

This work is licenced under the Creative Commons Attribution-NonCommercial 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Building a LoRa node and connecting it to The Things Network (TTN)

Version 1.0 Frank Beks March, Robin Harris and Rob Miles

In this exercise we are going to build a LoRa node and connect it to The Things Network. At the completion of the exercise you should be logging sensor readings (temperature, pressure and humidity) on The Things Network, ready for incorporation into any application that you might fancy creating.

Identifying the components

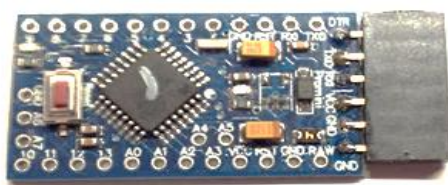
These are the components that you will be using. You can get a complete set at TinyTronics - <https://www.tinytronics.nl/shop/en>

- RFM Module RFM95W (868 Mhz)
- Pro Mini 3.3V 8Mhz (5V will NOT work!!)
- FT232RL-3.3v-5v-TTL-USB-Serial-Port-Adapter (has to have a 3.3V option!)
- USB cable for FTDI
- 1x Led 3mm red
- 1x resistor 470 ohm
- Header 6p female (Arduino programming)
- Header 40p divided in: 2x 4p, 1x 6p, 1x 3p and 1x 2p
- PCB 'DougLarue'
- Sensor BME280 (Temperature and air pressure)
- Battery holder 2x AA
- 2 AA batteries
- optional: 2x female headers 2p (battery and led connection)

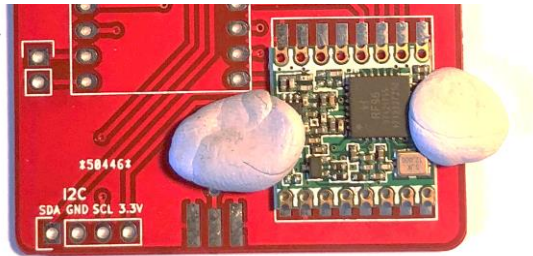
Building the Node

The first thing we need to do is build the node. This will require some soldering.

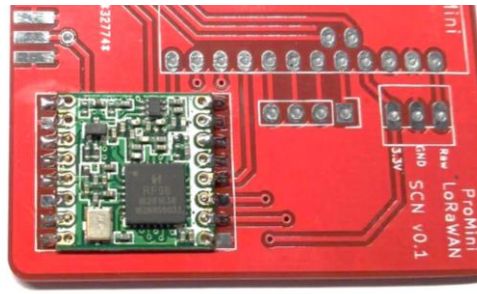
Connect the 6p female header to the Arduino. To get it low profile bend it so it is straight with the PCB, or you can leave it sticking up.



Position the RFM95 radio chip. The crystal should be in the right lower corner (as shown in the picture). Start with just a little solder in two opposite corners. You can use some Blu-tack to hold it in place while you solder it

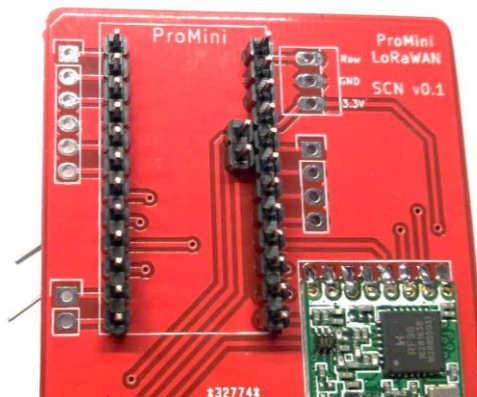


Align the solder pads and if the module is in place, solder the other pads. Do use as little as possible solder! Check the connections carefully!

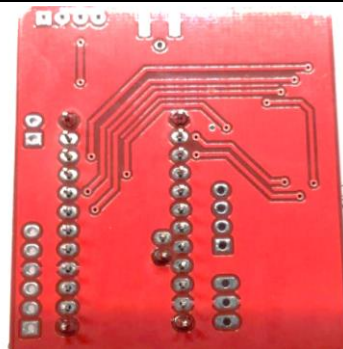


Put two 12p headers into the ProMini socket. Also put the 2p header inside the ProMini area. **Don't forget these.**

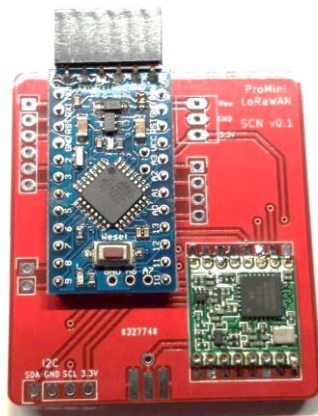
By holding them with a piece of paper you can turn around the PCB. (You can also use the ProMini to keep the pins in place by using an elastic band.)



First solder the end pins, align the headers straight and solder all the other pins.



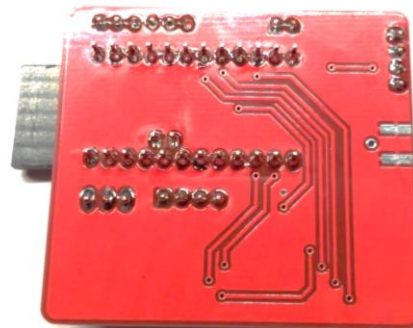
Turn around the PCB and solder the ProMini in place. Do not forget the two pin header!



Now solder the 6p, 2p, 3p and 4p header.



When you've finished the underside of the board should look like this. Check your connections carefully for shorts between pins or dry joints.

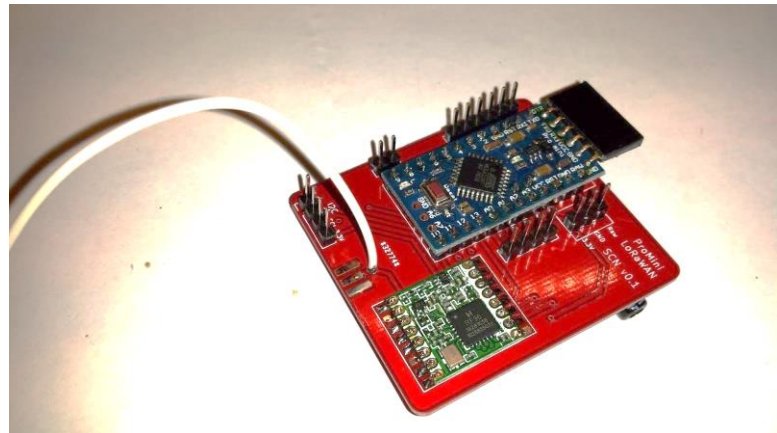


Cut a piece of wire, length 82.2 mm. This is a $\frac{1}{4}$ wave length antenna. You may add some short soldering end.

Wavelength $= 300 / 868 = 0.3456\text{m}$ Wavelength $/ 4 = 0.0864\text{ m}$
correction $0.95 * 0.0864 = 0.0822\text{m}$ (to determine the precise length you can do some experiments, or use proper test equipment like a SWR meter)

Solder the Antenna into the hole near the connector pins. It is also possible to solder a SMA edge connector.

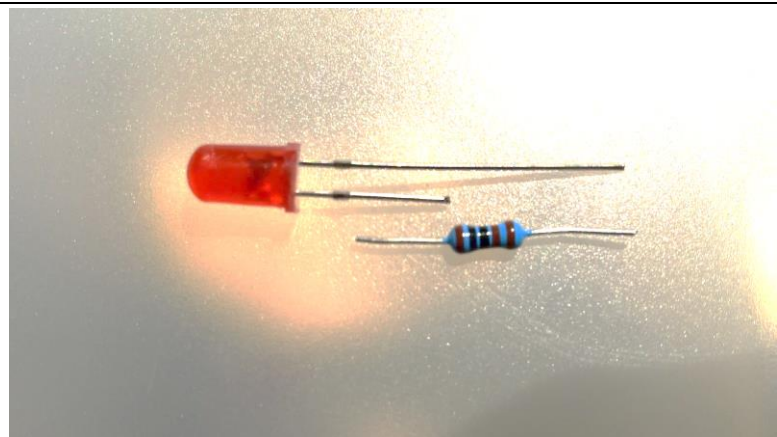
You might like to make the wire a bit too long, and then measure and cut it once it has been fitted.



Solder a header on the BME280 barometric sensor

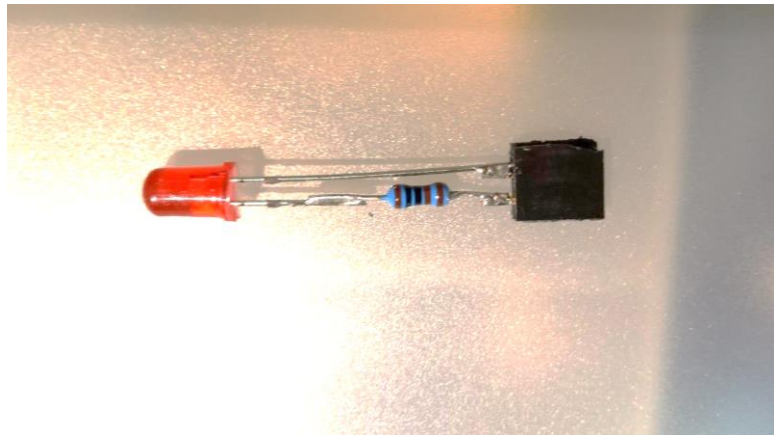


Prepare the LED, this requires a 'helping hand' for soldering. Cut the longest wire off the LED ('+'), and the 1K resistor as shown

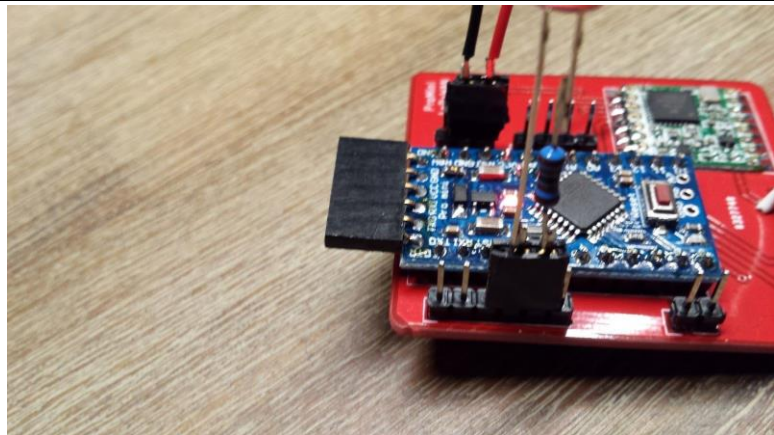


Now solder the resistor, you may use a spare female header to create a 'connector' or some female jumper wires

If you want a fancy finish, ask us for some heat shrink tubing we can put around the resistor to make it tidier and reduce the chance of the led wires touching each other.

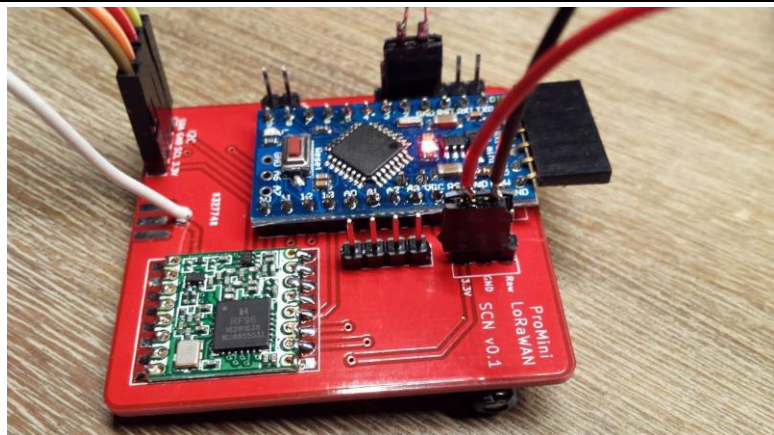


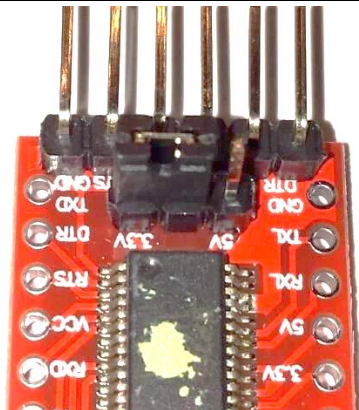
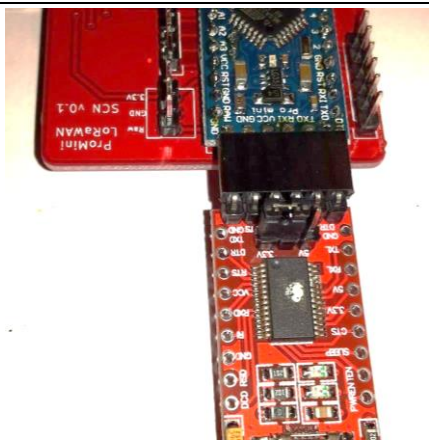
Put the LED 'connector' to pins 4 and 5 of the 6 pin jumper, as shown (Pin 4: GND, PIN 5 - D2), the resistor pin ('+') goes to pin 5 / D2. You may even solder the pins directly



You can solder the battery holder on the pins GND (black) and 3.3V (red), you may use a female header here as well.

Don't put the batteries in just yet.



<p>Put the Jumper on the FTDI board on 3.3V position!</p>	
<p>Connect the USB cable to the FTDI board, the FTDI board to the Arduino as shown.</p>	

Starting up the Arduino Environment

Now we need to get the software into our node to make it work. We are going to start by installing the Arduino integrated development environment (IDE) and then we'll configure it for our hardware.

Installing the Arduino IDE

You can download the software for free from the Arduino site.

<https://www.arduino.cc/en/Guide/HomePage>

Install the Arduino IDE on your favourite platform (Mac, Linux or Windows). These examples work only with the latest Arduino IDE (tested on 1.6.9).

Deploying our first program

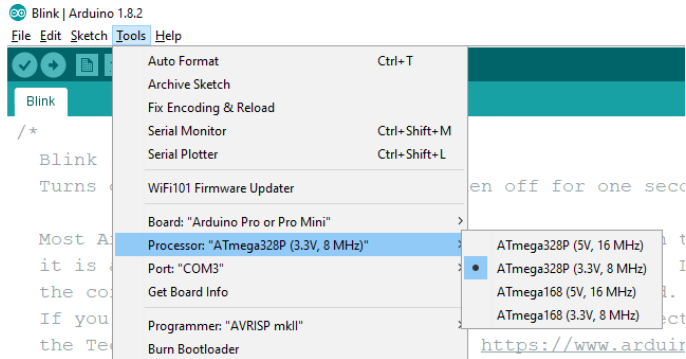
The device we are using is described here: <https://www.arduino.cc/en/Guide/ArduinoProMini>

To program it we connect the programming board, as we did in the last step above. Ensure that the jumper on the programming board is on the '3V' side.

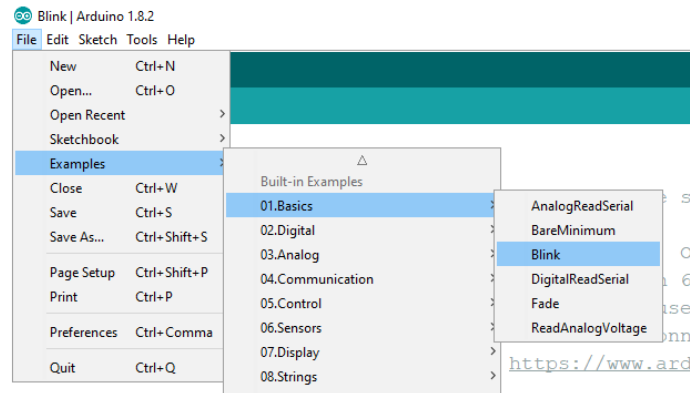
Connect the programmer and the Pro Mini so that the components are both facing up. Use the USB cable that is supplied to plug the programmer into your computer.

Start the Arduino IDE and select the Tools menu. Make sure that you are using Start the Arduino IDE and that the board that is selected is Arduino Pro or Pro mini' with the Processor 'ATmega328 (3.3V,

8Mhz). In the screenshot below the programmer is connected to COM6, this may be different on your system.

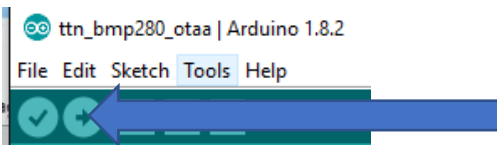


We are going to test the board with a program that will flash a LED. Open up the File menu and select the Blink program as shown below.



This will open a new IDE with the blink program inside it.

Press the arrow at the top left of the IDE to deploy the program to your Arduino and run it. You should see a LED on the Arduino flash once a second.



Great: we have a working programming environment!

Testing the LED

You should have a LED connected to digital input/output pin 2. Now we can test this led. We are going to do this by re-defining the address that is used by the blink program. Add the following statement immediately before the setup method in the Blink program:

```
#define LED_BUILTIN 2
```

Pro tip: You can copy the above statement out of this document and paste it into your program code to make the typing easier. When you have finished the program should look like this:

This work is licensed under the Creative Commons Attribution-NonCommercial 4.0 International License. Page 7

```

Blink | Arduino 1.8.2
File Edit Sketch Tools Help
Blink $
  modified 2 Sep 2016
  by Arturo Guadalupi

  modified 8 Sep 2016
  by Colby Newman
*/

#define LED_BUILTIN 2

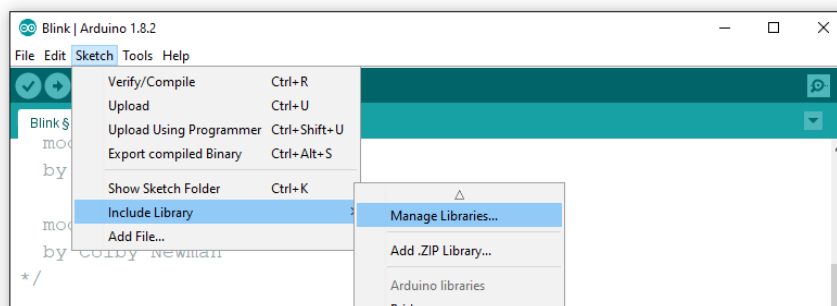
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

```

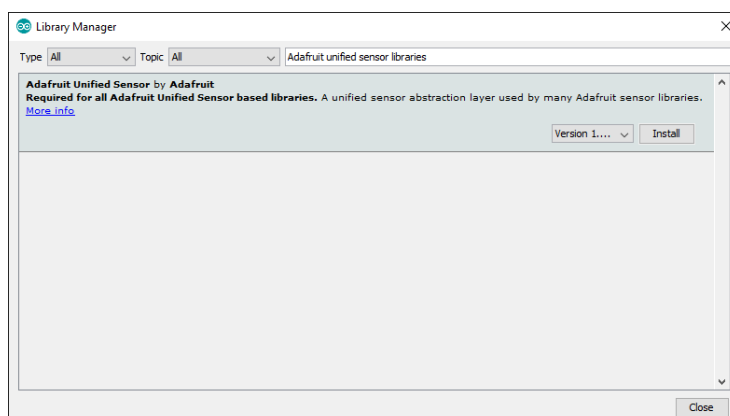
Now run the program again. This time the LED that you connected to the board will flash. If it doesn't check your wiring (perhaps you plugged the LED in the wrong way around).

Using the BME280 sensor

Now we need to get the sensor working. Before we can use the sensor, we need to install some Arduino libraries that our software will use to communicate with it. Open the Manage Libraries dialog.

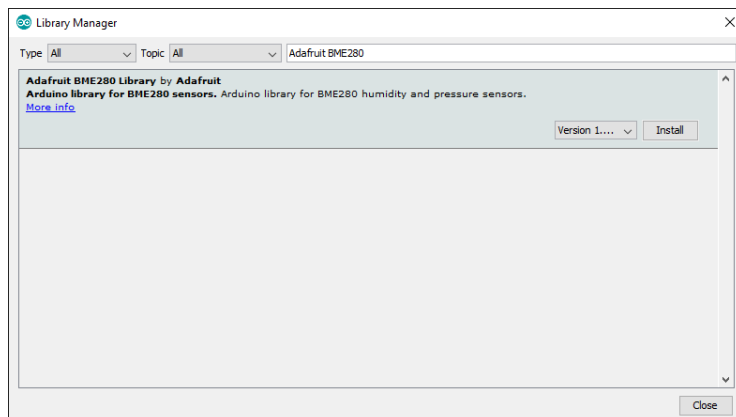


Search the libraries for “Adafruit unified sensor libraries”. Select the library and click install.



Now that we have the unified library, the next thing we need is the BME280 library. Make another search for “Adafruit BME280” and install that library.

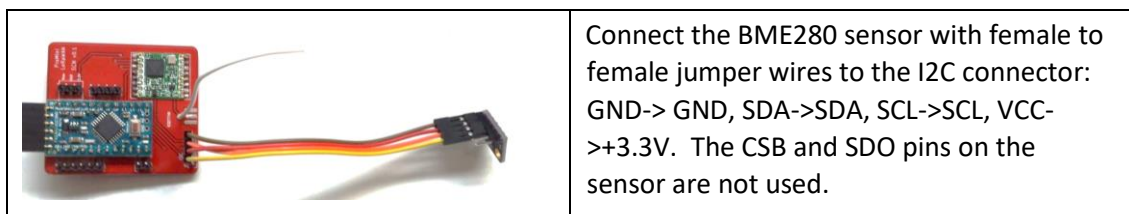
This work is licensed under the Creative Commons Attribution-NonCommercial 4.0 International License. Page 8



Now close the library manager. The libraries are installed on the PC for use by that user of the PC (they are actually in the Documents folder on your PC). This means that if you switch to a different PC you will have to install the libraries again.

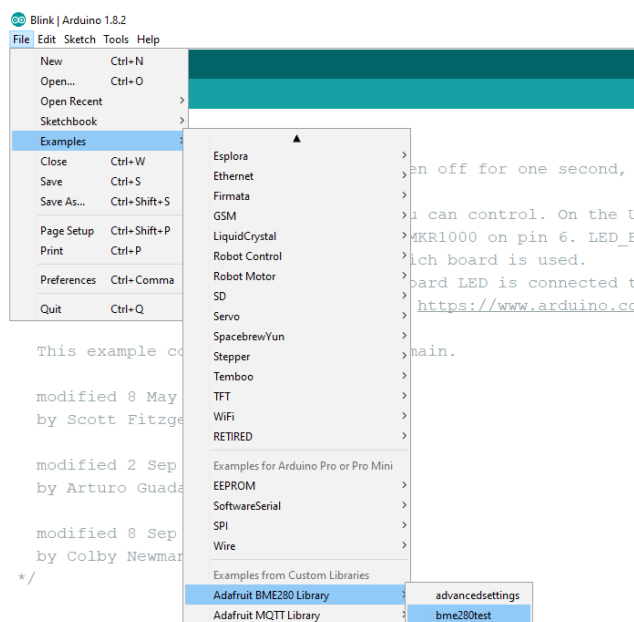
Connect the BME280 sensor

Now we need to connect the sensor to the Arduino board. We are going to be using the I2C (pronounced “I squared C”) connection.



Load the BME280 Test Program

Now we are going to run a test program to make sure that our sensor works correctly. Load the example as shown below.



This work is licensed under the Creative Commons Attribution-NonCommercial 4.0 International License. Page 9

If you run this example it won't work. We need to make a change to the example code because we are using a version of the BME280 sensor which has a different address on the I2C connection that we are using. We just need to change one statement to add the new address. Find the line that says:

```
status = bme.begin();
```

change this to:

```
status = bme.begin(0x76);
```

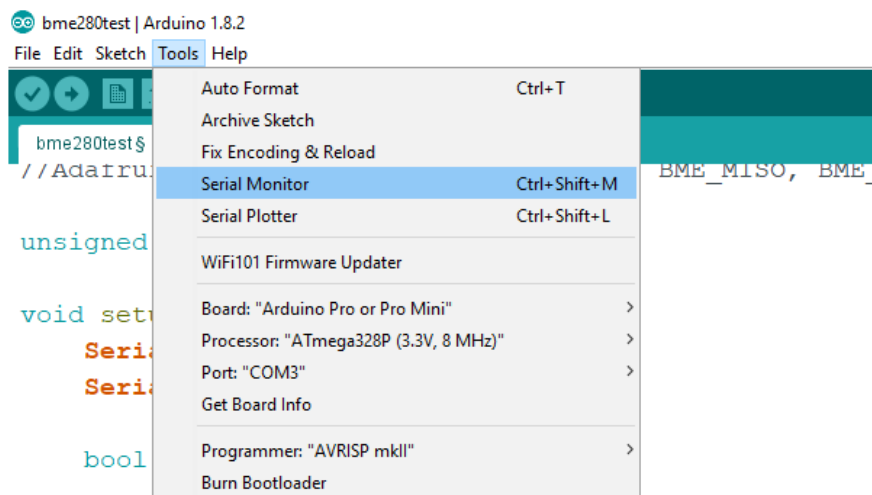
The program should now look as follows. The line that we have changed is highlighted.

```
void setup() {
  Serial.begin(9600);
  Serial.println(F("BME280 test"));

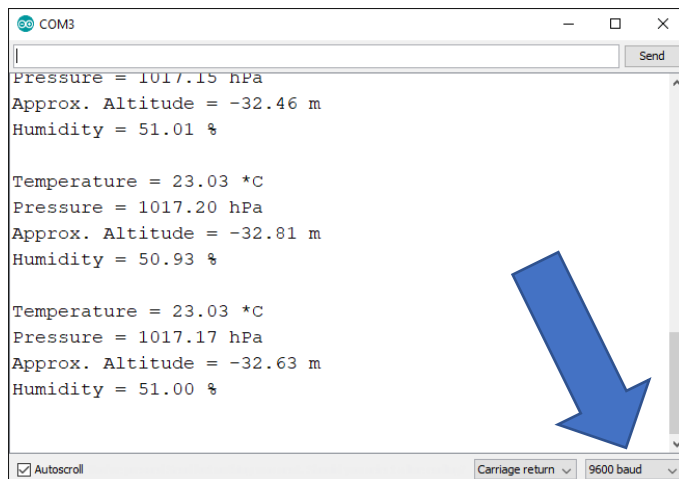
  bool status;

  // default settings
  // (you can also pass in a Wire library object like &Wire2)
  status = bme.begin(0x76);
  if (!status) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
  }
}
```

Now run the program. This will connect to the sensor and start fetching values from it. We can see the values by opening the Serial Monitor in the Tools menu, as shown below.



This should show you the readings the sensor is producing. If you don't see anything, or you see random characters, you might need to change the baud rate of the terminal connection. This is the rate at which the Arduino sends messages to your computer. The example program uses 9600 baud, and so the setting in the Serial Monitor should match this. You can set it on the Serial Monitor on the bottom right hand side, as indicated below.



```
COM3
Send

Pressure = 1017.15 hPa
Approx. Altitude = -32.46 m
Humidity = 51.01 %

Temperature = 23.03 *C
Pressure = 1017.20 hPa
Approx. Altitude = -32.81 m
Humidity = 50.93 %

Temperature = 23.03 *C
Pressure = 1017.17 hPa
Approx. Altitude = -32.63 m
Humidity = 51.00 %

Autoscroll Carriage return 9600 baud
```

Put your finger on the sensor (the small metal unit) and see the temperature rise. Now that we have some readings, we connect our board to The Things Network. But perhaps you might like to have a coffee first....

Getting started on 'The Things Network'

The Things Network is free to use and provides a great place to develop and deploy your LoRa applications. It is the network to which the Hull LoRa gateways are connected. Once registered on the site you can create LoRa applications to which LoRa nodes are connected.

Create an Account

To use the dashboard, you need a The Things Network account. You can create an account here: <https://account.thethingsnetwork.org/users/login> .

After registering and validating your email address, you will be able to log in to The Things Network Dashboard.

The Things Network Dashboard

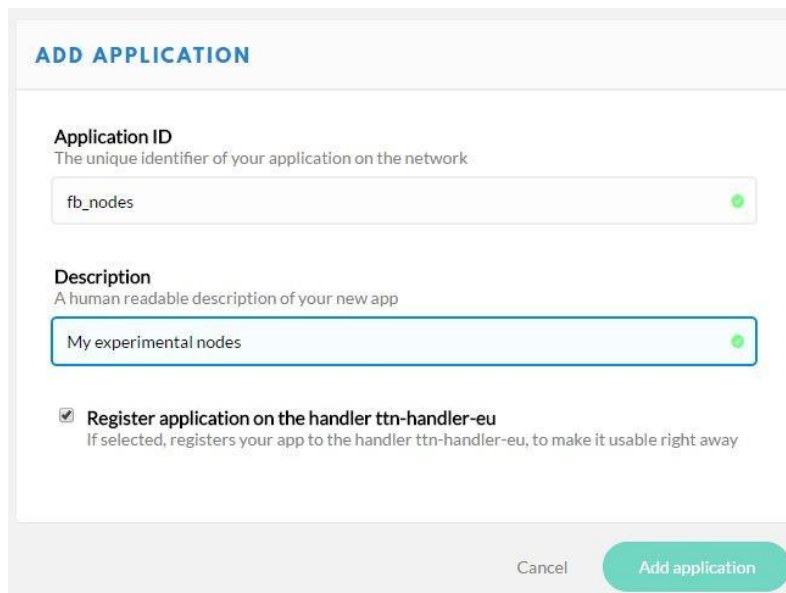
Your applications and devices are managed by The Things Network Dashboard.

[Create an Application](https://console.thethingsnetwork.org/applications) <https://console.thethingsnetwork.org/applications>

Choose 'Add application'

Give your Application a unique ID. You can use ONLY lowercase! You can add a unique number to get uniqueness over the ttn network (this is a global ID).

Your description can be any description you like.

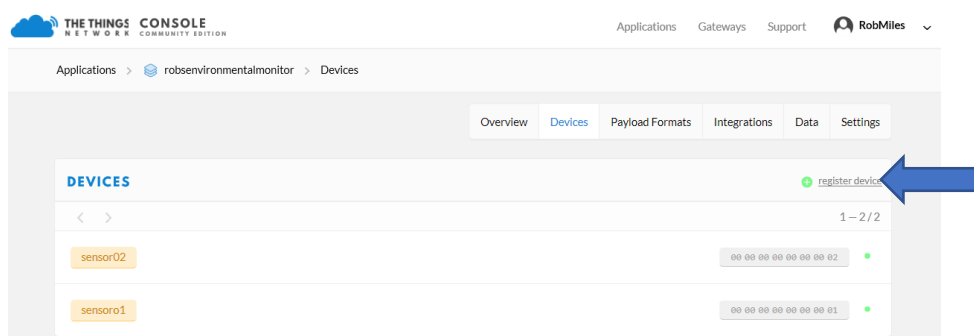


Add a node using Activation by Personalization (ABP)

A node is bound to a particular application and can only be used by that application. The node contains security key values which are used to create the messages that are sent to the LoRa gateway. There are two ways that a node can be bound to an application: Activation By Personalization (ABP) and Over the Air Activation (OTAA).

Activation by Personalization (ABP) is a method where the security keys are stored in the device (i.e. we will build the key values into the programs in the Arduino). This is harder to manage if you have a large number of nodes, because each node has to be uniquely configured. If a node uses Over The Air Activation (OTAA) the Arduino code contains application only and requests an access key the first time it connects to the gateway and joins the application. The node then stores the key value in EEPROM and uses this value in future messages.

For our first tests we are going to use Activation By Personalization. The first step is to create the Device itself. Go to your Things Network dashboard and select the Devices tab. Then select the register device link on the right.



Now you need to give the device a name and a network ID as shown below. The name is a string containing lowercase letters, digits and the underscore character. The network ID is a sequence of 8 bytes, expressed as hex digits. I've made my third sensor and given it a very unimaginative number.

REGISTER DEVICE
[bulk import devices](#)

Device ID
This is the unique identifier for the device in this app. The device ID will be immutable.

Device EUI
The device EUI is the unique identifier for this device on the network. You can change the EUI later.

App Key
The App Key will be used to secure the communication between you device and the network.

App EUI

Cancel Register

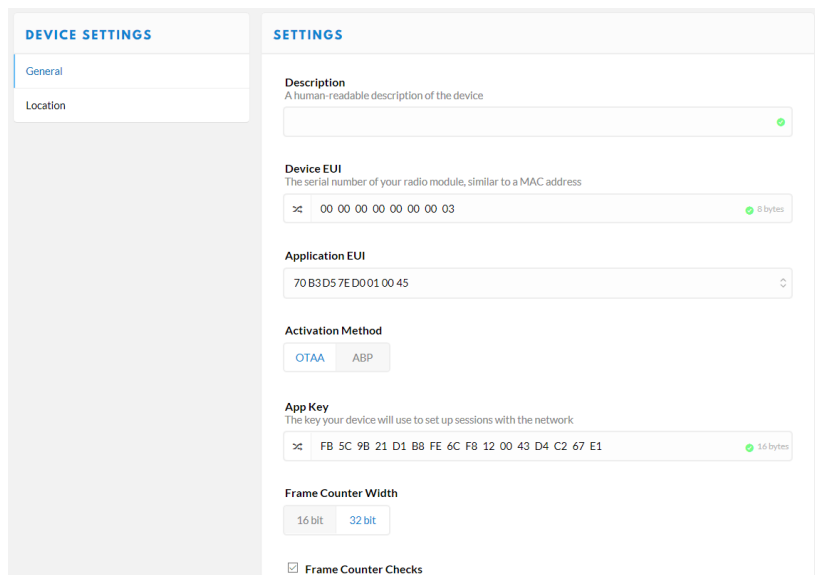
Once you've entered these values (note that the App key is created for you and the App EUI is set for this application) you can hit Register to register the device.

[Overview](#)
[Data](#)
[Settings](#)

DEVICE OVERVIEW

Application ID robservenvironmentalmonitor
Device ID sensor03
Activation Method OTAA
Device EUI
Application EUI
App Key
Status never seen
Frames up 0 [reset frame counters](#)
Frames down 0

The next thing we need to do is configure the node to use Access by Personalization (APB). Note that the device above is show as using the OTAA method. Click the Settings button at the top of the device properties to open the settings dialog for this node.



Click on APB under the Activation Method and then, to make development easier, uncheck the Frame Counter Checks checkbox at the bottom of the settings dialog. If this setting is checked the LoRa gateway will reject a packet if the gateway decides it has already seen a packet with that number. This can cause confusion during development, where you might set the packet counter to 0 each time you rebuild your software.

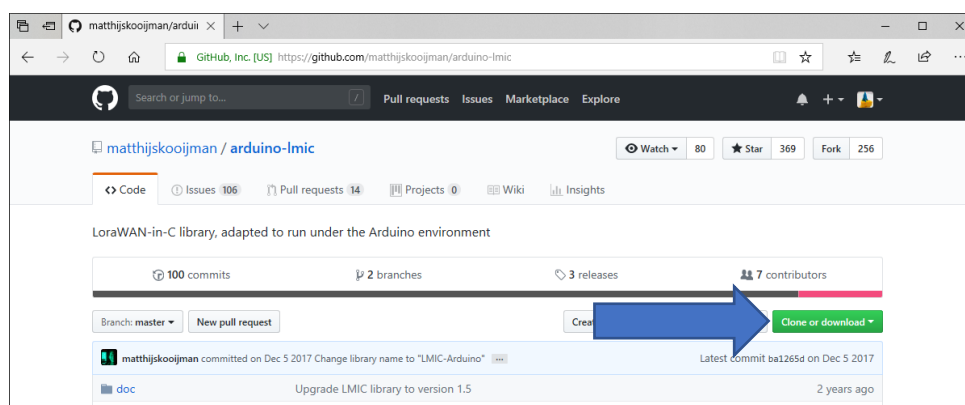
Once you have selected these options, scroll to the bottom of the settings dialog and click Save. Leave this page open, you'll be referring to it later.

Download the LMIC and Low Power libraries from GitHub

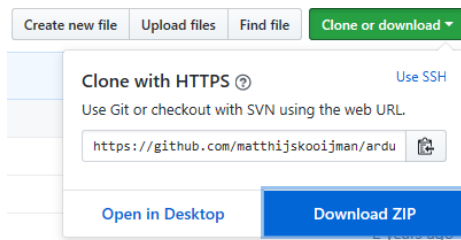
Now that the Things Network has a record for this node we have to create the Arduino software for the node and copy the APB settings into it. The software uses the LMIC and Low Power libraries, so the first thing we have to do is obtain these and install them. We only have to do this once for our computer.

Let's start with the LMIC library. Go to the GitHub page for the library. You can find it here:

<https://github.com/matthijskooijman/arduino-lmic>

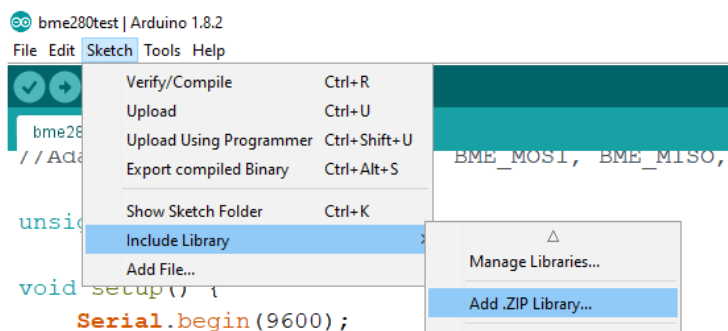


Click the "Clone or download" button and select Download ZIP.

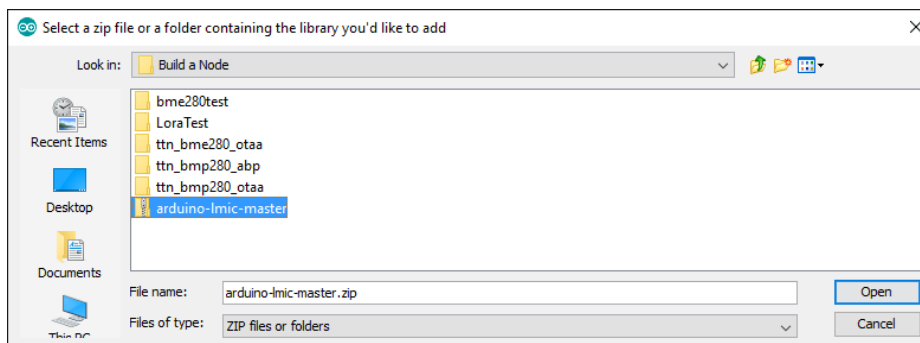


When you click Download Zip you'll be asked to specify a location for the download. It doesn't matter where you put this, but you should make sure you remember the location, because you will be searching for the file in a moment.

Once you have the file on your machine you need to add it to the Arduino libraries on your machine. Go to the Arduino IDE and Select 'Sketch->Include Library->Add .ZIP Library'



A dialog box will open where you can specify the library to add. Navigate to the folder where you downloaded the library, select it and click Open.



Now you need to repeat the process to download the low power library we are going to use. This contains code that will put the processor into a mode that uses hardly any battery power. At the end of this document you can find some hardware tips to reduce power consumption even more. You can find the low power library here:

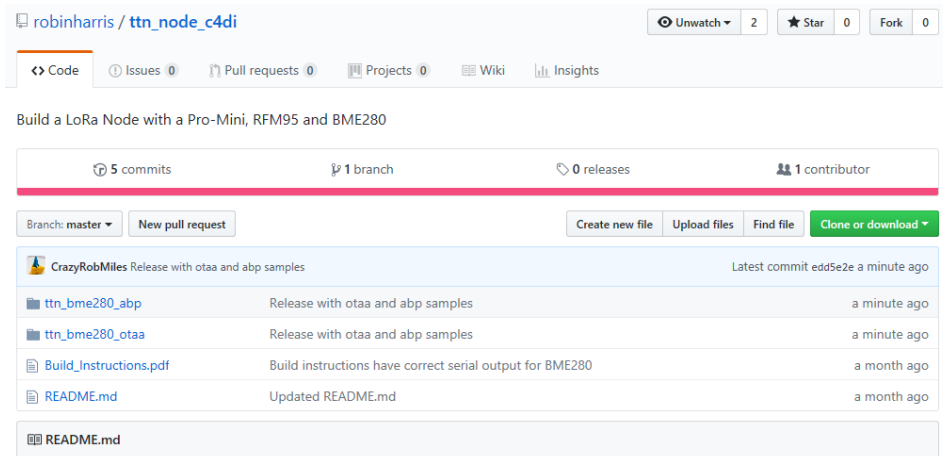
<https://github.com/rocketscream/Low-Power>

Download the Sample Programs

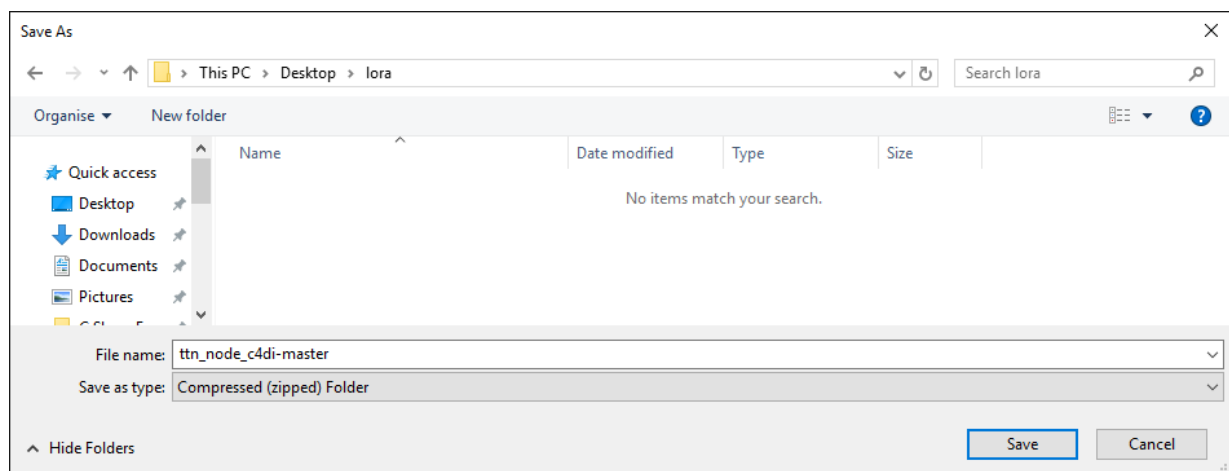
Now we need to download the programs. All the sample files, and this document are also on GitHub. You can download the ZIP and unzip the file to get the examples:

Go to https://github.com/robinharris/ttn_node_c4di

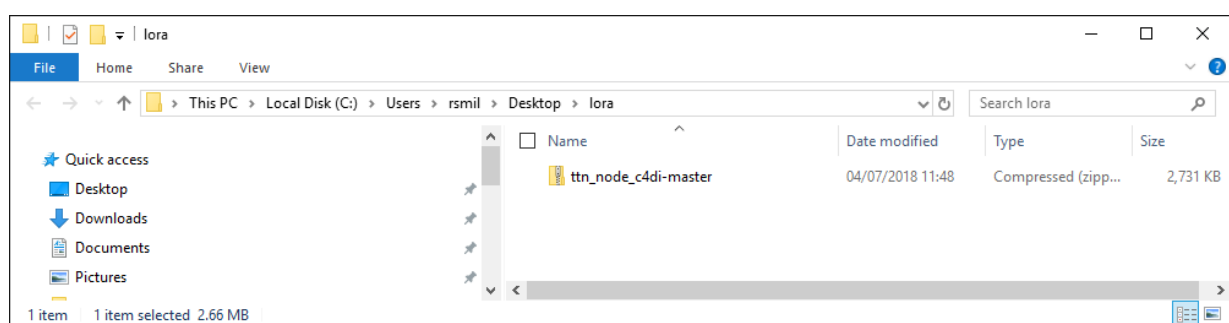
This work is licensed under the Creative Commons Attribution-NonCommercial 4.0 International License. Page 15



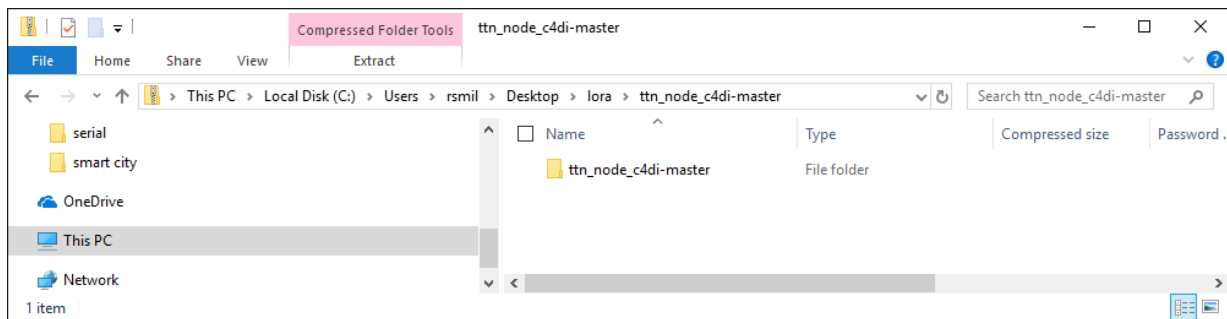
Choose 'Clone or download' as you did for the other library files and download the ZIP into a folder on your machine.



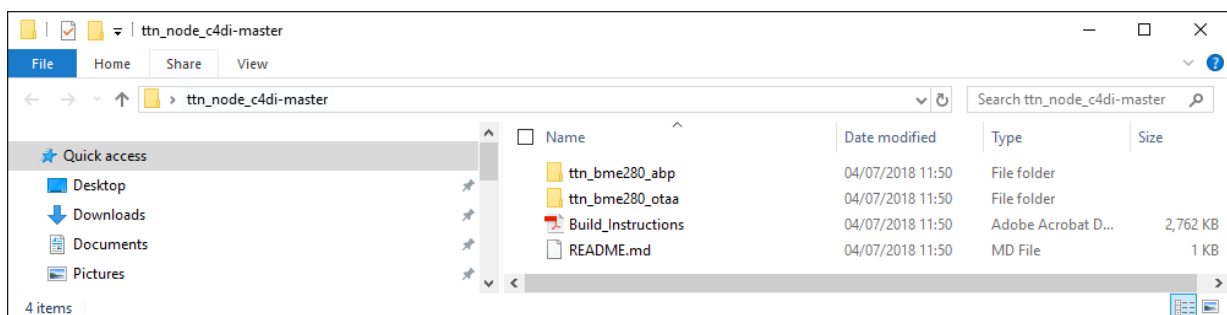
Navigate to the folder where the file was download.



Now double click on the archive to open it.



Drag the folder in the zip file onto your desktop to work on. This will unzip the files in the archive and make them ready for use. Navigate to the folder.



Navigate into the folder **ttn_bme280_abp** and open the example **ttn_bme280_abp.ino**. This should open the example program with the Arduino IDE.

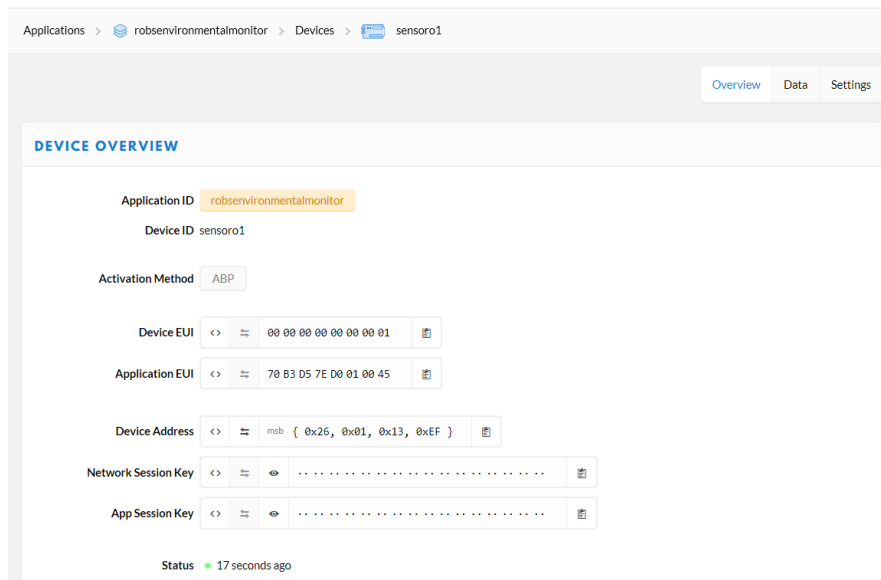
Two things to bear in mind about our sample code:

1. The code has already been configured so that it uses the BME280 devices at the correct address (0x76).
2. The baud rate for these programs is set to 115200 baud. This means that you will have to adjust the baud rate for the Serial Monitor if you want to see output from these programs.

Configure the Arduino software to access to The Things Network

Now we need to copy some values from the TTN configuration page into the Arduino code. This is the personalization part of Authentication By Personalization. Note that each new device that you add will have to have its own set of configuration values. You can use Over The Air Authentication if you want to avoid this, we'll mention this later in the document.

Navigate to the Device settings page in TTN for the device that you created earlier



Some of the values are only displayed as rows of dots. To understand how to use these values, these are the buttons you should use:



Click this to make the values in the settings visible



Click this to make the values turn into text that you can paste into programs

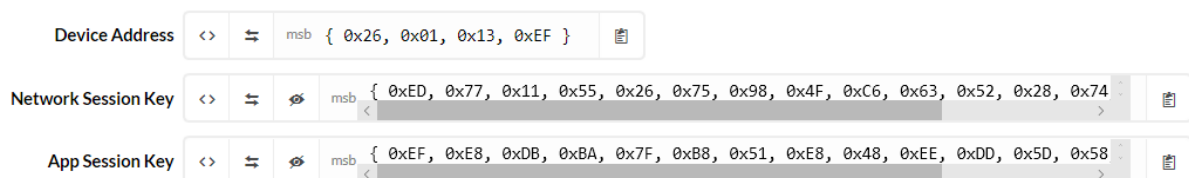


Click this to copy the text into the Windows clipboard, so that you can easily paste the into your program.



Click to reverse the byte order of the values as displayed, so that you can make sure that the values copied into the program are in the correct order. This so that we can work with different devices, which order values in memory in different ways. We want to use msb (most significant byte first) format for the values that we are pasting.

The screenshot below shows some values ready to copy into the program.



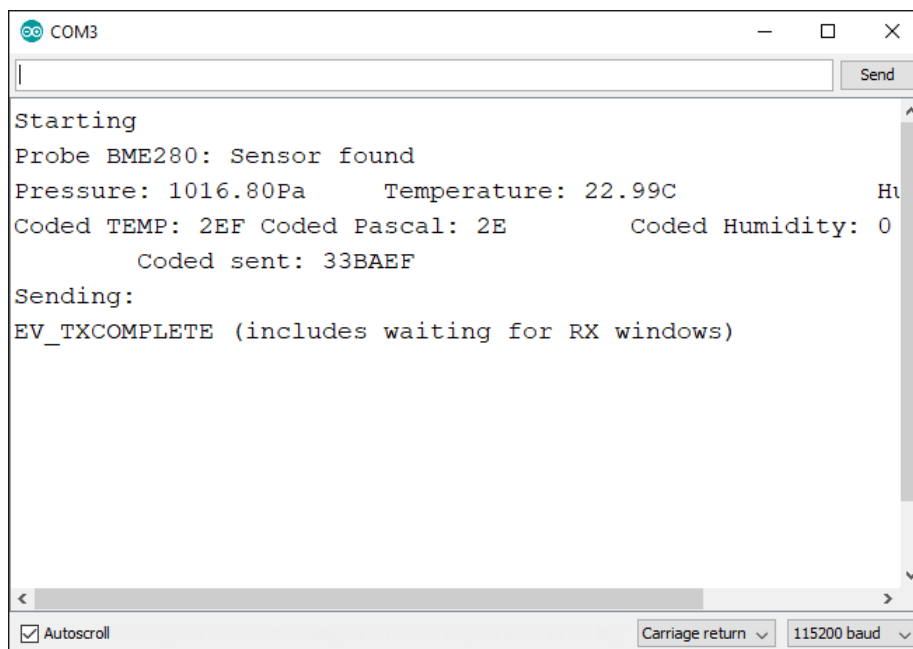
These values go into our program as shown below. The only other fiddly thing you need to remember is that to get the Device address you need to "scrunch" the hex digits together to make a single value.


```

65 // LoRaWAN end-device address (DevAddr)
66 // Copy this from the Device address key on TTN properties for the node
67 // Note that you have to "scrunch" the hex digits to make a single large value
68 static const uint_t DEVADDR = 0x260113EF; // <-- Change this address for every node!
69
70 // LoRaWAN NwkSKey, network session key
71 // Copy this from the Network Session Key entry on TTN properties for the
72 // node. Use the msb ordering
73 static const PROGMEM uint_t NWKSKY[16] = { 0xED, 0x77, 0x11, 0x55, 0x26, 0x75, 0x98, 0x4F,
74                                             0xC6, 0x63, 0x52, 0x28, 0x74, 0x43, 0xEA, 0xAF };
75 // LoRaWAN AppSKey, application session key
76 // Copy this from the App Session Key on TTN properties for the
77 // node. Use the msb ordering
78 static const PROGMEM uint_t APPSKY[16] = { 0xEF, 0xE8, 0xDB, 0xBA, 0x7F, 0xB8, 0x51, 0xE8,
79                                             0x48, 0xEE, 0xDD, 0x5D, 0x58, 0xD2, 0xAE, 0x50 };
80

```

These are the actual settings values of a node of mine, so please don't copy them verbatim. Once you have updated the settings you should be able to deploy the program to your node. To ensure the program is working correctly, open the serial monitor in the Arduino SDK and make sure that the node is sending messages:




```

COM3
Starting
Probe BME280: Sensor found
Pressure: 1016.80Pa      Temperature: 22.99C      Hu
Coded TEMP: 2EF Coded Pascal: 2E      Coded Humidity: 0
      Coded sent: 33BAEF
Sending:
EV_TXCOMPLETE (includes waiting for RX windows)

```

Now we can take a look in on the TTN website to see if the data has arrived. Select the Data view for your node to see the raw data as it arrives.



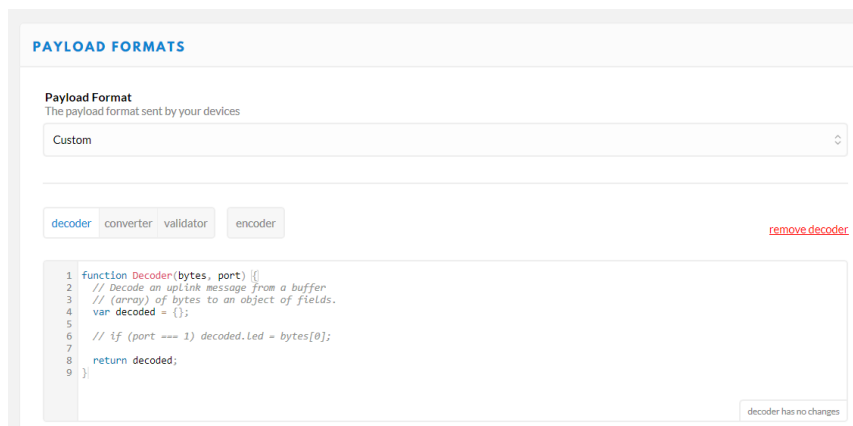
APPLICATION DATA				
Filters				
time	counter	port		
16:32:31	0	1	dev id: sensor01	payload: 31 BB 20

Note that the payload value is the same as the one that was printed by the Arduino program. This means that our data has made it through the connection. This is coded information, because the Arduino contains an algorithm that puts temperature, pressure and humidity into three bytes. This is

the code that does it, note that you don't have to know how this works, but it is here if you fancy finding out more.

```
temperature = bme280.readTemperature();
humidity = bme280.readHumidity();
pascal = bme280.readPressure();
pascal=pascal/100;
temperature = constrain(temperature,-24,40); //temp in range -24 to 40 (64 steps)
pascal=constrain(pascal,970,1034); //pressure in range 970 to 1034 (64 steps)*/
t_value=uint16_t((temperature*(100/6.25)+2400/6.25)); //0.0625 deg. steps + offset
// no negative values
p_value=uint16_t((pascal-970)/1); //1 mbar steps, offset 970.
s_value=(p_value<<10) + t_value; // putting the bits in the right place
h_value = uint8_t(humidity);
buffer[0]=s_value&0xFF; //lower byte
buffer[1]=s_value>>8; //higher byte
buffer[2]=h_value;
```

To get the right display format, we can create a decoder in our application that takes the encoded values and makes them into data to be stored by our applications. Select your application and open the 'Payload Formats' tab:



Enter the function below, overwriting the standard function:

```
function Decoder(bytes, port) {
  var humidity = bytes[2];
  var mbar = 970+((bytes[1]>>2) & 0x3F);
  var temperature = -2400 + 6.25*(((bytes[1]&0x03)<<8)|bytes[0]);
  // Decode an uplink message from a buffer
  // (array) of bytes to an object of fields.
  var decoded = { humidity: humidity, mbar:mbar, celcius:temperature/100};
  return decoded;
}
```

This function takes the bytes of data received from the node and uses then to create three values.

[decoder](#)
[converter](#)
[validator](#)
[encoder](#)

```
1 function Decoder(bytes, port) {
2   var humidity = bytes[2];
3   var mbar = 970+((bytes[1]>>2) & 0x3F);
4   var temperature = -2400 + 6.25*(((bytes[1]&0x03)<<8)|bytes[0]);
5   // Decode an uplink message from a buffer
6   // (array) of bytes to an object of fields.
7   var decoded = { humidity: humidity, mbar:mbar, celcius:temperature/100};
8   return decoded;
9 }
```

You can enter bytes into the web page and test them to see what results are produced. Note that the order of the bytes displayed when they are received is the reverse of the order in which they were transmitted.

Payload

31 BB 20

3 bytes

1

Test

```
{
  "celcius": 27.0625,
  "humidity": 32,
  "mbar": 1016
}
```

Payload was valid

If the function works OK you can return to your data and discover that incoming messages are now converted into their values.

APPLICATION DATA

|| pause

clear

Filters

uplink

downlink

activation

ack

error

timecounterport

▲ 16:46:56

14

1

dev id: [sensor01](#)

payload: 2BBB22

celcius: 26.6875

humidity: 34

mbar: 1016

▲ 16:45:54

13

1

dev id: [sensor01](#)

payload: 2ABB21

To find out more about encoding LoRa messages into sequences of bytes you can take a look at <https://github.com/thesolarnomad/lora-serialization> .

Sending messages to a node

To send bytes to our node we can use the 'Downlink' section of a device. In the console, click on your device and scroll down in the Overview page. Enter one byte (two hex digits, 05 for example) and click Send.

DOWNLINK

bytes
fields

FPort 1

05
1 byte

Send

The information will be send to the node after the next time the node contacts the gateway. With a sample rate of approx. 60 seconds, that is the maximum time this data will be received. The Led will blink this amount of times, with a maximum of 10 (0x0A).

Using Over The Air Authentication (optional)

If you don't want to have to personalize each node for the network you can use OTAA. The configuration process is identical, create a node description on The Things Network Site

Register the device

Now register your device. You have to define an address for yourself. This is a kind of MAC address, but because there is no registration yet of these, define some RANDOM 8 byte value, using your postcode and some values in between.

Device ID

This is the unique identifier for the device in this app. The device ID will be immutable.

sensor01

Device EUI

The device EUI is the unique identifier for this device on the network. You can change the EUI later.

56 29 AA BB 56 29 CC DD
8 bytes

App Key

The App Key will be used to secure the communication between you device and the network.

App Key will be generated

App EUI

70 B3 D5 7E F0 00 1B 79

DEVICE OVERVIEW

Application ID

fb_nodes

Device ID

sensor_01

Activation Method

OTAA


Device EUI

<> 56 29 AA BB 56 29 CC DE hex

Application EUI

<> 70 B3 D5 7E D0 00 17 BE hex

App Key

<>  hex

Device Address

<> 26 01 21 78 hex

There are three values you should copy into your Arduino Code. Dev EUI (the device identifier), App EUI (The application identifier) and the App key.

Dev EUI

With the <> sign you can select different 'views'. We need the '**LSB**' view, also called little Endian or Least Significant Byte First.

NOTE: For the App Key we need MSB, not LSB.

DEVICE OVERVIEW

Application ID

fb_nodes

Device ID

sensor_01

Activation Method

OTAA


Device EUI

<> { 0xDE, 0xCC, 0x29, 0x56, 0xBB, 0xAA, 0x29, 0x56 } lsb

Application EUI

<> { 0xBE, 0x17, 0x00, 0xD0, 0x7E, 0xD5, 0xB3, 0x70 } lsb

App Key

<>  hex

Device Address

<> 26 01 21 78 hex

Both Dev EUI and App EUI will be used in LSB or little Endian format.

First copy the Dev EUI with the clip board sign on the right. Paste the string into your code at the static const u1_t DEVEUI. Remember to respect the semicolon at the end of the line.


```
// This EUI must be in little-endian format, so least-significant-byte
// first. When copying an EUI from ttnctl output, this means to reverse
// the bytes. For TTN issued EUIs the last bytes should be 0xD5, 0xB3,
// 0x70.
```

```
static const u1_t DEVEUI[8] = { 0xDD, 0xCC, 0x29, 0x56, 0xBB, 0xAA, 0x29, 0x56 };
static const u1_t APPEUI[8] = { 0xBE, 0x17, 0x00, 0xD0, 0x7E, 0xD5, 0xB3, 0x70 };
```

```
// This key should be in big endian format (or, since it is not really a
// number but a block of memory, endianness does not really apply). In
// practice, a key taken from ttnctl can be copied as-is.
// The key shown here is the semtech default key.
```

APP EUI

Next copy APP EUI, select **LSB** for this one as well. Copy to APPEUI in your code.

APP KEY

The last key is a secret hash, so this one just shows when you click the 'EYE' icon. After that you can copy this to the clipboard. This one can be copied in **MSB** format.

Copy the 16 bytes APPKEY after static const u1_t APPKEY[16] = .

The BMP280 sensor should be connected!

Upload your code to the Arduino.

The LED will blink during the JOIN process. The Arduino gets his keys from TTN, thereby all information sent will be encrypted with these keys.

You can also follow this process by activating the 'data' view:

APPLICATION DATA						
<div>uplink</div> <div>downlink</div> <div>activation</div>						
	cntr	port	dev id	payload	fields	
▲ 22:56:24	2	1	sensor_01	D8 12		
▲ 22:55:08	1	1	sensor_01	D8 12		
▲ 22:54:03	0	1	sensor_01	D8 12		
⚡ 22:53:57			sensor_01	Activation		

After a few seconds, the LED will stop blinking. You will see the information coming in into the dashboard. This is coded information, because some algorithm is used to put temperature and humidity in two bytes.

To get the right display format, we can create a decoder in our application. Select your application and open the 'Payload Functions':

Speeding up joining the network

To help joining OTAA faster (seconds instead of 7 minutes) you may alter the LMIC library. You can find LMIC.C file in <My Documents>\Arduino\Libraries\Arduino-lmic-master\src\lmic

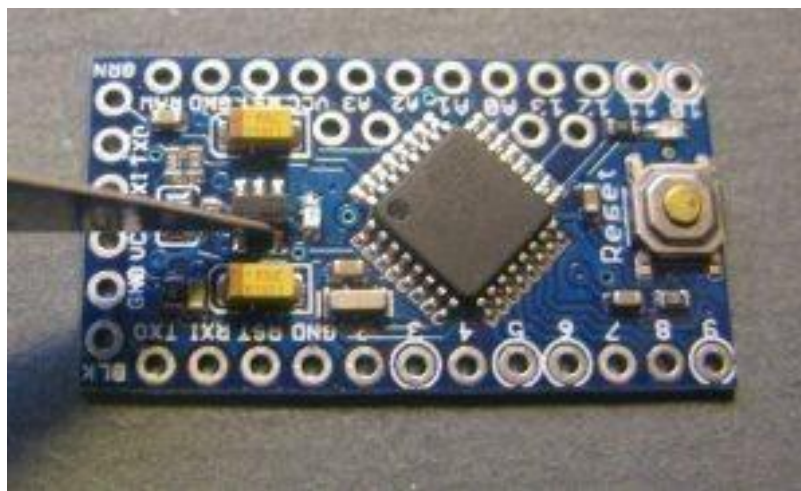
1. Open the file in your favourite source code browser (or use WordPad)
2. Go to line 684 (or find 'setDrJoin' to find the line below)
3. Change setDrJoin(DRCHG_SET, DR_SF7); to: setDrJoin(DRCHG_SET, DR_SF9);

Low Power Arduino (Optional)

There is an option to 'low power' the Arduino. BUT this requires some precise soldering! The setup will work without removing the components. If you are a novice where soldering is concerned, skip the first part! It requires a sharp knife to remove the power converter, and to cut the connections for the two on board leds. If you want to battery power your node for a longer time (weeks or more) this is the option for you. Current in sleep mode will be reduced to <100uA. Look at the instructions at the end of this manual

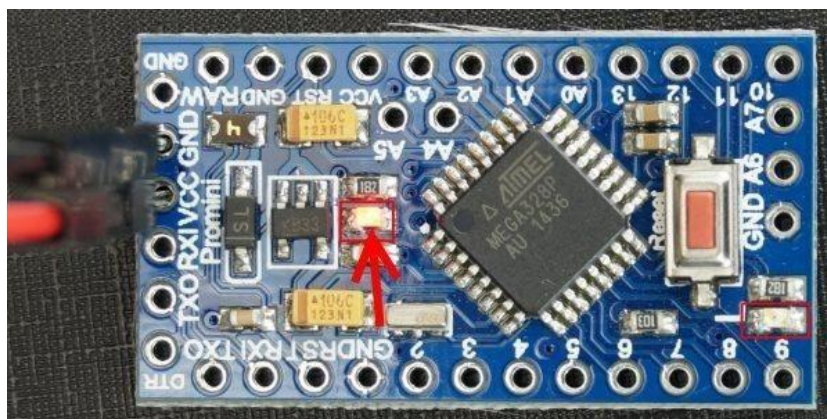
Remove the Regulator

The easiest way to remove the regulator and not damage the board is to use a very sharp scalpel to cut through the regulator leads at the point they join the regulator body.



Remove the LEDs

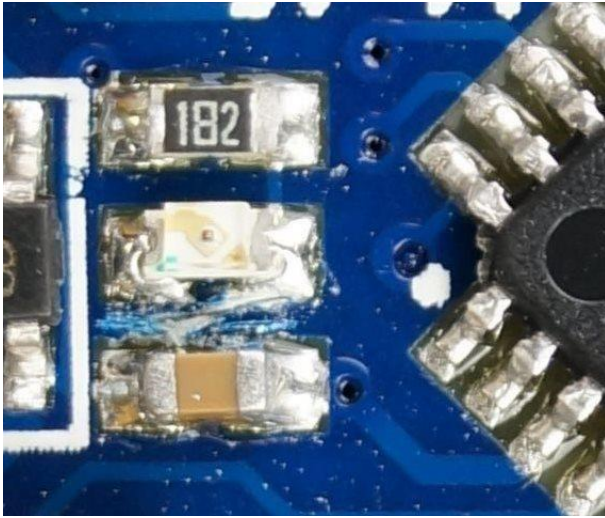
There are two LEDs on the Pro Mini board. The two LEDs are marked with a red square in the following picture. The power LED is marked with an arrow. If you are not sure where the power LED is on your board, then you can just power it and you will see the LED.



When you found the power LED, then try to locate at least one trace that leads to the LED. In the second picture below, I marked the traces on my board. A high-resolution picture with a lot of light helps to find the traces.

When you found a trace to the power LED, then you take a knife and break the trace, so that it will not conduct any more. You can see my result in the third picture below.

This work is licensed under the Creative Commons Attribution-NonCommercial 4.0 International License. Page 26



Removing the LEDs can be tricky, so it's easier to remove the series resistors for the LEDs instead. This version of Pro Mini also has a resistor feedback network for the regulator across VCC, these consume power so should be removed. Just push the resistors aside with a soldering iron. The picture shows the Pro Mini with unwanted parts removed, locations of the removed components are circled in red.

You can find out more at:

<https://andreasrohner.at/posts/Electronics/How-to-modify-an-Arduino-Pro-Mini-clone-for-low-power-consumption/>