 bytes encoded as described above. It creates the three Javascript objects ‘temperature’, ‘humidity’, and ‘vbattery’. Each object has two fields: ‘value’ holds the value and ‘uom’ gives the unit of measure. The source code can simply be copied and pasted into the ‘decoder’ tab in the TTN console after having selected the application. Choose the option ‘Custom’ in the ‘Payload Format’ field. Note that when you also want to handle other sensor nodes sending packets on different LoRaWAN ports, then the Payload Decoder Function can be extended after the end of the `if (port==7) { ... }` statement by adding `else if (port==8) { ... }` `else if (port==9) { ... }` etc.

The Things Network - OGC SensorWeb Integration

The presented Payload Decoder Function works also with the TTN-OGC SWE Integration for the [52° North Sensor Observation Service \(SOS\)](#). This software component can be downloaded from this [repository](#). It connects a TTN application with a running transactional [Sensor Observation Service 2.0.0 \(SOS\)](#). Data packets received from TTN are imported into the SOS. The SOS persistently stores sensor data from an arbitrary number of sensor nodes and can be queried for the most recent as well as for historic sensor data readings. The 52° North SOS comes with its own REST API and a nice web client allowing to browse the stored sensor data in a convenient way.

We are running an instance of the 52° North SOS and the TTN-OGC SWE Integration. The web client for this LoRaWAN sensor node can be accessed [on this page](#). Here is a screenshot showing the webclient:

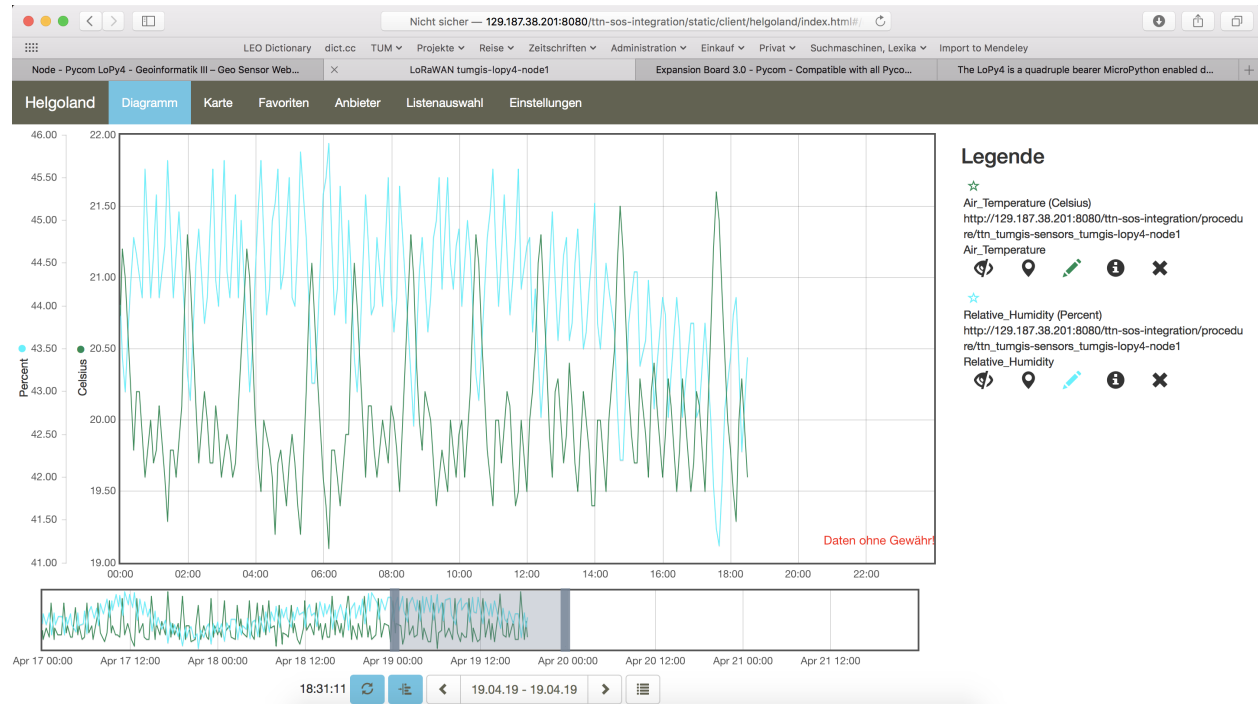


Fig. 27: Web client for data visualization

2.8.4 Code files

Listing 26: boot.py

```

1 from network import LoRa
2 import socket
3 import time
4 import ubinascii
5 import pycom
6 from machine import Pin
7 import dht22
8
9 # Initialise LoRa in LORAWAN mode.
10 # Please pick the region that matches where you are using the device:
11 # Asia = LoRa.AS923
12 # Australia = LoRa.AU915

```

(continues on next page)

(continued from previous page)

```

13 # Europe = LoRa.EU868
14 # United States = LoRa.US915
15 lora = LoRa(mode=LoRa.LORAWAN, region=LoRa.EU868)
16
17 # Create the OTAA authentication parameters:
18 # directly copy the values from the Things Network Console and replace the
19 # xxxx's and yyyy's by these values (do not prepend anything like '0x' or similar)
20 app_eui = ubinascii.unhexlify('xxxxxxxxxxxxxxxx')
21 app_key = ubinascii.unhexlify('yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy')
22
23 print("Initializing DHT22 Sensor... ",end='')
24 dht = dht22.device(Pin.exp_board.G22)
25 print("ready!\n")
26
27 pycom.heartbeat(False)
28 pycom.rgbled(0xFF0000)
29
30 # join a network using OTAA (Over the Air Activation)
31 lora.join(activation=LoRa.OTAA, auth=(app_eui, app_key), timeout=0)
32
33 # wait until the module has joined the network
34 while not lora.has_joined():
35     time.sleep(2.5)
36     print('Not yet joined...')
37
38 pycom.rgbled(0x00FF00)
39 time.sleep(1)
40
41 # create a LoRa socket
42 s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
43
44 # set the LoRaWAN data rate
45 s.setsockopt(socket.SOL_LORA, socket.SO_DR, 5)
46
47 # set the LoRaWAN port number for transmitted packets
48 s.bind(7)
49
50 pycom.heartbeat(True)

```

Listing 27: main.py

```

1 from network import LoRa
2 import socket
3 import time
4 import ubinascii
5 import pycom
6 from machine import Pin
7 import dht22
8
9 while (True):
10     pycom.heartbeat(False)
11     pycom.rgbled(0x800000)
12
13     # start a new measurement (taking 2 seconds) to ensure that the next
14     # retrieval of readings has current values
15     dht.trigger()

```

(continues on next page)

(continued from previous page)

```

16     time.sleep(0.2)
17
18     # now start a 2nd measurements which - according to the DHT22 datasheet -
19     # delivers the measured values from the previous reading
20     hasreading=False
21     numtrials=0
22
23     while(not hasreading):
24         hasreading=dht.trigger()
25         numtrials=numtrials+1
26         if hasreading:
27             print("RH = {}% T = {}C".format(dht.humidity, dht.temperature))
28         else:
29             print(dht.status)
30
31     hum_msb=int(dht.humidity*100/256)
32     hum_lsb=int(dht.humidity*100%256)
33
34     tmp_int=int(dht.temperature*100)
35     # if temperature value is negative, then represent it by its 2's complement (16_
↪bit)
36     if (tmp_int<0):
37         tmp_int=65536+tmp_int
38
39     tmp_msb=int(tmp_int/256)
40     tmp_lsb=int(tmp_int%256)
41     print("RH = {} {} T = {} {}".format(hum_msb, hum_lsb, tmp_msb, tmp_lsb))
42     pycom.rgbled(0x000040)
43
44     # make the socket blocking
45     # (waits for the data to be sent and for the 2 receive windows to expire)
46     s.setblocking(True)
47
48     # send some data
49     s.send(bytes([tmp_msb, tmp_lsb, hum_msb, hum_lsb, 0, 0]))
50
51     # make the socket non-blocking
52     # (because if there's no data received it will block forever...)
53     s.setblocking(False)
54
55     # get any data received (if any...)
56     data = s.recv(64)
57     print(data)
58
59     pycom.heartbeat(True)
60     # wait for such a time period that we have one measurement every 300 seconds
61     time.sleep(300-numtrials*4-3)

```

Listing 28: dht22.py

```

1  # dht22.py
2  #
3  # Class file for accessing the DHT22 temperature and humidity sensor using a WiPy 2.0.
4  #
5  # 2018 - Erik de Lange

```

(continues on next page)

(continued from previous page)

```

7
8 from machine import Pin
9
10 import pycom
11 import time
12
13 class device:
14
15     def __init__(self, pin):
16         self.temperature = None
17         self.humidity = None
18         self.status = "NoConversionStartedError"
19         self.pin = Pin(pin, mode=Pin.OPEN_DRAIN)
20
21     def trigger(self):
22         self.pin(1)
23         time.sleep(2) # enforce two second read interval
24
25         self.pin(0) # send start signal (1ms low).
26         time.sleep_ms(1)
27
28         pulses = pycom.pulses_get(self.pin, 100) # capture communication
29
30         self.pin.init(Pin.OPEN_DRAIN)
31
32         if len(pulses) != 82: # 40 data bit plus one acknowledge expected
33             self.status = "ReadError - received {} only pulses".format(len(pulses))
34             return False
35
36         bits = []
37
38         for level, duration in pulses[1:]:
39             if level == 1:
40                 bits.append(0 if duration < 50 else 1) # convert to 0 or 1
41
42         data = []
43
44         for n in range(5):
45             byte = 0
46             for i in range(8): # shift 8 bits into a byte
47                 byte <<= 1
48                 byte += bits[n * 8 + i]
49             data.append(byte)
50
51         int_rh, dec_rh, int_t, dec_t, csum = data
52
53         if ((int_rh + dec_rh + int_t + dec_t) & 0xFF) != csum:
54             self.status = "Checksum Error"
55             return False
56
57         self.humidity = ((int_rh * 256) + dec_rh) / 10
58         self.temperature = (((int_t & 0x7F) * 256) + dec_t) / 10
59         if (int_t & 0x80) > 0:
60             self.temperature *= -1
61
62         self.status = "OK"
63         return True

```

(continues on next page)

(continued from previous page)

```

64
65
66 if __name__ == "__main__":
67     dht = device(Pin.exp_board.G22)
68
69     for _ in range(5):
70         if dht.trigger() == True:
71             print("RH = {}% T = {}C".format(dht.humidity, dht.temperature))
72         else:
73             print(dht.status)

```

Listing 29: TTN payload decoder

```

1 function Decoder (bytes, port) {
2     var result = {};
3     var transformers = {};
4
5     if (port==7) {
6         transformers = {
7             'temperature': function transform (bytes) {
8                 value=bytes[0]*256 + bytes[1];
9                 if (value>=32768) value=value-65536;
10                return value/100.0;
11            },
12            'humidity': function transform (bytes) {
13                return (bytes[0]*256 + bytes[1])/100.0;
14            },
15            'vbattery': function transform (bytes) {
16                return (bytes[0]*256 + bytes[1])/100.0;
17            },
18        }
19
20        result['temperature'] = {
21            value: transformers['temperature'](bytes.slice(0, 2)),
22            uom: 'Celsius',
23        }
24
25        result['humidity'] = {
26            value: transformers['humidity'](bytes.slice(2, 4)),
27            uom: 'Percent',
28        }
29
30        result['vbattery'] = {
31            value: transformers['vbattery'](bytes.slice(4, 6)),
32            uom: 'Volt',
33        }
34    }
35
36    return result;
37 }

```

2.8.5 References

- [Pycom LoPy4 product homepage](#)
- [LoPy4 specification document](#)

- LoPy4 pinout specification
- LoPy4 online documentation (incl. description of software libraries)
- MicroPython language and library reference
- LoPy4 Getting Started (hardware & software setup, installation of Pymakr IDE)

On the Expansion Board 3.0

- Pycom Expansion Board 3.0 product homepage
- Expansion Board 3.0 documentation (pinout, datasheet)

On the DHT22 sensor

- MicroPython library for the DHT22 sensor
- DHT22 datasheet

2.9 Seeeduino LoRaWAN

2.9.1 Hardware

Microcontroller

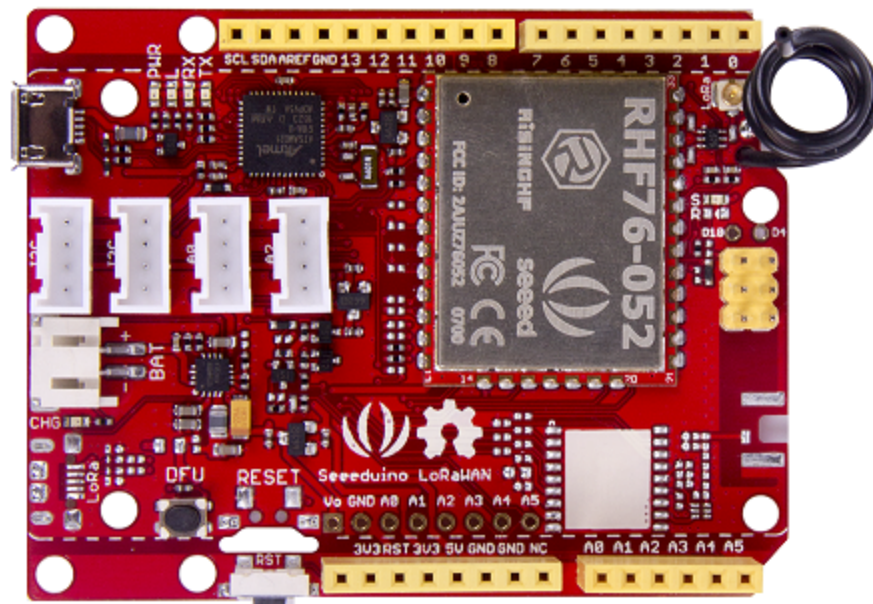


Fig. 28: Seeeduino LoRaWAN microcontroller from Seeed Studio.

The Seeeduino LoRaWAN module is operated by the 32bit microcontroller ATSAMD21G18 (ARM® Cortex®-M0+) running at 48MHz. It has 256 KB flash memory (to store the program code) and 32 KB of RAM (to store variables, status information, and buffers). The operating voltage of the board is 3.3V (this is important when attaching sensors

and other peripherals; they also must operate on 3.3V). The board offers 20 general purpose digital input/output pins (20 GPIOs), 6 analog input pins (with 12bit analog digital converters (ADC)), 1 analog output pin (with 10bit digital analog converter (DAC)), 2 serial ports (2 programmable Universal Asynchronous Receiver and Transmitters, UARTs). The board comes with an embedded lithium battery management chip and status indicator led, which allows to directly connect a 3.7V LiPo rechargeable battery that will be automatically recharged when the board is powered over its USB connector. The battery voltage level can be queried from analog input A4, the charging status (charging, full) from analog input A5. The Seeeduino LoRaWAN (without GPS module) is available in German shops for around 37 €.

The LoRa transmitter and receiver is encapsulated within an RHF76-052AM module from the Chinese company RisingHF. The RF module contains its own microcontroller, which implements the LoRaWAN protocol. The module is connected via the serial interface to the ATSAM21G18 microcontroller and can be controlled by sending so-called ‘AT’ commands. The implemented LoRaWAN functionality is compatible with LoRaWAN Class A/C. The explanation of all supported commands as well as a number of examples on how to use the Seeeduino LoRaWAN are given on the [Seeeduino LoRaWAN Wiki](#).

The board has 4 on-board Grove connectors. ‘Grove’ is a [framework developed by the company Sreed Studio](#) standardizing the connectors, operating voltages, and pin configurations for attaching peripherals like [sensors, actuators, and displays](#) to microcontrollers. Note that the Grove modules need to be able to operate (also) on 3.3V (instead of only with 5V), because the Seeeduino LoRaWAN board only provides 3.3V to the Grove connectors. Important hint: if you want to use the Grove ports, make sure to include the command “digitalWrite(38, HIGH)” in the setup() routine of your program. A low level on that pin deactivates the power supply of the four Grove ports.

The board has also the typical [Arduino UNO connectors](#) allowing to attach so-called [Arduino shields](#) (however, please note that the shields must be working with 3.3V; the normal operating voltage for the Arduino UNO microcontroller and its shields is 5V).

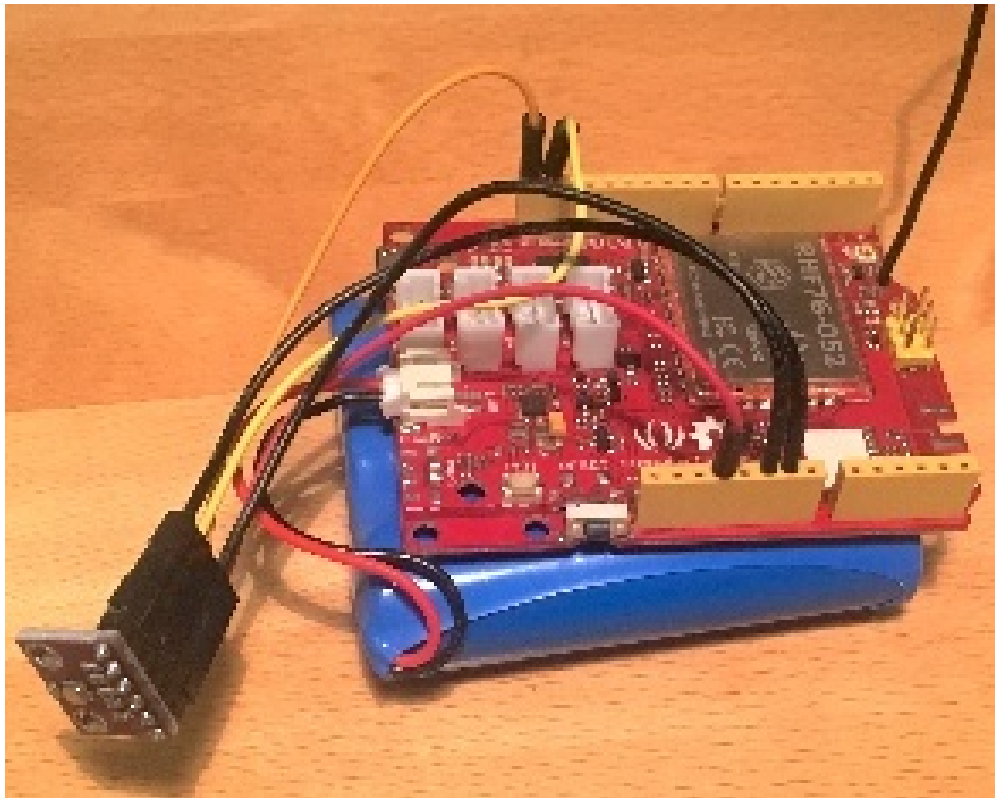


Fig. 29: The Seeeduino LoRaWAN GPS microcontroller with a 6600 mAh lithium polymer (LiPo) battery (bottom), and an attached BME280 temperature / humidity / barometer sensor module.

Sensor

We attached a [Bosch BME280 sensor](#) module to the extension connectors of the microcontroller board using 5 wires. The employed BME280 sensor board is a cheap [no-name product](#). VCC and GND are connected to 3.3V and GND of the microcontroller board respectively. SCL and SDA from the sensor board are connected to SCL and SDA of the microcontroller board. SDO from the sensor board is also connected to GND of the microcontroller; it selects 0x76 as the I2C device address (a high level, i.e. 3.3V, would set the device address to 0x77 - this is relevant, if two sensor modules should be operated on the same I2C bus). Note that there is also a Seeed Grove BME280 module available which alternatively can be used and connected to the first I2C Grove connector of the Seeeduino LoRaWAN board. The BME280 measures temperature in the range -40 - 85 °C, with $\pm 1.0^{\circ}\text{C}$ accuracy; 0% - 100% relative humidity with $\pm 3\%$ accuracy; and atmospheric pressure in the range 300 - 1100 hPa (1 hPa= one hundred Pa) with ± 1.0 hPa accuracy. It offers the two interface standards I2C and SPI (we are using I2C here and the default I2C address 0x76). The atmospheric pressure changes with altitude, hence, the BME280 can also be used to measure the approximate altitude of a place.

2.9.2 Software

The sensor node has been programmed using the [Arduino IDE](#). Please note, that in the Arduino framework a program is called a ‘Sketch’.

In order to support the “Seeeduino LoRaWAN” board with the Arduino IDE, make sure to have installed the package “Seeed SAMD boards by Seeed Studio” in version 1.3.0 using the [board manager](#) in the Arduino IDE. This is also explained on a [dedicated webpage](#) from Seeed Studio. The sketch requires the software libraries “RTCZero”, “Arduino_BME280”, “Adafruit_Sensor”, “Wire”, and “LoRaWAN”. The first three have to be installed using the [library manager](#) of the Arduino IDE, the fourth library is already installed with the Arduino IDE and the latter library comes with the “Seeeduino LoRaWAN” board installation.

After the sketch has successfully established a connection to The Things Network it reports the air temperature, relative humidity, air pressure, altitude, and the voltage of a (possibly) attached LiPo battery every 5 minutes. All five values are being encoded in two byte integer values each and then sent as a 10 bytes data packet to the respective TTN application using LoRaWAN port 33. Please note, that LoRaWAN messages can be addressed to ports 1-255 (port 0 is reserved); these ports are similar to port numbers 0-65535 when using the Internet TCP/IP protocol. Voltage, pressure, altitude, and humidity values are always greater or equal to 0, but the temperature value can also become negative. Negative values are represented as a [two’s complement](#); this must be considered in the Payload Decoding Function used in The Things Network (see below).

In between two sensor readings the microcontroller, the LoRaWAN module, and the sensor module are going into deep sleep mode to save battery power. During LoRaWAN data transmission the device draws up to 65mA current. When in sleep mode the entire node only draws around 0.06 mA power. Hence, with a 6600 mAh 3.7V LiPo battery and the current version of the sketch the system should be able to run for many years before recharging (not taking into account the self-discharging rate of the battery).

The source code is provided in the following section [Arduino Sketch for Seeeduino LoRaWAN sensor node](#)

2.9.3 Services

The services used for this sensor-node are:

- [TheThingsNetwork](#) service for LoRaWAN network service.
- [TheThingsNetwork - OGC SensorWeb](#) integration for uploading LoRaWAN sensor data into OGC infrastructure.

Registration of the sensor node with The Things Network (TTN)

The LoRaWAN protocol makes use of a number of different identifiers, addresses, keys, etc. These are required to unambiguously identify devices, applications, as well as to encrypt and decrypt messages. The names and meanings are nicely explained on a dedicated [TTN web page](#).

The sketch given above connects the sensor node with The Things Network (TTN) using the Over-the-Air-Activation (OTAA) mode. In this mode, we use the three keys AppEUI, DevEUI, AppKey. The DevEUI should normally be delivered with the sensor node by the manufacturer. However, it seems that there is no explicit DevEUI provided with the Seeeduino LoRaWAN module. Therefore, it has to be generated automatically together with the other two keys using the TTN console. Each sensor node must be manually registered in the [TTN console](#) before it can be started. This assumes that you already have a TTN user account (which needs to be created otherwise). In the TTN console [create a new device](#) with also the DevEUI being automatically generated. After the registration of the device the respective keys (AppEUI, DevEUI, AppKey) can be copied from the TTN console and must be pasted into the proper places in the source code of the sketch above. Please make sure that you choose for each of the three keys are in the correct byte ordering (all are in MSB, i.e. in the same ordering as given in the TTN console). A detailed explanation of these steps is given [on this page](#). Then the sketch can be compiled and uploaded to the Seeeduino LoRaWAN microcontroller. Note that the three constants (AppEUI, DevEUI, AppKey) must be changed in the source code for every new sensor node.

Using the OTAA mode has the advantage over the ABP (activation by personalization) mode that during connection the session keys are newly created which improves security. Another advantage is that the packet counter is automatically reset to 0 both in the node and in the TTN application.

TTN Payload Decoding

Everytime a data packet is received by a TTN application a dedicated Javascript function is being called (Payload Decoder Function). This function can be used to decode the received byte string and to create proper Javascript objects or values that can directly be read by humans when looking at the incoming data packet. This is also useful to format the data in a specific way that can then be forwarded to an external application (e.g. a sensor data platform like [MyDevices](#) or [Thingspeak](#)).

Such a forwarding can be configured in the TTN console in the “Integrations” tab. [TTN payload decoder for Seeeduino LoRaWAN sensor node](#) given here checks if a packet was received on LoRaWAN port 33 and then assumes that it consists of the 10 bytes encoded as described above. It creates the five Javascript objects ‘temperature’, ‘humidity’, ‘pressure’, ‘altitude’, and ‘vbattery’. Each object has two fields: ‘value’ holds the value and ‘uom’ gives the unit of measure. The source code can simply be copied and pasted into the ‘decoder’ tab in the TTN console after having selected the application. Choose the option ‘Custom’ in the ‘Payload Format’ field. Note that when you also want to handle other sensor nodes sending packets on different LoRaWAN ports, then the Payload Decoder Function can be extended after the end of the `if (port==33) { ... }` statement by adding `else if (port==7) { ... }` `else if (port==8) { ... }` etc.

The Things Network - OGC SensorWeb Integration

The presented Payload Decoder Function works also with the TTN-OGC SWE Integration for the [52° North Sensor Observation Service \(SOS\)](#). This software component can be downloaded from this [repository](#). It connects a TTN application with a running transactional [Sensor Observation Service 2.0.0 \(SOS\)](#). Data packets received from TTN are imported into the SOS. The SOS persistently stores sensor data from an arbitrary number of sensor nodes and can be queried for the most recent as well as for historic sensor data readings. The 52° North SOS comes with its own REST API and a nice web client allowing to browse the stored sensor data in a convenient way.

We are running an instance of the 52° North SOS and the TTN-OGC SWE Integration. The web client for this LoRaWAN sensor node can be accessed [on this webpage](#). Here is a screenshot showing the webclient:

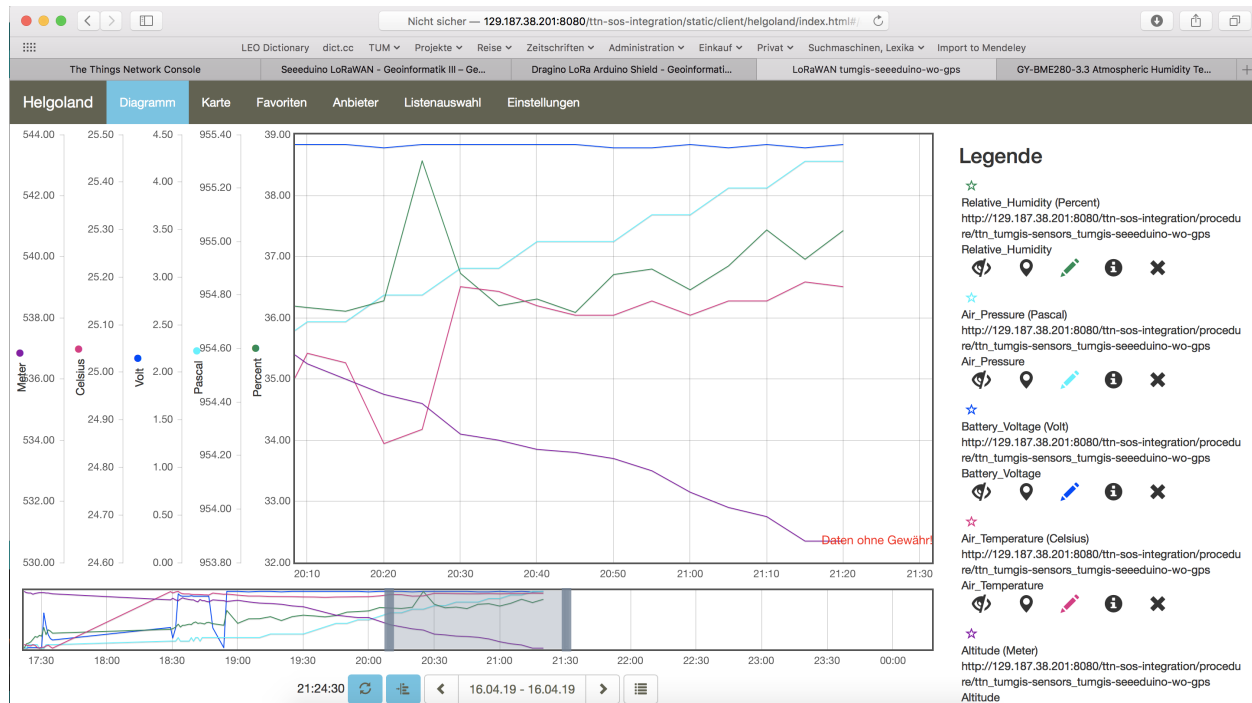


Fig. 30: Web client for data visualization

2.9.4 Code files

Listing 30: Arduino Sketch for Seeeduino LoRaWAN sensor node

```

1  /*****
2   * Arduino Sketch for a LoRaWAN sensor node that is registered with
3   * 'The Things Network' (TTN) www.thethingsnetwork.org
4   *
5   * Filename: Seeeduino_LoRaWAN_GPS_BME280_OTAA_Sleep_Adafruit_V2.ino
6   *
7   * Author: Thomas H. Kolbe, thomas.kolbe@tum.de
8   * Version: 1.0.1
9   * Last update: 2019-04-17
10  *
11  * This sketch works with a Seeeduino LoRaWAN microcontroller board (with or
12  * without embedded GPS module). See http://wiki.seeedstudio.com/Seeeduino_LoRAWAN/
13  * It requires a Seeed Grove BME280 air temperature, relative humidity,
14  * and air pressure sensor module attached to the I2C Grove connector of
15  * the microcontroller board. The current configuration assumes that
16  * the BME280 is configured to I2C device address 0x76 (default).
17  * The sketch makes a connection to The Things Network (TTN) using
18  * LoRaWAN in OTAA mode. It then sends a data packet of 10 bytes to
19  * LoRaWAN port 33 around every 5 minutes. The packet contains the
20  * following 5 integer values (16 bit, most significant byte (MSB) first):
21  * 1. temperature in Celsius (signed, multiplied by 100)
22  * 2. relative humidity in percent (unsigned, multiplied by 100)
23  * 3. air pressure in Pascal (unsigned, divided by 10)
24  * 4. current altitude in Meters (unsigned, multiplied by 10)
25  * 5. battery voltage in millivolt (unsigned)
  
```

(continues on next page)

(continued from previous page)

```

26  * These values have to be decoded by the LoRaWAN network controller
27  * using a proper "payload decoder function" written in Javascript.
28  *
29  * Note that when the board is powered over the USB connector and
30  * no battery is connected, the measured battery voltage is incorrect.
31  *
32  * If the board shall be running on a lithium polymer (LiPo) battery,
33  * it is recommended to remove the green power LED from the board or
34  * to cut the connection between the LED and the resistor lying above
35  * of it as the LED constantly draws around 8mW of power. In order to
36  * save energy the sketch puts the GPS module on the board to standby
37  * mode right from the beginning. After each measurement and data transfer
38  * the LoRaWAN module and the sensor is put to standby mode, too, and the
39  * microcontroller goes into deep sleep mode. All components require
40  * a total current of around 0.34mA during sleep mode and up to 65mA
41  * during LoRa transmission for the board version with GPS. The board
42  * version without GPS only requires 0.06mA during sleep mode. Since the
43  * entire system is mostly sleeping, the GPS board should be running
44  * around 2 years on a 6600mAh LiPo battery before recharging
45  * (6600mAh / 0.34mA / 24 = 808 days). The non GPS board version should
46  * even run for more than 10 years...
47  *
48  * This code is based on example code given on the Seeeduino LoRaWAN
49  * wiki page. It utilizes the Open Source libraries "Adafruit_BME280"
50  * and "Adafruit_Sensor" provided by the company Adafruit and the
51  * library "LoRaWan.h" provided by Seeed Studio.
52  *****/
53
54  #include <RTCZero.h>
55  #include <LoRaWan.h>
56  #include <Wire.h>
57  #include <Adafruit_Sensor.h>
58  #include <Adafruit_BME280.h>
59
60  // Keep the following line, if the board is a Seeeduino LoRaWAN with GPS,
61  // otherwise comment the line out
62
63  // #define HAS_GPS 1
64
65  #define BME280_ADDRESS      (0x76)    // I2C device address of the BME280 sensor
66
67  // The barometer of the BME280 can also be used to estimate the current
68  // altitude of the device, if the air pressure at sea level (NN) is known.
69  // The following value has to be set to the current air pressure at NN (in hPa)
70  // in order to give reasonable altitude estimations. Note that this value is
71  // slowly changing over time. For Munich the current value can be obtained
72  // from https://www.meteo.physik.uni-muenchen.de/mesomikro/stadt/messung.php
73
74  #define SEALEVELPRESSURE_HPA (1017.8)
75
76  Adafruit_BME280 bme280;
77
78  RTCZero rtc;
79
80  unsigned char data[10];           // buffer for the LoRaWAN data packet to be_
    ↪ transferred
81  char buffer[256];                // buffer for text messages received from the_
    ↪ LoRaWAN module for display

```

(continues on next page)

(continued from previous page)

```

82
83
84 void setup(void)
85 {
86     digitalWrite(38, HIGH);           // Provide power to the 4 Grove connectors of
87     ↪ the board
88
89     for(int i = 0; i < 26; i++)        // Set all pins to HIGH to save power
90     ↪ (reduces the
91     {                                  // current drawn during deep sleep by around
92     ↪ 0.7mA).
93         if (i!=13) {                 // Don't switch on the onboard user LED (pin
94     ↪ 13).
95             pinMode(i, OUTPUT);
96             digitalWrite(i, HIGH);
97         }
98     }
99
100     delay(5000);                      // Wait 5 secs after reset/booting to give
101     ↪ time for potential upload
102
103     // of a new sketch (sketches cannot be
104     ↪ uploaded when in sleep mode)
105     SerialUSB.begin(115200);          // Initialize USB/serial connection
106     delay(500);
107     // while(!SerialUSB);
108     SerialUSB.println("Seeeduino LoRaWAN board started!");
109
110     if(!bme280.begin(BME280_ADDRESS)) { // Initialize the BME280 sensor module
111     SerialUSB.println("BME280 device error!");
112     }
113
114     // Set the BME280 to a very low power operation mode (c.f. chapter 3.5
115     // "Recommended modes of operation" in the BME280 datasheet. See
116     // https://cdn-shop.adafruit.com/datasheets/BST-BME280_DS001-10.pdf );
117     // proper values can only be queried every 60s
118     bme280.setSampling(Adafruit_BME280::MODE_FORCED,
119         Adafruit_BME280::SAMPLING_X16, // temperature
120         Adafruit_BME280::SAMPLING_X16, // pressure
121         Adafruit_BME280::SAMPLING_X16, // humidity
122         Adafruit_BME280::FILTER_OFF );
123
124     // nrgSave.begin(WAKE_RTC_ALARM);
125     // rtc.begin(TIME_H24);
126
127 #ifdef HAS_GPS
128     Serial.begin(9600);                // Initialize serial connection to the GPS
129     ↪ module
130     delay(500);
131     Serial.write("$PMTK161,0*28\r\n"); // Switch GPS module to standby mode as we don
132     ↪ 't use it in this sketch
133 #endif
134
135     lora.init();                       // Initialize the LoRaWAN module
136
137     memset(buffer, 0, 256);            // clear text buffer
138     lora.getVersion(buffer, 256, 1);
139     memset(buffer, 0, 256);            // We call getVersion() two times, because
140     ↪ after a reset the LoRaWAN module can be

```

(continues on next page)

(continued from previous page)

```

131   lora.getVersion(buffer, 256, 1);    // in sleep mode and then the first call only.
↳wakes it up and will not be performed.
132   SerialUSB.print(buffer);
133
134   memset(buffer, 0, 256);
135   lora.getId(buffer, 256, 1);
136   SerialUSB.print(buffer);
137
138   // The following three constants (AppEUI, DevEUI, AppKey) must be changed
139   // for every new sensor node. We are using the LoRaWAN OTAA mode (over the
140   // air activation). Each sensor node must be manually registered in the
141   // TTN console at https://console.thethingsnetwork.org before it can be
142   // started. In the TTN console create a new device with the DevEUI also
143   // being automatically generated. After the registration of the device the
144   // three values can be copied from the TTN console. A detailed explanation
145   // of these steps is given in
146   // https://learn.adafruit.com/the-things-network-for-feather?view=all
147
148   // The EUIs and the AppKey must be given in big-endian format, i.e. the
149   // most-significant-byte comes first (as displayed in the TTN console).
150   // For TTN issued AppEUIs the first bytes should be 0x70, 0xB3, 0xD5.
151
152   // void setId(char *DevAddr, char *DevEUI, char *AppEUI);
153   lora.setId(NULL, "xxxxxxxxxxxxxxxx", "yyyyyyyyyyyyyyyy");
154
155   // setKey(char *NwkSKey, char *AppSKey, char *AppKey);
156   lora.setKey(NULL, NULL, "zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz");
157
158   lora.setDeciveMode(LWOTAA);           // select OTAA join mode (note that
↳setDeciveMode is not a typo; it is misspelled in the library)
159   // lora.setDataRate(DR5, EU868);       // SF7, 125 kbps (highest data rate)
160   lora.setDataRate(DR3, EU868);         // SF9, 125 kbps (medium data rate and
↳range)
161   // lora.setDataRate(DR0, EU868);       // SF12, 125 kbps (lowest data rate,
↳highest max. distance)
162
163   // lora.setAdaptiveDataRate(false);
164   lora.setAdaptiveDataRate(true);       // automatically adapt the data rate
165
166   lora.setChannel(0, 868.1);
167   lora.setChannel(1, 868.3);
168   lora.setChannel(2, 868.5);
169   lora.setChannel(3, 867.1);
170   lora.setChannel(4, 867.3);
171   lora.setChannel(5, 867.5);
172   lora.setChannel(6, 867.7);
173   lora.setChannel(7, 867.9);
174
175   // The following two commands can be left commented out;
176   // TTN works with the default values. (It also works when
177   // uncommenting the commands, though.)
178   // lora.setReceiceWindowFirst(0, 868.1);
179   // lora.setReceiceWindowSecond(869.525, DR0);
180
181   lora.setDutyCycle(false);             // for debugging purposes only - should
↳normally be activated
182   lora.setJoinDutyCycle(false);         // for debugging purposes only - should
↳normally be activated

```

(continues on next page)

(continued from previous page)

```

183     lora.setPower(14);                // LoRa transceiver power (14 is the
184     ↪maximum for the 868 MHz band)
185
186     // while(!lora.setOTAAJoin(JOIN));
187     while(!lora.setOTAAJoin(JOIN,20)); // wait until the node has successfully
188     ↪joined TTN
189
190     lora.setPort(33);                // all data packets are sent to LoRaWAN
191     ↪port 33
192 }
193
194 void loop(void)
195 {
196     bool result = false;
197     float temperature, altitude, pressure, humidity;
198     int16_t int16_temperature, int16_humidity, int16_pressure, int16_altitude, int16_
199     ↪vbat;
200
201     bme280.takeForcedMeasurement(); // wake the sensor up for the next readings
202
203     //get and print temperatures
204     SerialUSB.print("Temp: ");
205     SerialUSB.print(temperature = bme280.readTemperature());
206     SerialUSB.print("C ");
207
208     //get and print atmospheric pressure data
209     SerialUSB.print("Pressure: ");
210     SerialUSB.print(pressure = bme280.readPressure());
211     SerialUSB.print("Pa ");
212
213     //get and print altitude data
214     SerialUSB.print("Altitude: ");
215     SerialUSB.print(altitude = bme280.readAltitude(SEALEVELPRESSURE_HPA));
216     SerialUSB.print("m ");
217
218     //get and print humidity data
219     SerialUSB.print("Humidity: ");
220     SerialUSB.print(humidity = bme280.readHumidity());
221     SerialUSB.print("% ");
222
223     //get and print battery voltage
224     SerialUSB.print("VBat: ");
225     SerialUSB.print(int16_vbat=lora.getBatteryVoltage());
226     SerialUSB.println("mV");
227
228     int16_temperature = temperature*100.0;
229     int16_humidity = humidity*100.0;
230     int16_pressure = pressure/10.0;
231     int16_altitude = altitude*10.0;
232
233     data[0] = (byte) (int16_temperature >> 8);
234     data[1] = (byte) (int16_temperature & 0x00FF);
235     data[2] = (byte) (int16_humidity >> 8);
236     data[3] = (byte) (int16_humidity & 0x00FF);
237     data[4] = (byte) (int16_pressure >> 8);
238     data[5] = (byte) (int16_pressure & 0x00FF);

```

(continues on next page)

(continued from previous page)

```

236     data[6] = (byte) (int16_altitude >> 8);
237     data[7] = (byte) (int16_altitude & 0x00FF);
238     data[8] = (byte) (int16_vbat >> 8);
239     data[9] = (byte) (int16_vbat & 0x00FF);
240
241     result = lora.transferPacket(data, 10, 5);    // send the data packet (10 bytes)
↳with a default timeout of 5 secs
242
243     if(result)
244     {
245         short length;
246         short rssi;
247
248         memset(buffer, 0, 256);
249         length = lora.receivePacket(buffer, 256, &rssi);
250
251         if(length)
252         {
253             SerialUSB.print("Length is: ");
254             SerialUSB.println(length);
255             SerialUSB.print("RSSI is: ");
256             SerialUSB.println(rssi);
257             SerialUSB.print("Data is: ");
258             for(unsigned char i = 0; i < length; i++)
259             {
260                 SerialUSB.print("0x");
261                 SerialUSB.print(buffer[i], HEX);
262                 SerialUSB.print(" ");
263             }
264             SerialUSB.println();
265         }
266     }
267
268     lora.setDeviceLowPower();    // bring the LoRaWAN module to sleep mode
269     doSleep((5*60-8)*1000);    // deep sleep for 292 secs (+ 3 secs transmission
↳time + 5 secs timeout = 300 secs period)
270     lora.setPort(33);          // send some command to wake up the LoRaWAN module
↳again
271 }
272
273 // The following function implements deep sleep waiting. When being called the
274 // CPU goes into deep sleep mode (for power saving). It is woken up again by
275 // the CPU-internal real time clock (RTC) after the configured time.
276 //
277 // A similar function would also be available in the standard "ArduinoLowPower"
↳library.
278 // However, in order to be able to use that library with the Seeeduino LoRaWAN board,
279 // four files in the package "Seeed SAMD boards by Seeed Studio Version 1.3.0" that is
280 // installed using the Arduino IDE board manager need to be patched. The reason is
↳that
281 // Seeed Studio have not updated their files to a recent Arduino SAMD version yet
282 // and the official "ArduinoLowPower" library provided by the Arduino foundation is
283 // referring to some missing functions. For further information see here:
284 // https://forum.arduino.cc/index.php?topic=603900.0 and here:
285 // https://github.com/arduino/ArduinoCore-samd/commit/
↳b9ac48c782ca4b82ffd7e65bf2c956152386d82b
286

```

(continues on next page)

(continued from previous page)

```

287 void doSleep(uint32_t millis) {
288     if (!rtc.isConfigured()) { // if called for the first time,
289         rtc.begin(false); // then initialize the real time clock (RTC)
290     }
291
292     uint32_t now = rtc.getEpoch();
293     rtc.setAlarmEpoch(now + millis/1000);
294     rtc.enableAlarm(rtc.MATCH_HHMMSS);
295
296     rtc.standbyMode(); // bring CPU into deep sleep mode (until woken up_
    ↳by the RTC)
297 }

```

Listing 31: TTN payload decoder for Seeeduno LoRaWAN sensor node

```

1  function Decoder (bytes, port) {
2      var result = {};
3      var transformers = {};
4
5      if (port==33) {
6          transformers = {
7              'temperature': function transform (bytes) {
8                  value=bytes[0]*256 + bytes[1];
9                  if (value>=32768) value=value-65536;
10                 return value/100.0;
11             },
12             'humidity': function transform (bytes) {
13                 return (bytes[0]*256 + bytes[1])/100.0;
14             },
15             'pressure': function transform (bytes) {
16                 return (bytes[0]*256 + bytes[1])/10.0;
17             },
18             'altitude': function transform (bytes) {
19                 return (bytes[0]*256 + bytes[1])/10.0;
20             },
21             'vbattery': function transform (bytes) {
22                 return (bytes[0]*256 + bytes[1])/1000.0;
23             }
24         }
25
26         result['temperature'] = {
27             value: transformers['temperature'](bytes.slice(0, 2)),
28             uom: 'Celsius',
29         }
30
31         result['humidity'] = {
32             value: transformers['humidity'](bytes.slice(2, 4)),
33             uom: 'Percent',
34         }
35
36         result['pressure'] = {
37             value: transformers['pressure'](bytes.slice(4, 6)),
38             uom: 'hPa',
39         }
40
41         result['altitude'] = {

```

(continues on next page)

(continued from previous page)

```

42     value: transformers['altitude'](bytes.slice(6, 8)),
43     uom: 'Meter',
44   }
45
46   result['vbattery'] = {
47     value: transformers['vbattery'](bytes.slice(8, 10)),
48     uom: 'Volt',
49   }
50 }
51
52 return result;
53 }
```

2.9.5 References

- [Seeeduino LoRaWAN microcontroller](#)
- [Seeeduino LoRaWAN Wiki with instructions](#)
- [A short presentation on LoRaWAN basics and using the Seeeduino LoRaWAN board](#)

On the RisingHF RHF76-052AM LoRaWAN module

- [Product homepage](#)
- [LoRaWAN Class A/C AT Command Specification](#)

On the Bosch BME280 sensor

- [Product details](#)
- [Datasheet](#)
- [Adafruit_BME280 library for the Arduino platform](#)
- [Adafruit_Sensor library for the Arduino platform](#)
- [Instructions from Adafruit Industries how to use the BME280 and the library](#)

2.10 Seeeduino LoRaWAN with GPS

2.10.1 Hardware

Micro-controller

The Seeeduino LoRaWAN module is operated by the 32bit microcontroller ATSAMD21G18 (ARM® Cortex®-M0+) running at 48MHz. It has 256 KB flash memory (to store the program code) and 32 KB of RAM (to store variables, status information, and buffers). The operating voltage of the board is 3.3V (this is important when attaching sensors and other peripherals; they also must operate on 3.3V). The board offers 20 general purpose digital input/output pins (20 GPIOs), 6 analog input pins (with 12bit analog digital converters (ADC)), 1 analog output pin (with 10bit digital analog converter (DAC)), 2 serial ports (2 programmable Universal Asynchronous Receiver and Transmitters, UARTs). The board comes with an embedded lithium battery management chip and status indicator led, which allows to directly connect a 3.7V LiPo rechargeable battery that will be automatically recharged when the board is powered over its USB connector. The battery voltage level can be queried from analog input A4, the charging status (charging, full) from analog input A5. There is an on-board [L70 GPS receiver module from the company Quectel Wireless Solutions](#) and a small chip antenna. The Seeeduino LoRaWAN GPS module is available in German shops from around 37 € to 45 €.



Fig. 31: Seeduino LoRaWAN GPS microcontroller from Seed Studio.

The LoRa transmitter and receiver is encapsulated within an RHF76-052AM module from the Chinese company RisingHF. The RF module contains its own microcontroller, which implements the LoRaWAN protocol. The module is connected via the serial interface to the ATSAM21G18 microcontroller and can be controlled by sending so-called ‘AT’ commands. The implemented LoRaWAN functionality is compatible with LoRaWAN Class A/C. The explanation of all supported commands as well as a number of examples on how to use the Seeeduino LoRaWAN are given on the [Seeeduino LoRaWAN Wiki](#).

The board has 4 on-board Grove connectors. ‘Grove’ is a [framework developed by the company Seeed Studio](#) standardizing the connectors, operating voltages, and pin configurations for attaching peripherals like [sensors](#), [actuators](#), and [displays](#) to microcontrollers. The board has also the typical [Arduino UNO connectors](#) allowing to attach so-called [Arduino shields](#) (however, please note that the shields must be working with 3.3V; the normal operating voltage for the Arduino UNO microcontroller and its shields is 5V).

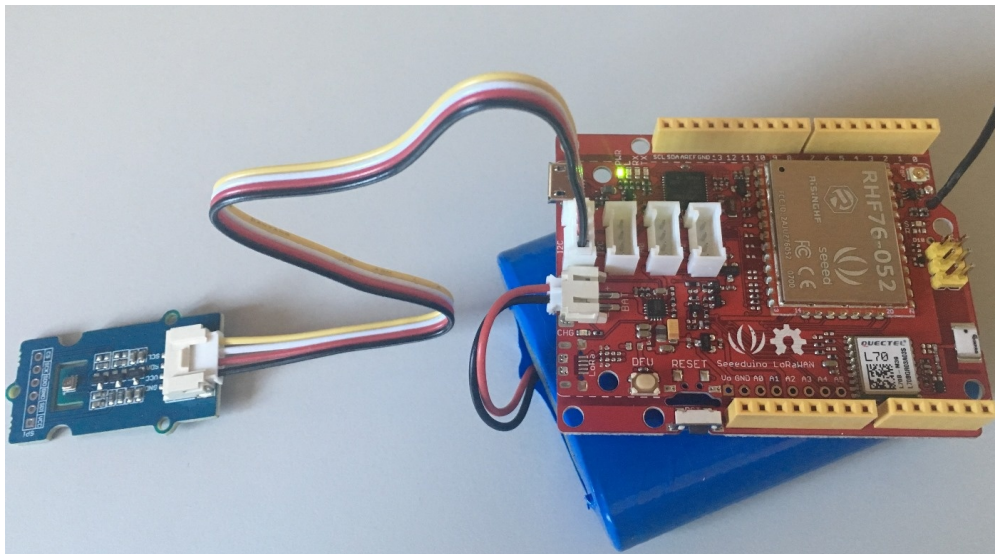


Fig. 32: The Seeeduino LoRaWAN GPS microcontroller with a 6600 mAh lithium polymer (LiPo) battery (bottom), and an attached [BME280 temperature / humidity / barometer sensor module](#).

Sensor

We attached a [Seeed BME280 Grove module](#) with a [Bosch BME280 sensor](#) to the first I2C Grove connector of the Seeeduino LoRaWAN board. The BME280 measures temperature in the range -40 - 85 °C, with $\pm 1.0^{\circ}\text{C}$ accuracy; 0%-100% relative humidity with $\pm 3\%$ accuracy; and atmospheric pressure in the range 300 - 1100 hPa (1 hPa= one hundred Pa) with ± 1.0 hPa accuracy. It offers the two interface standards I2C and SPI (we are using I2C here and the default I2C address 0x76). The atmospheric pressure changes with altitude, hence, the BME280 can also be used to measure the approximate altitude of a place.

2.10.2 Software

The sensor node has been programmed using the [Arduino IDE](#). Please note, that in the Arduino framework a program is called a ‘Sketch’.

In order to support the “Seeeduino LoRaWAN” board with the Arduino IDE, make sure to have installed the package “Seeed SAMD boards by Seeed Studio” in version 1.3.0 using the [board manager](#) in the Arduino IDE. This is also explained on a [dedicated webpage](#) from Seeed Studio. The sketch requires the software libraries “RTCZero”, “Arduino_BME280”, “Adafruit_Sensor”, “Wire”, and “LoRaWAN”. The first three have to be installed using the [library](#)

[manager](#) of the Arduino IDE, the fourth library is already installed with the Arduino IDE and the latter library comes with the “Seeeduino LoRaWAN” board installation.

After the sketch has successfully established a connection to The Things Network it measures the air temperature, humidity, and pressure every 30 seconds. From the measured values also the altitude of the device (in meters above ground) is estimated. All four values are being encoded in two byte integer values each and then sent as an 8 bytes data packet to the respective TTN application using LoRaWAN port 8. Please note, that LoRaWAN messages can be addressed to ports 1-255 (port 0 is reserved); these ports are similar to port numbers 0-65535 when using the Internet TCP/IP protocol. Voltage, pressure, altitude, and humidity values are always greater or equal to 0, but the temperature value can also become negative. Negative values are represented as a [two's complement](#); this must be considered in the Payload Decoding Function used in The Things Network (see below).

In between two sensor readings the microcontroller, the LoRaWAN module, and the sensor module are going into deep sleep mode to save battery power. During LoRaWAN data transmission the device draws up to 65mA current. When in sleep mode the entire node only draws around 0.06 mA power. Hence, with a 6600 mAh 3.7V LiPo battery and the current version of the sketch the system should be able to run for many years before recharging (not taking into account the self-discharging rate of the battery).

The source code is provided in the following section *Arduino Sketch for Seeeduino LoRaWAN with GPS sensor node*

2.10.3 Services

The services used for this sensor-node are:

- *TheThingsNetwork* service for LoRaWAN network service.
- *TheThingsNetwork - OGC SensorWeb* integration for uploading LoRaWAN sensor data into OGC infrastructure.

Registration of the sensor node with The Things Network (TTN)

The LoRaWAN protocol makes use of a number of different identifiers, addresses, keys, etc. These are required to unambiguously identify devices, applications, as well as to encrypt and decrypt messages. The names and meanings are nicely explained on a dedicated [TTN web page](#).

The sketch given above connects the sensor node with The Things Network (TTN) using the Over-the-Air-Activation (OTAA) mode. In this mode, we use the three keys AppEUI, DevEUI, AppKey. The DevEUI should normally be delivered with the sensor node by the manufacturer. However, it seems that there is no explicit DevEUI provided with the Seeeduino LoRaWAN module. Therefore, it has to be generated automatically together with the other two keys using the TTN console. Each sensor node must be manually registered in the [TTN console](#) before it can be started. This assumes that you already have a TTN user account (which needs to be created otherwise). In the TTN console [create a new device](#) with also the DevEUI being automatically generated. After the registration of the device the respective keys (AppEUI, DevEUI, AppKey) can be copied from the TTN console and must be pasted into the proper places in the source code of the sketch above. Please make sure that you choose for each of the three keys are in the correct byte ordering (all are in MSB, i.e. in the same ordering as given in the TTN console). A detailed explanation of these steps is given [on this page](#). Then the sketch can be compiled and uploaded to the Seeeduino LoRaWAN microcontroller. Note that the three constants (AppEUI, DevEUI, AppKey) must be changed in the source code for every new sensor node.

Using the OTAA mode has the advantage over the ABP (activation by personalization) mode that during connection the session keys are newly created which improves security. Another advantage is that the packet counter is automatically reset to 0 both in the node and in the TTN application.

TTN Payload Decoding

Everytime a data packet is received by a TTN application a dedicated Javascript function is being called (Payload Decoder Function). This function can be used to decode the received byte string and to create proper Javascript objects or values that can directly be read by humans when looking at the incoming data packet. This is also useful to format the data in a specific way that can then be forwarded to an external application (e.g. a sensor data platform like [MyDevices](#) or [Thingspeak](#)).

Such a forwarding can be configured in the TTN console in the “Integrations” tab. *TTN payload decoder for Seeeduino LoRaWAN with GPS sensor node* given here checks if a packet was received on LoRaWAN port 33 and then assumes that it consists of the 10 bytes encoded as described above. It creates the five Javascript objects ‘temperature’, ‘humidity’, ‘pressure’, ‘altitude’, and ‘vbattery’. Each object has two fields: ‘value’ holds the value and ‘uom’ gives the unit of measure. The source code can simply be copied and pasted into the ‘decoder’ tab in the TTN console after having selected the application. Choose the option ‘Custom’ in the ‘Payload Format’ field. Note that when you also want to handle other sensor nodes sending packets on different LoRaWAN ports, then the Payload Decoder Function can be extended after the end of the if (port==33) { ... } statement by adding else if (port==7) { ... } else if (port==8) { ... } etc.

The Things Network - OGC SensorWeb Integration

The presented Payload Decoder Function works also with the TTN-OGC SWE Integration for the [52° North Sensor Observation Service \(SOS\)](#). This software component can be downloaded from this [repository](#). It connects a TTN application with a running transactional [Sensor Observation Service 2.0.0 \(SOS\)](#). Data packets received from TTN are imported into the SOS. The SOS persistently stores sensor data from an arbitrary number of sensor nodes and can be queried for the most recent as well as for historic sensor data readings. The 52° North SOS comes with its own REST API and a nice web client allowing to browse the stored sensor data in a convenient way.

We are running an instance of the 52° North SOS and the TTN-OGC SWE Integration. The web client for this LoRaWAN sensor node can be accessed [on this webpage](#). Here is a screenshot showing the webclient:

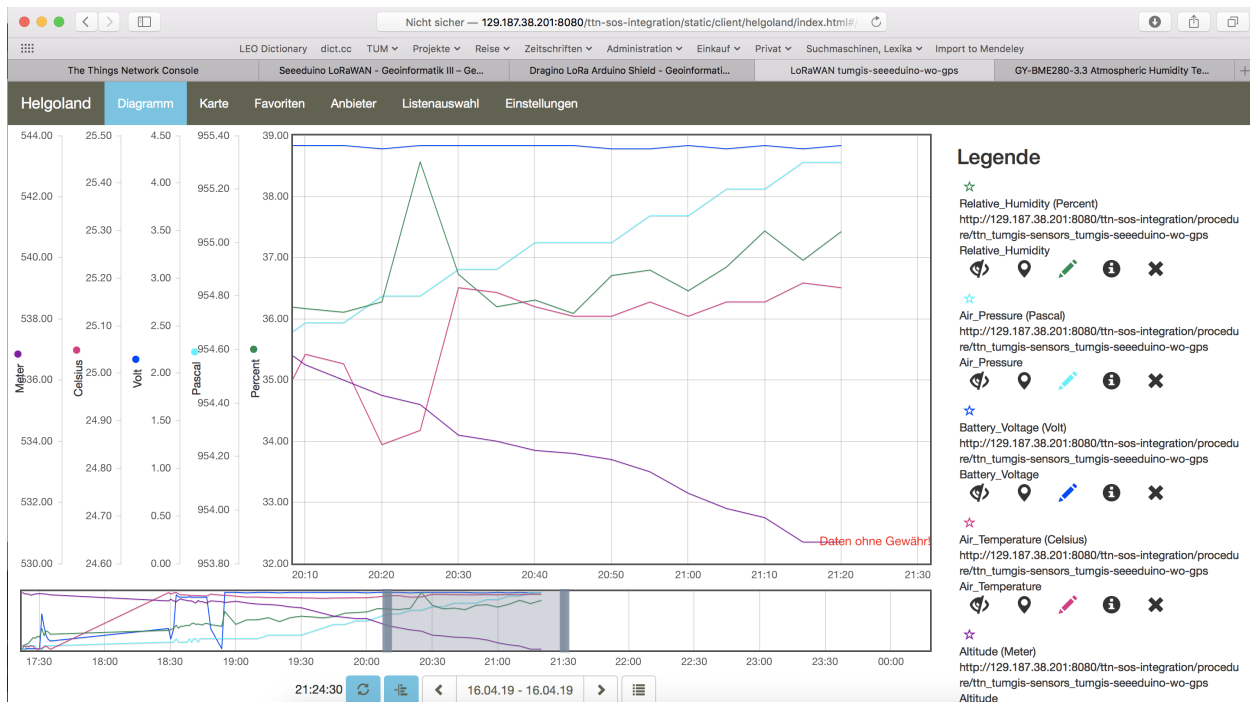


Fig. 33: Web client for data visualization

2.10.4 Code files

Listing 32: Arduino Sketch for Seeeduino LoRaWAN with GPS sensor node

```

1  /*****
2  * Arduino Sketch for a LoRaWAN sensor node that is registered with
3  * 'The Things Network' (TTN) www.thethingsnetwork.org
4  *
5  * Filename: Seeeduino_LoRaWAN_GPS_BME280_OTAA_Sleep_Adafruit_V2.ino
6  *
7  * Author: Thomas H. Kolbe, thomas.kolbe@tum.de
8  * Version: 1.0.1
9  * Last update: 2019-04-17
10 *
11 * This sketch works with a Seeeduino LoRaWAN microcontroller board (with or
12 * without embedded GPS module). See http://wiki.seeedstudio.com/Seeeduino_LoRAWAN/
13 * It requires a Seeed Grove BME280 air temperature, relative humidity,
14 * and air pressure sensor module attached to the I2C Grove connector of
15 * the microcontroller board. The current configuration assumes that
16 * the BME280 is configured to I2C device address 0x76 (default).
17 * The sketch makes a connection to The Things Network (TTN) using
18 * LoRaWAN in OTAA mode. It then sends a data packet of 10 bytes to
19 * LoRaWAN port 33 around every 5 minutes. The packet contains the
20 * following 5 integer values (16 bit, most significant byte (MSB) first):
21 * 1. temperature in Celsius (signed, multiplied by 100)
22 * 2. relative humidity in percent (unsigned, multiplied by 100)
23 * 3. air pressure in Pascal (unsigned, divided by 10)
24 * 4. current altitude in Meters (unsigned, multiplied by 10)
25 * 5. battery voltage in millivolt (unsigned)
26 * These values have to be decoded by the LoRaWAN network controller
27 * using a proper "payload decoder function" written in Javascript.
28 *
29 * Note that when the board is powered over the USB connector and
30 * no battery is connected, the measured battery voltage is incorrect.
31 *
32 * If the board shall be running on a lithium polymer (LiPo) battery,
33 * it is recommended to remove the green power LED from the board or
34 * to cut the connection between the LED and the resistor lying above
35 * of it as the LED constantly draws around 8mW of power. In order to
36 * save energy the sketch puts the GPS module on the board to standby
37 * mode right from the beginning. After each measurement and data transfer
38 * the LoRaWAN module and the sensor is put to standby mode, too, and the
39 * microcontroller goes into deep sleep mode. All components require
40 * a total current of around 0.34mA during sleep mode and up to 65mA
41 * during LoRa transmission for the board version with GPS. The board
42 * version without GPS only requires 0.06mA during sleep mode. Since the
43 * entire system is mostly sleeping, the GPS board should be running
44 * around 2 years on a 6600mAh LiPo battery before recharging
45 * (6600mAh / 0.34mA / 24 = 808 days). The non GPS board version should
46 * even run for more than 10 years...
47 *
48 * This code is based on example code given on the Seeeduino LoRaWAN
49 * wiki page. It utilizes the Open Source libraries "Adafruit_BME280"
50 * and "Adafruit_Sensor" provided by the company Adafruit and the
51 * library "LoRaWan.h" provided by Seeed Studio.
52 *****/

```

(continues on next page)

(continued from previous page)

```

53
54 #include <RTCZero.h>
55 #include <LoRaWAN.h>
56 #include <Wire.h>
57 #include <Adafruit_Sensor.h>
58 #include <Adafruit_BME280.h>
59
60 // Keep the following line, if the board is a Seeeduino LoRaWAN with GPS,
61 // otherwise comment the line out
62
63 #define HAS_GPS 1
64
65 #define BME280_ADDRESS      (0x76)    // I2C device address of the BME280 sensor
66
67 // The barometer of the BME280 can also be used to estimate the current
68 // altitude of the device, if the air pressure at sea level (NN) is known.
69 // The following value has to be set to the current air pressure at NN (in hPa)
70 // in order to give reasonable altitude estimations. Note that this value is
71 // slowly changing over time. For Munich the current value can be obtained
72 // from https://www.meteo.physik.uni-muenchen.de/mesomikro/stadt/messung.php
73
74 #define SEALEVELPRESSURE_HPA (1017.8)
75
76 Adafruit_BME280 bme280;
77
78 RTCZero rtc;
79
80 unsigned char data[10];           // buffer for the LoRaWAN data packet to be_
    ↳ transferred
81 char buffer[256];                 // buffer for text messages received from the_
    ↳ LoRaWAN module for display
82
83
84 void setup(void)
85 {
86     digitalWrite(38, HIGH);       // Provide power to the 4 Grove connectors of_
    ↳ the board
87
88     for(int i = 0; i < 26; i++)    // Set all pins to HIGH to save power_
    ↳ (reduces the
89     {                             // current drawn during deep sleep by around_
    ↳ 0.7mA).
90         if (i!=13) {              // Don't switch on the onboard user LED (pin_
    ↳ 13).
91             pinMode(i, OUTPUT);
92             digitalWrite(i, HIGH);
93         }
94     }
95
96     delay(5000);                  // Wait 5 secs after reset/booting to give_
    ↳ time for potential upload
97                                   // of a new sketch (sketches cannot be_
    ↳ uploaded when in sleep mode)
98     SerialUSB.begin(115200);      // Initialize USB/serial connection
99     delay(500);
100    // while(!SerialUSB);
101    SerialUSB.println("Seeeduino LoRaWAN board started!");

```

(continues on next page)

(continued from previous page)

```

102
103   if(!bme280.begin(BME280_ADDRESS)) { // Initialize the BME280 sensor module
104       SerialUSB.println("BME280 device error!");
105   }
106
107   // Set the BME280 to a very low power operation mode (c.f. chapter 3.5
108   // "Recommended modes of operation" in the BME280 datasheet. See
109   // https://cdn-shop.adafruit.com/datasheets/BST-BME280_DS001-10.pdf );
110   // proper values can only be queried every 60s
111   bme280.setSampling(Adafruit_BME280::MODE_FORCED,
112                       Adafruit_BME280::SAMPLING_X16, // temperature
113                       Adafruit_BME280::SAMPLING_X16, // pressure
114                       Adafruit_BME280::SAMPLING_X16, // humidity
115                       Adafruit_BME280::FILTER_OFF );
116
117   // nrgSave.begin(WAKE_RTC_ALARM);
118   // rtc.begin(TIME_H24);
119
120   #ifndef HAS_GPS
121       Serial.begin(9600); // Initialize serial connection to the GPS
122   ↪ module
123       delay(500);
124       Serial.write("$PMTK161,0*28\r\n"); // Switch GPS module to standby mode as we don
125   ↪ 't use it in this sketch
126   #endif
127
128   lora.init(); // Initialize the LoRaWAN module
129
130   memset(buffer, 0, 256); // clear text buffer
131   lora.getVersion(buffer, 256, 1);
132   memset(buffer, 0, 256); // We call getVersion() two times, because
133   ↪ after a reset the LoRaWAN module can be
134   lora.getVersion(buffer, 256, 1); // in sleep mode and then the first call only
135   ↪ wakes it up and will not be performed.
136   SerialUSB.print(buffer);
137
138   memset(buffer, 0, 256);
139   lora.getId(buffer, 256, 1);
140   SerialUSB.print(buffer);
141
142   // The following three constants (AppEUI, DevEUI, AppKey) must be changed
143   // for every new sensor node. We are using the LoRaWAN OTAA mode (over the
144   // air activation). Each sensor node must be manually registered in the
145   // TTN console at https://console.thethingsnetwork.org before it can be
146   // started. In the TTN console create a new device with the DevEUI also
147   // being automatically generated. After the registration of the device the
148   // three values can be copied from the TTN console. A detailed explanation
149   // of these steps is given in
150   // https://learn.adafruit.com/the-things-network-for-feather?view=all
151
152   // The EUIs and the AppKey must be given in big-endian format, i.e. the
153   // most-significant-byte comes first (as displayed in the TTN console).
154   // For TTN issued AppEUIs the first bytes should be 0x70, 0xB3, 0xD5.
155
156   // void setId(char *DevAddr, char *DevEUI, char *AppEUI);
157   lora.setId(NULL, "xxxxxxxxxxxxxxxx", "yyyyyyyyyyyyyyyy");

```

(continues on next page)

```
// setKey(char *NwkSKey, char *AppSKey, char *AppKey);  
lora.setKey(NULL, NULL, "zzzzzzzzzzzzzzzzzzzzzzzzzzzzzz");  
  
lora.setDeciveMode(LWOTAA);           // select OTAA join mode (note that_  
→setDeciveMode is not a typo; it is misspelled in the library)  
// lora.setDataRate(DR5, EU868);       // SF7, 125 kbps (highest data rate)  
lora.setDataRate(DR3, EU868);         // SF9, 125 kbps (medium data rate and_  
→range)  
// lora.setDataRate(DR0, EU868);       // SF12, 125 kbps (lowest data rate,_  
→highest max. distance)  
  
// lora.setAdaptiveDataRate(false);  
lora.setAdaptiveDataRate(true);      // automatically adapt the data rate  
  
lora.setChannel(0, 868.1);  
lora.setChannel(1, 868.3);  
lora.setChannel(2, 868.5);  
lora.setChannel(3, 867.1);  
lora.setChannel(4, 867.3);  
lora.setChannel(5, 867.5);  
lora.setChannel(6, 867.7);  
lora.setChannel(7, 867.9);  
  
// The following two commands can be left commented out;  
// TTN works with the default values. (It also works when  
// uncommenting the commands, though.)  
// lora.setReceiceWindowFirst(0, 868.1);  
// lora.setReceiceWindowSecond(869.525, DR0);  
  
lora.setDutyCycle(false);            // for debugging purposes only - should_  
→normally be activated  
lora.setJoinDutyCycle(false);        // for debugging purposes only - should_  
→normally be activated  
  
lora.setPower(14);                  // LoRa transceiver power (14 is the_  
→maximum for the 868 MHz band)  
  
// while(!lora.setOTAAJoin(JOIN));  
while(!lora.setOTAAJoin(JOIN,20));   // wait until the node has successfully_  
→joined TTN  
  
lora.setPort(33);                   // all data packets are sent to LoRaWAN_  
→port 33  
}  
  
void loop(void)  
{  
    bool result = false;  
    float temperature, altitude, pressure, humidity;  
    int16_t int16_temperature, int16_humidity, int16_pressure, int16_altitude, int16_  
→vbat;  
  
    bme280.takeForcedMeasurement();   // wake the sensor up for the next readings  
  
    //get and print temperatures  
    SerialUSB.print("Temp: ");  
    SerialUSB.print(temperature = bme280.readTemperature());
```

(continued from previous page)

```

203     SerialUSB.print("C  ");
204
205     //get and print atmospheric pressure data
206     SerialUSB.print("Pressure: ");
207     SerialUSB.print(pressure = bme280.readPressure());
208     SerialUSB.print("Pa  ");
209
210     //get and print altitude data
211     SerialUSB.print("Altitude: ");
212     SerialUSB.print(altitude = bme280.readAltitude(SEALEVELPRESSURE_HPA));
213     SerialUSB.print("m  ");
214
215     //get and print humidity data
216     SerialUSB.print("Humidity: ");
217     SerialUSB.print(humidity = bme280.readHumidity());
218     SerialUSB.print("%  ");
219
220     //get and print battery voltage
221     SerialUSB.print("VBat: ");
222     SerialUSB.print(int16_vbat=lora.getBatteryVoltage());
223     SerialUSB.println("mV");
224
225     int16_temperature = temperature*100.0;
226     int16_humidity = humidity*100.0;
227     int16_pressure = pressure/10.0;
228     int16_altitude = altitude*10.0;
229
230     data[0] = (byte) (int16_temperature >> 8);
231     data[1] = (byte) (int16_temperature & 0x00FF);
232     data[2] = (byte) (int16_humidity >> 8);
233     data[3] = (byte) (int16_humidity & 0x00FF);
234     data[4] = (byte) (int16_pressure >> 8);
235     data[5] = (byte) (int16_pressure & 0x00FF);
236     data[6] = (byte) (int16_altitude >> 8);
237     data[7] = (byte) (int16_altitude & 0x00FF);
238     data[8] = (byte) (int16_vbat >> 8);
239     data[9] = (byte) (int16_vbat & 0x00FF);
240
241     result = lora.transferPacket(data, 10, 5);    // send the data packet (10 bytes)
    ↪ with a default timeout of 5 secs
242
243     if(result)
244     {
245         short length;
246         short rssi;
247
248         memset(buffer, 0, 256);
249         length = lora.receivePacket(buffer, 256, &rssi);
250
251         if(length)
252         {
253             SerialUSB.print("Length is: ");
254             SerialUSB.println(length);
255             SerialUSB.print("RSSI is: ");
256             SerialUSB.println(rssi);
257             SerialUSB.print("Data is: ");
258             for(unsigned char i = 0; i < length; i++)

```

(continues on next page)

(continued from previous page)

```

259         {
260             SerialUSB.print("0x");
261             SerialUSB.print(buffer[i], HEX);
262             SerialUSB.print(" ");
263         }
264         SerialUSB.println();
265     }
266 }
267
268 lora.setDeviceLowPower(); // bring the LoRaWAN module to sleep mode
269 doSleep((5*60-8)*1000); // deep sleep for 292 secs (+ 3 secs transmission_
↪time + 5 secs timeout = 300 secs period)
270 lora.setPort(33); // send some command to wake up the LoRaWAN module_
↪again
271 }
272
273 // The following function implements deep sleep waiting. When being called the
274 // CPU goes into deep sleep mode (for power saving). It is woken up again by
275 // the CPU-internal real time clock (RTC) after the configured time.
276 //
277 // A similar function would also be available in the standard "ArduinoLowPower"_
↪library.
278 // However, in order to be able to use that library with the Seeeduino LoRaWAN board,
279 // four files in the package "Seeed SAMD boards by Seeed Studio Version 1.3.0" that is
280 // installed using the Arduino IDE board manager need to be patched. The reason is_
↪that
281 // Seeed Studio have not updated their files to a recent Arduino SAMD version yet
282 // and the official "ArduinoLowPower" library provided by the Arduino foundation is
283 // referring to some missing functions. For further information see here:
284 // https://forum.arduino.cc/index.php?topic=603900.0 and here:
285 // https://github.com/arduino/ArduinoCore-samd/commit/
↪b9ac48c782ca4b82ffd7e65bf2c956152386d82b
286
287 void doSleep(uint32_t millis) {
288     if (!rtc.isConfigured()) { // if called for the first time,
289         rtc.begin(false); // then initialize the real time clock (RTC)
290     }
291
292     uint32_t now = rtc.getEpoch();
293     rtc.setAlarmEpoch(now + millis/1000);
294     rtc.enableAlarm(rtc.MATCH_HHMMSS);
295
296     rtc.standbyMode(); // bring CPU into deep sleep mode (until woken up_
↪by the RTC)
297 }

```

Listing 33: TTN payload decoder for Seeeduino LoRaWAN with GPS sensor node

```

1 function Decoder (bytes, port) {
2     var result = {};
3     var transformers = {};
4
5     if (port==33) {
6         transformers = {
7             'temperature': function transform (bytes) {

```

(continues on next page)

(continued from previous page)

```

8      value=bytes[0]*256 + bytes[1];
9      if (value>=32768) value=value-65536;
10     return value/100.0;
11 },
12 'humidity': function transform (bytes) {
13     return (bytes[0]*256 + bytes[1])/100.0;
14 },
15 'pressure': function transform (bytes) {
16     return (bytes[0]*256 + bytes[1])/10.0;
17 },
18 'altitude': function transform (bytes) {
19     return (bytes[0]*256 + bytes[1])/10.0;
20 },
21 'vbattery': function transform (bytes) {
22     return (bytes[0]*256 + bytes[1])/1000.0;
23 }
24 }
25
26 result['temperature'] = {
27     value: transformers['temperature'](bytes.slice(0, 2)),
28     uom: 'Celsius',
29 }
30
31 result['humidity'] = {
32     value: transformers['humidity'](bytes.slice(2, 4)),
33     uom: 'Percent',
34 }
35
36 result['pressure'] = {
37     value: transformers['pressure'](bytes.slice(4, 6)),
38     uom: 'hPa',
39 }
40
41 result['altitude'] = {
42     value: transformers['altitude'](bytes.slice(6, 8)),
43     uom: 'Meter',
44 }
45
46 result['vbattery'] = {
47     value: transformers['vbattery'](bytes.slice(8, 10)),
48     uom: 'Volt',
49 }
50 }
51
52 return result;
53 }

```

2.10.5 References

- [Seeeduino LoRaWAN GPS microcontroller](#)
- [Seeeduino LoRaWAN Wiki with instructions](#)
- [A short presentation on LoRaWAN basics and using the Seeeduino LoRaWAN board](#)
- [L70 GPS receiver module](#) from the company Quectel Wireless Solutions

- Adafruit GPS library (can be used with the L70 GPS module)

On battery saving / using the deep sleep mode

- Adafruit Feather 32u4 LoRa - long transmission time after deep sleep - End Devices (Nodes) - The Things Network
- Full Arduino Mini LoraWAN and 1.3uA Sleep Mode - End Devices (Nodes) - The Things Network
- Adding Method to Adjust hal_ticks Upon Waking Up from Sleep · Issue #109 · matthijskooijman/arduino-lmic
- minilora-test/minilora-test.ino at cbe686826bd84fac8381de47b5f5b02dd47c2ca0 · tkerby/minilora-test
- Arduino-LMIC library with low power mode - Mario Zwiers

2.11 Sodaq ONE

2.11.1 Hardware

Micro-controller



Fig. 34: SODAQ ONE-EU-RN2483-V3 from SODAQ. SODAQ ONE tutorial with explanations, schematics, datasheets, and examples. SODAQ ONE module pinout

The SODAQ ONE-EU-RN2483-V3 is a very compact module operated by the 32bit ATSAM21G18 microcontroller running at 48 MHz. It has 256 KB flash memory (to store the program code) and 32 KB of RAM (to store variables, status information, and buffers), and up to 16 KB of emulated EEPROM (to store configuration data). The operating voltage of the board is 3.3V (this is important when attaching sensors and other peripherals; they also must operate on 3.3V). The board offers 14 general purpose analog/digital input/output pins (14 GPIOs) all of which can also be used for PWM output, and one of which can be used as an analog output pin (with 10bit digital analog converter (DAC)). The microcontroller has three serial ports (programmable Universal Asynchronous Receiver and Transmitter, UART); the first is connected internally via a USB/Serial converter to the USB port of the board, the second is connected internally to the LoRaWAN module, and the third is freely usable using GPIO pins D12 (TX) and D13 (RX). Furthermore, the board has one I2C port and four of the GPIO lines can be used as SPI port. The SODAQ ONE comes with an embedded Lithium polymer battery management chip and status indicator led, which allows to directly connect a 3.7V LiPo rechargeable battery that will be automatically recharged when the board is powered over its USB connector. Above, a solar charge controller is embedded allowing to directly connect a photovoltaic panel (4.5V to 6V) to recharge the battery. The board also features an RGB LED, a LSM303AGR module (3 axis magnetometer and 3 axis accelerometer), and an uBlox EVA 8M GPS module. The SODAQ ONE board is available from the manufacturer in Rotterdam, The Netherlands, for around 100 €.

The LoRa transmitter and receiver is encapsulated within a [Microchip RN2483](#) LoRaWAN module. It uses the LoRa chip SX1276 from the company Semtech and is dedicated to the 868 MHz frequency band. The RF module contains its own microcontroller, which implements the LoRaWAN protocol. The module is connected via the serial interface to the ATSAM21G18 microcontroller and can be controlled by sending text commands. The implemented LoRaWAN functionality is compatible with LoRaWAN Class A. The detailed explanation of the module is given in the [RN2483 datasheet](#) and all supported commands in the [RN2483 Command Reference](#).

The SODAQ ONE is installed on a [ONE Base board](#) offering 7 on-board Grove connectors. ‘Grove’ is a framework developed by the company [Seeed Studio](#) standardizing the connectors, operating voltages, and pin configurations for attaching peripherals like [sensors](#), [actuators](#), and [displays](#) to microcontrollers. Please note that grove modules to be used with the SODAQ ONE must be working with 3.3V (the normal operating voltage for the Arduino UNO microcontroller and its shields is 5V).

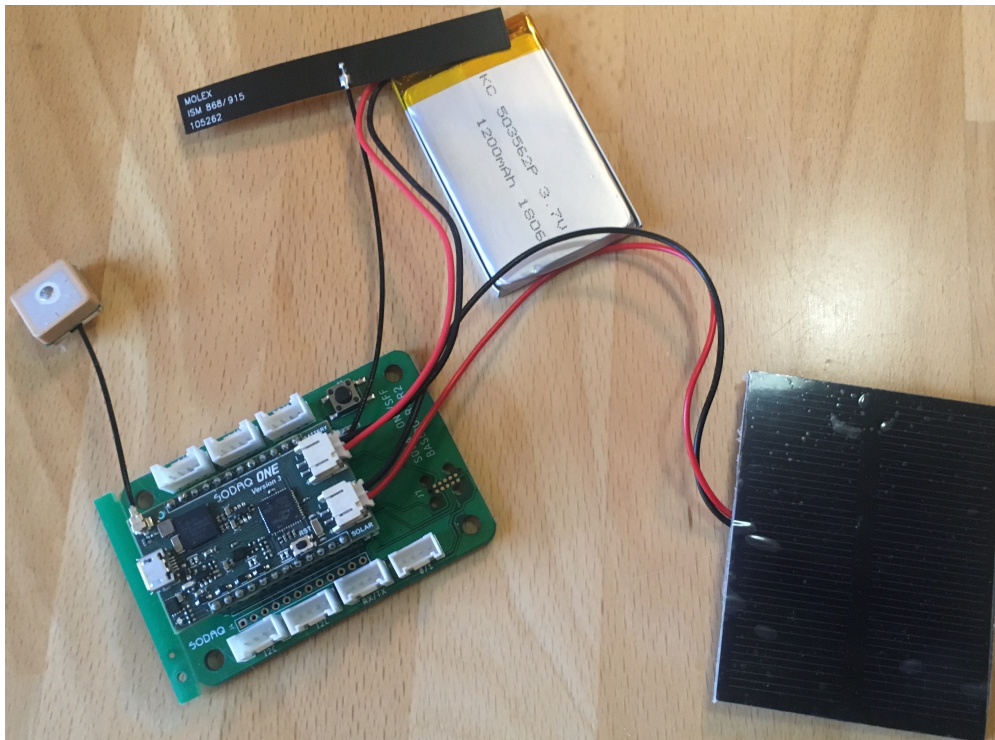


Fig. 35: The SODAQ ONE-EU-RN2483-V3 installed on a [ONE Base](#) board with attached GPS antenna (left), LoRa patch antenna (top), a 1200 mAh lithium polymer (LiPo) battery (top), and an attached 0.5W photovoltaic panel.

Sensor

The embedded [LSM303AGR](#) is an ultra-low-power 3D electronic compass featuring linear acceleration full scales of $\pm 2g/\pm 4g/\pm 8g/\pm 16g$ and a magnetic field dynamic range of ± 50 gauss. The module can be configured to generate an interrupt signal for free-fall, motion detection and magnetic field detection. It is connected to the microcontroller via I2C bus. When the SODAQ ONE is used to track moving objects (like bikes, people, or animals), the LSM303AGR sensor can be used to wake up the microcontroller and the GPS from power saving mode (deep sleep). This way, the power consumption of the SODAQ ONE can be minimized (only when a movement has been detected, a new GPS position fix has to be acquired). The detailed explanation of the module is given in the [LSM303AGR datasheet](#).

The embedded [uBlox EVA 8M GPS](#) module is a standard precision GNSS receiver with 72 channels supporting GPS and GLONASS. The module is capable to report up to 18 positions per second (18 Hz). It is connected to the microcontroller via I2C bus and supports different power saving modes. A detailed explanation of the module is given in the [uBlox EVA 8M datasheet](#).

2.11.2 Software

The section is still to be written.

2.11.3 Services

The section is still to be written.

2.11.4 Code files

Listing 34: Arduino Sketch for Sodaq ONE sensor node

```
No source code yet!
```

Listing 35: TTN payload decoder for Sodaq ONE sensor node

```
Yet to be written
```

2.11.5 References

- [SODAQ ONE-EU-RN2483-V3 microcontroller board](#)
- [SODAQ ONE tutorial, schematics, libraries, examples](#)
- [SODAQ ONE Base board](#)
- [SODAQ main github page](#)
- [Microchip RN2483 LoRaWAN module datasheet](#)
- [Microchip RN2483 LoRaWAN module command reference](#)
- [SODAQ RN2483 library](#)
- [RN2483 Firmware Updater](#)
- [LSM303AGR module datasheet](#)
- [SODAQ LSM303AGR library](#)
- [uBlox EVA 8M GPS module](#)
- [SODAQ uBlox GPS library](#)
- [SODAQ Universal Tracker](#)
- [SODAQ ONE Tracker v3](#)

2.12 Wemos TTGO T-Beam

2.12.1 Hardware



Fig. 36: TTGO T-Beam from Wemos. TTGO T-Beam pinout, example code.

Micro-controller

The [Wemos TTGO T-Beam](#) is especially suited for mobile operations, because it has an on-board battery holder for an 18650 lithium polymer (LiPo) battery, a GPS receiver, a LoRa transceiver module (using the LoRa chip SX1276) dedicated to the 868 MHz frequency band, and an embedded battery charger. The module is operated by the [Espressif ESP32 microcontroller](#) board, which contains a dual-core Xtensa 32bit LX6 processor running with up to 240MHz, 4 MB of flash memory (to store the program code and some files within a file system), and 520 KB of RAM (to store variables, status information, and buffers). The ESP32 module also has built-in WiFi and Bluetooth LE connectivity. In addition, the TTGO T-Beam has 4 MB of PSRAM (pseudo static RAM) that is used as a memory extension for the ESP32. The operating voltage of the board is 3.3V (this is important when attaching sensors and other peripherals; they also must operate on 3.3V). The board offers 18 general purpose input/output pins (18 GPIOs), from which up to 12 can be used as analog input pins (with 12bit analog digital converters (ADC)) and one as analog output pin (8bit digital analog converter (DAC)). Some GPIO pins can be used as serial port (programmable Universal Asynchronous Receiver and Transmitter, UART), I2C port, SPI port, and I2S port. The USB port is connected internally via a USB/Serial converter to another serial port (UART). The WiFi and Bluetooth antenna are mounted on the TTGO T-Beam board. A small GPS antenna is connected via a pigtail to an U.FL / IPX connector. The LoRa antenna has to be connected via an SMA-type connector. The TTGO T-Beam is available from Chinese sellers for around 23 € (I have not found a European shop where it can be bought yet).

The LoRa transmitter and receiver is encapsulated within a LoRa module. It uses the LoRa chip SX1276 from the company Semtech and is dedicated to the 868 MHz frequency band. The LoRa module is connected via SPI interface to the microcontroller and all of the required connections of the LoRa transceiver pins with the microcontroller are already built-in on the TTGO T-Beam board. Since the module only implements the LoRa physical layer, the LoRaWAN protocol stack must be implemented in software on the microcontroller. We are using the Arduino library LMIC for that purpose (see below). The implemented LoRaWAN functionality is compatible with LoRaWAN Class A/C.

Sensor

While the GPS module of the TTGO T-Beam is labelled as a uBlox NEO-6 module, this does not seem to be true. The module seems to be of a Chinese brand instead that is not fully compatible with the uBlox NEO-6.

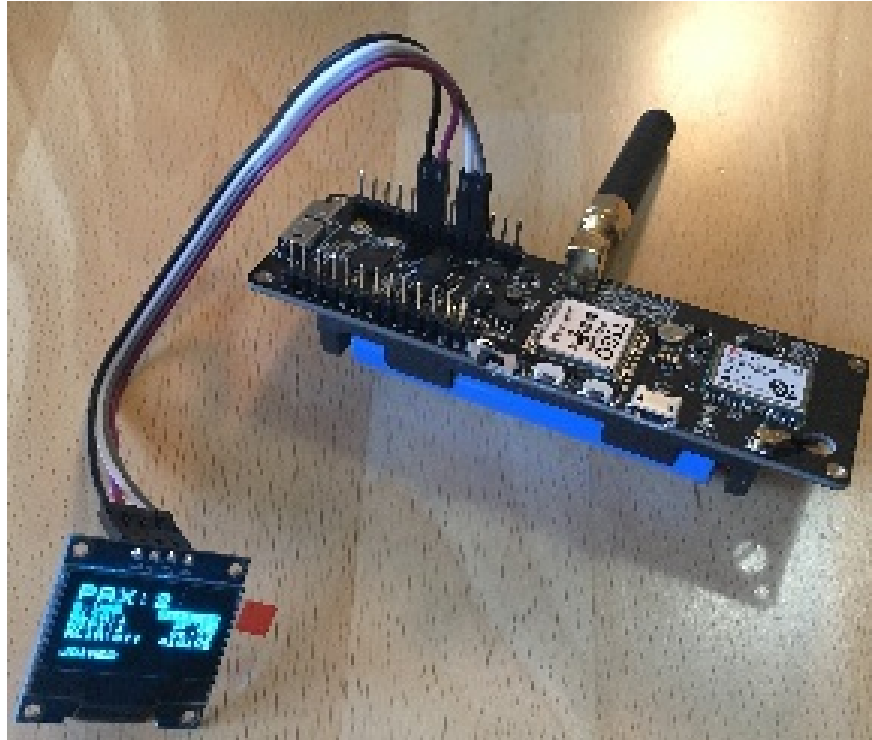


Fig. 37: The Wemos TTGO T-Beam with a 2200 mAh 18650 lithium polymer (LiPo) battery (in the battery holder on the backside of the board) with attached OLED display.

The embedded [uBlox EVA 8M GPS](#) module is a standard precision GNSS receiver with 72 channels supporting GPS and GLONASS. The module is capable to report up to 18 positions per second (18 Hz). It is connected to the microcontroller via I2C bus and supports different power saving modes. A detailed explanation of the module is given in the [uBlox EVA 8M datasheet](#).

Display

We have attached a 0.96 in monochrome OLED display to the I2C bus of the TTGO T-Beam module. The display is using an SD1306 controller and has a resolution of 64 x 64 pixels.

2.12.2 Software

The section is still to be written.

2.12.3 Services

The section is still to be written.

2.12.4 Code files

Listing 36: Arduino Sketch for Wemos TTGO T-Beam sensor node

```
1 No source code yet!
```

Listing 37: TTN payload decoder for Wemos TTGO T-Beam sensor node

```
1 Yet to be written
```

2.12.5 References

- [Wemos TTGO T-Beam pinout, libraries, examples](#)
- [TTGO T-Beam TinyMicros Wiki \(links to schematics, GPS datasheet\)](#)
- [TTGO T-Beam – Kompakter Knochen zum Mappen](#)
- [TTNMapper on the TTGO T-Beam](#)
- [TTNMapper on the TTGO T-Beam with Deep Sleep](#)
- [Wifi & BLE driven passenger flow metering with cheap ESP32 boards](#)

CHAPTER 3

Indices and tables

- `genindex`
- `search`