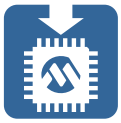
21080 IoT3

Connecting your IoT Device with LoRaWAN™ to The Things Network A Global IoT Data Network

Lab Manual



Heath Marvin, Luc Archambault, Gregory Demont
Microchip Technology Inc.

MICROCHIP
MASTERs Conference

Connecting your IoT Device with LoRaWAN™ to the Things Network A Global IoT Data Network

Table of Contents

Reference Materials

All the documents and solutions of the lab exercises may be found in the MASTERs folder.
It is located at: **C:\MASTERS\21080**

Introduction

This LAB Manual is intended to be used in the classroom as well as at home after MASTERs.
For the classrooms, it is assumed that everything you need is already installed on the computers. For home use, the installation instructions can be found at the end of the labs under the heading "Appendix".

Lab Equipment

For this lab, you will need the following tools:

Hardware

A. Student desk

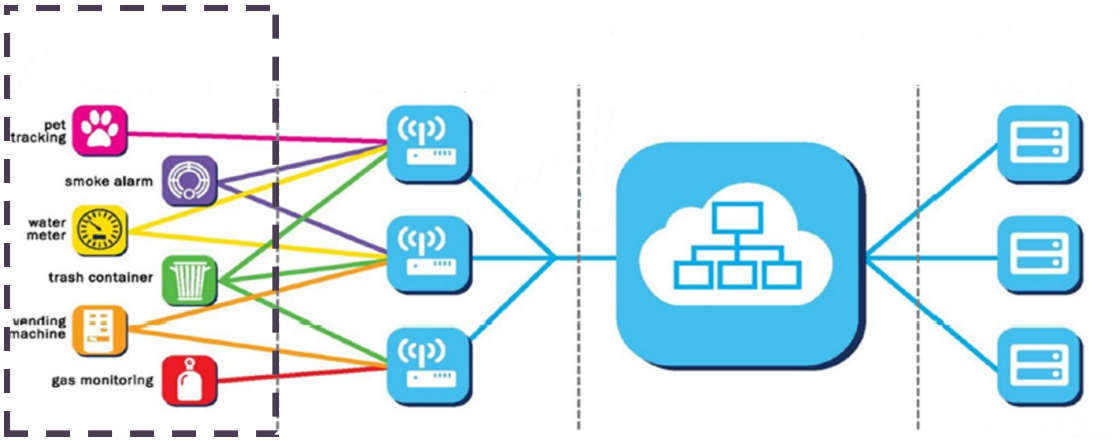
SODAQ ExpLoRer Starter Kit, which contains the RN2903A module
Personal computer with an USB port
Micro-USB cable to interface the Starter Kit to the PC

B. Presenter desk

The Things Gateway

Software

- Arduino IDE 1.8.1 or higher (http://www.arduino.cc)



Lab 1

Collect Data on the ExpLoRer Starter Kit

Purpose

Create simple Arduino Sketch with code that:

- Collects temperature data from the embedded temperature sensor
- Collects devEUI and appKey of the RN2903 LoRaWAN module
- Displays temperature and activation keys on the Serial Monitor

Requirements

Development Environment:	Arduino IDE 1.8.1 or above
Hardware Tools:	ExpLoRer Starter Kit
Lab files on class PC:	C:\MASTERS\21080\Lab1\...

Your lab PC has the Arduino IDE already installed. If you are following these steps on your own PC, you'll need to follow the steps in Appendix B to install the correct software.

Objective

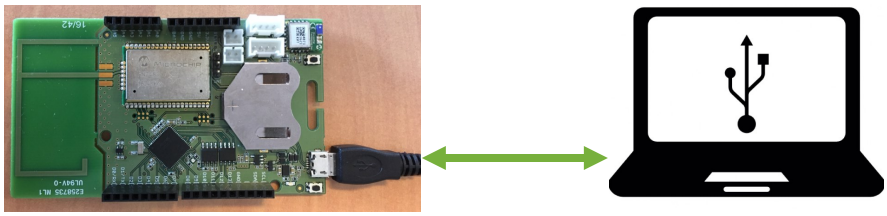
For the first steps of this lab, the instructor will walk you through the process of compiling, uploading, and running the provided code in order to verify that your setup is correct. Then, you will add code to read temperature data from the MCP9700 Temperature Sensor, collects the activation keys from the RN2903 LoRaWAN module which are needed for the next labs and finally displays the data in the Serial Monitor.

Follow-Along Steps

The first steps of Lab 1 are follow-along. Follow the steps along with the instructor to verify your setup.

Step 1: Hardware Installation

a. **Connect** your ExpLoRer Starter Kit to the PC through the micro-USB cable.
b. **Wait** for USB driver installation and COM port mounting. The USB port powers the board and enables the user to communicate with the kit.



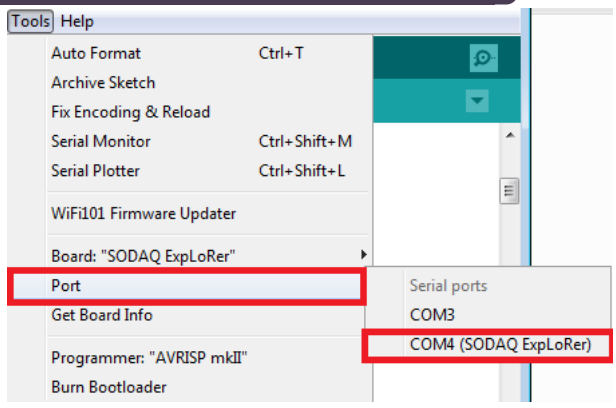
Step 2: Start Arduino IDE and open the sketch file

a. **Launch** Arduino IDE from the desktop
b. **Open** Lab1 by selecting from the menu: **File ▸ Open …** and selecting the Sketch file located at:

C:\MASTERS\21080\Lab1\Lab1.ino

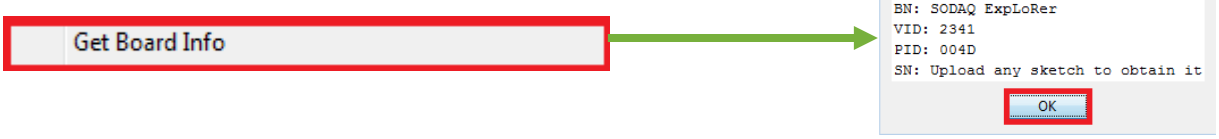
Step 3: Check the communication with the kit

a. **Select** the mounted COM port from the menu:
Tools ▸ Port ▸ COMx (SODAQ ExpLoRer)
The name of the board should appear in bracket.
Make sure to select the COM port referring to the SODAQ ExpLoRer.
Note that the current hardware setup; board name and serial port; should be visible at the bottom right corner of the IDE. SODAQ ExpLoRer on COM4



Make sure you have selected the SODAQ ExpLoRer Boards as well.

b. **Check** the communication by selecting **Tools ▸ Get Board Info**



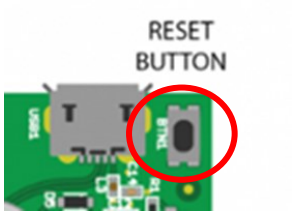
Step 4: Verify/compile

a. **Verify/compile** the sketch by clicking the toolbar button.
The first step to getting a sketch ready for transfer over to the Arduino is to Verify/Compile it. That means check it over for mistakes and then translate it into an application that is compatible with the hardware.

Verify/compile

Step 6: Upload the sketch to the board

a. **Press** the reset button (near the USB connector) twice within a second to place the ExpLoRer board into bootloader mode. Note that a new COM port dedicated for uploading the sketch is mounted.



b. **Select** the new mounted COM port from the menu:
Tools ▸ Port ▸ COMx (SODAQ ExpLoRer)

c. **Upload** the sketch to the board by clicking the toolbar button.

Upload

Step 7: Use the Serial Monitor

After the programming process, the board is resetting with the original COM port.

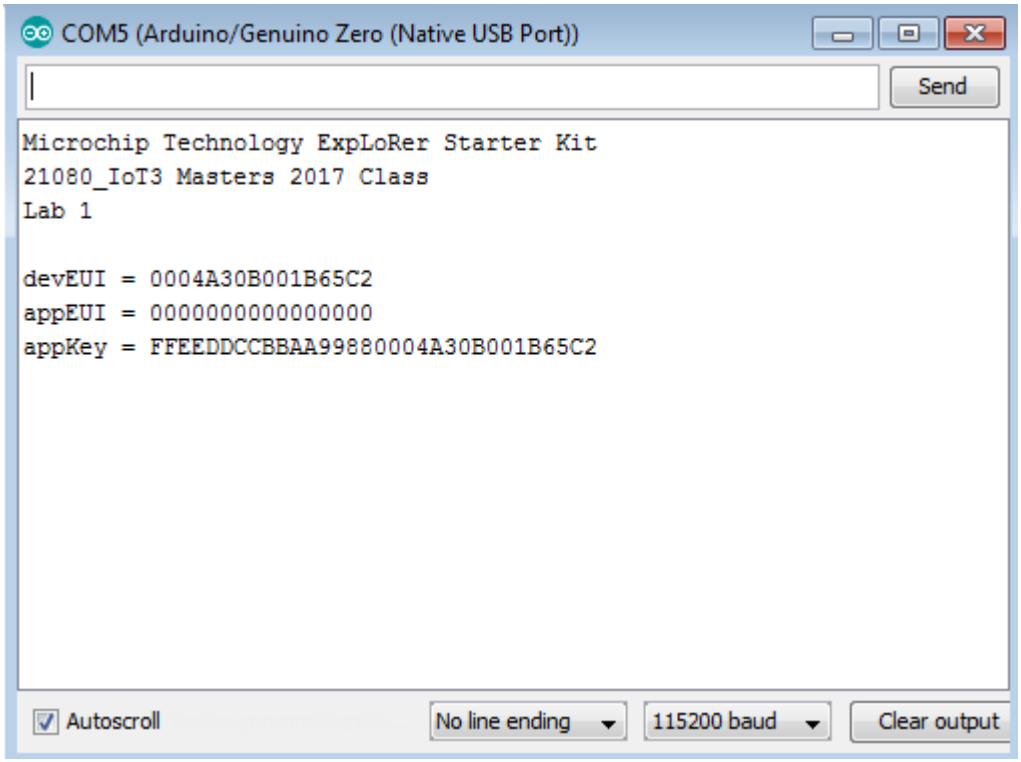
a. **Select** the previous COM port from the menu:
Tools ▸ Port ▸ COMx (SODAQ ExpLoRer)

b. **Open** the Serial Monitor window by clicking the toolbar button

Serial Monitor

Follow-Along Results

If your setup is correct, you will see a message displayed on the terminal



On Your Own — Collect and display data

Step 8: Collect devEUI and appKey

Write your unique keys displayed on the terminal like this:

00	04	A3	0B	00	1A	25	08

These keys will be needed for the **Lab 2.**

devEUI (8-bytes)

appKey (16-bytes)

On Your Own — Collect and display data

Step 9: Collect data from the temperature sensor

The ExpLoRer Starter Kit integrates a low-power linear active thermistor IC, the MCP9700A, has an accuracy of +/- 2°C from 0°C to +70°C while consuming 6uA (typical) of operating current. The sensor generates output voltage in proportion to temperature. Output voltage is given with the following formula: Vout = Tc x Ta + V0

Where:
⇒ Vout = Sensor output voltage
⇒ Tc = Constant value that increase output in every 1.0°C (sensitivity equals 10mV/°C)
⇒ Ta = Ambient temperature
⇒ V0 = Sensor output voltage at 0°C (500mV according to the datasheet)

Therefore, temperature can be calculated from the following formula: Ta = (Vout - V0) / Tc
Measuring Vout using analog to digital converter let us know the ambient temperature.

The MCP9700A IC is connected to the SAMD21 microcontroller through the pin PA4. The name TEMP_SENSOR has been given to this pin. Please refer to the Appendix A—Hardware Information to see the list of pin definition.

Inside the setup() routine, the pin TEMP_SENSOR is configuring to behave as an input.

```
pinMode(TEMP_SENSOR, INPUT) ;
```

The ExpLoRer board have 12-bit ADC capabilities: `analogReadResolution(12) ;`
This will return values from analogRead() function between 0 and 4095.

```
void setup()
{
  while (!debugSerial) ;
  debugSerial.begin(115200) ;
  debugSerial.println("Microchip Technology ExpLoRer Starter Kit") ;
  debugSerial.println("21080_IoT3 Masters 2017 Class") ;
  debugSerial.println("Lab 1") ;

  // ------------------------------------------------------------------------
  // Init section
  // ------------------------------------------------------------------------
  // Temperature sensor
  pinMode(TEMP_SENSOR, INPUT) ;
  analogReadResolution(12) ;
```

On Your Own — Collect and display data

Step 9: Collect data from the temperature sensor

We will construct the code **step by step** to read temperature from the sensor.
Your code will be added in the loop() function.

Replace the "`// ### Your code here ###`" comments with the necessary code to complete the data collection from the temperature sensor.

```
void loop()
{

  …

  // Step 9.1 Read value from analog input
  // ### Your code here ###
  // Step 9.2 Convert the value to voltage
  // ### Your code here ###
  // Step 9.3 Calibrate to 0°C
  // ### Your code here ###
  // Step 9.4 Convert to temperature value
  // ### Your code here ###
  // Step 9.5 Display the temperature on the serial monitor ###
  // ### Your code here ###
  // Step 9.6 Change to a 3 sec delay before reading a new value
  delay(10000) ;
}
```

Step 9.1: Read value from analog input

First, you need to **establish a variable** to store the raw value which will be between 0 and 4095 coming from the temperature sensor. This is perfect for an `int` datatype.

```
int sensorValue = analogRead(TEMP_SENSOR) ;
```

Step 9.2: Convert the value to voltage

The A/D sampling on the ExpLoRer board can detect 0-3.3 Volts in 4096 steps. Thus, the 12-bit ADC with a maximum input of 3.3VDC can resolve the measurement into 3.3/4096 = 0.81mV. The ADC resolution is the smallest distinguishable change in input. The resolution is the change in input that causes the digital output to change by 1. So you'll need to **create another variable**, a `float`, and do a little math to scale the value between the full scale (0.0 and 3.3), divide the maximum (3.3) by 4096 and multiply that by `sensorValue`. For ease of reading, we will use millivolt convention. Note that dividing by 4096 would be interpreted by compiler as integer operation and the result in this case will equal 0. Because of this, you'll need to divide by 4096.0 for the compiler to treat the division as a floating point operation.

```
float mVolts = (float)sensorValue * 3300 / 4096.0 ;
```

On Your Own — Collect and display data

Step 9.3: Calibrate to 0°C

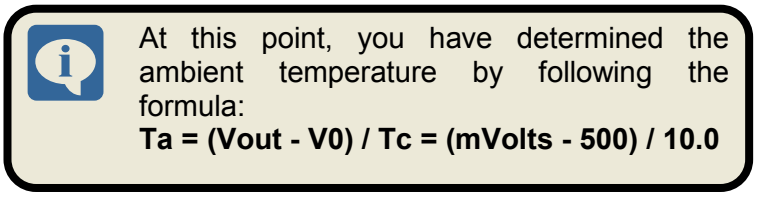
In the previous step, we have determined the sensor output voltage (Vout) in millivolts. According to the datasheet of the MCP9700A device, the sensor output voltage equal 500mV at 0°C (V0).The difference between voltage reading from the sensor and 500mV is linearly dependent on temperature. Let's **add** another `float` variable to calculate this difference.

```
float temp = (mVolts – 500) ;
```

Step 9.4: Convert to temperature value

The previous calculated value must be **divided** by 10.0 mV/°C, which is the temperature coefficient of the MCP9700A device, to finally obtain the ambient temperature value.

```
temp = temp / 10.0 ;
```



Step 9.5: Display the temperature on the serial monitor

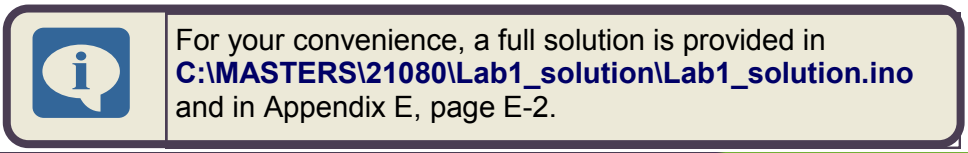
You need to **print** this information to your serial window. You can do this with the command `Serial.printlnt()`. Notice the ExpLoRer board has 4 hardware serials. One of them is `SerialUSB` for printing out data over the USB cable. For your convenience, the serial port name has been defined with the name `debugSerial`. **By convention, we will always use `SerialUSB` for debug purpose over the USB cable in the sketch file.**

```
debugSerial.println(temp) ;
```

Step 9.6: Add a 3 sec delay before reading a new value

You will **pause** the program for 3 seconds instead of 10 seconds. The method `delay(ms)` should be use to pause the program for a certain amount of time in milliseconds.

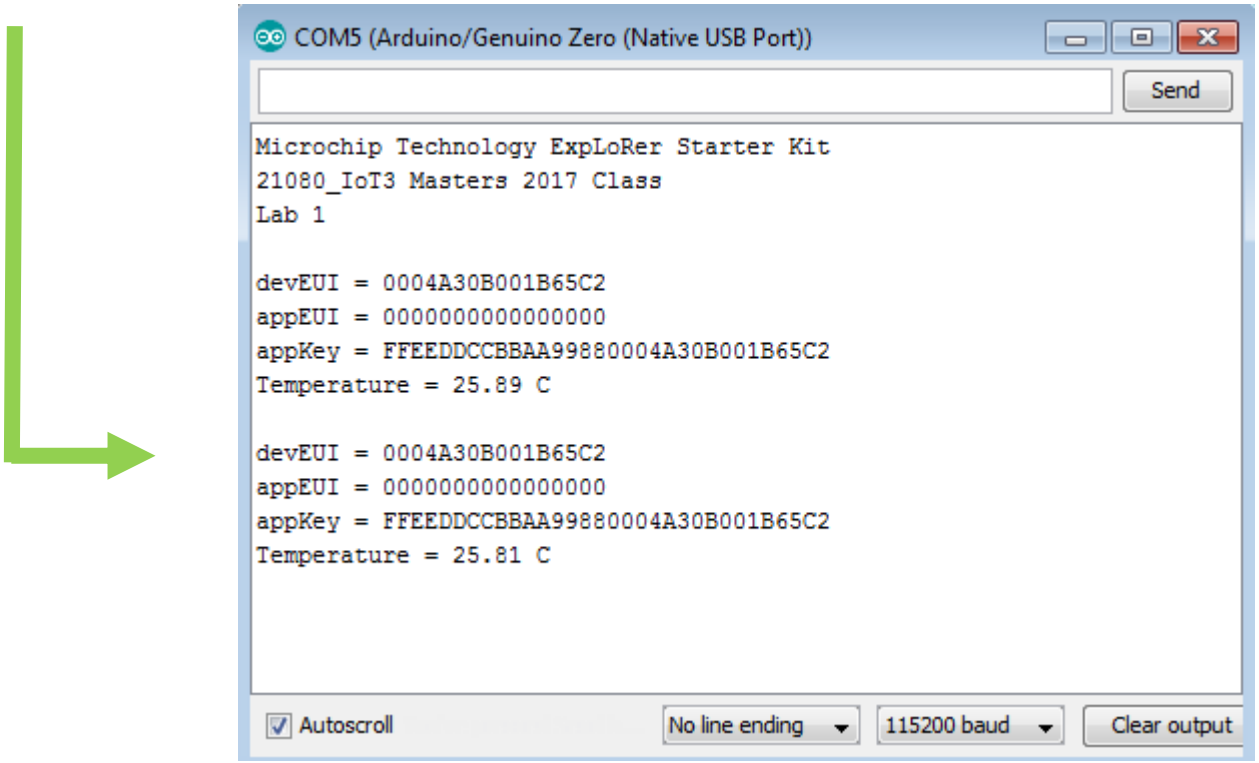
```
delay(3000) ;
```



Just like in steps 4 to 7, **compile** the Sketch and **program** the ExpLoRer board. **Open** the serial monitor to **view data** sent by your working Arduino Software.

Results

You should see the current ambient temperature displayed on the terminal window and the value is updated every 3 seconds.

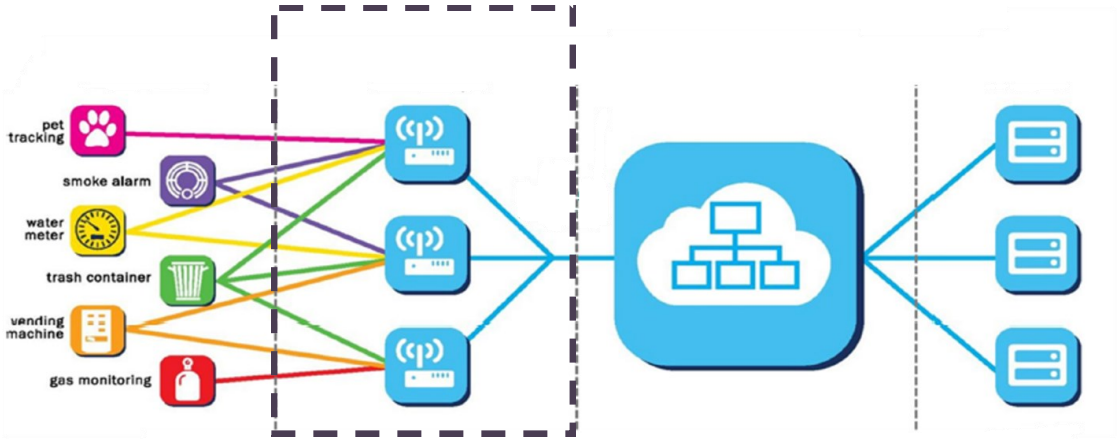


Conclusions

At this point, you should be familiar with the steps necessary to write a simple Sketch and program it to the ExpLoRer board.

What we've done so far:

- Collects temperature data from the embedded temperature sensor
- Collects devEUI and appKey of the LoRaWAN module
- Displays temperature and activation keys on the Serial Monitor



Lab 2

Provision your end node device within your TTN account

Purpose

Getting familiar with The Things Network

- Create a TTN account
- Create a LoRaWAN application
- Create a LoRaWAN device
- Register a gateway

Requirements

Development Environment: Mainstream browser like Google Chrome or Mozilla Firefox
Hardware Tools: Computer

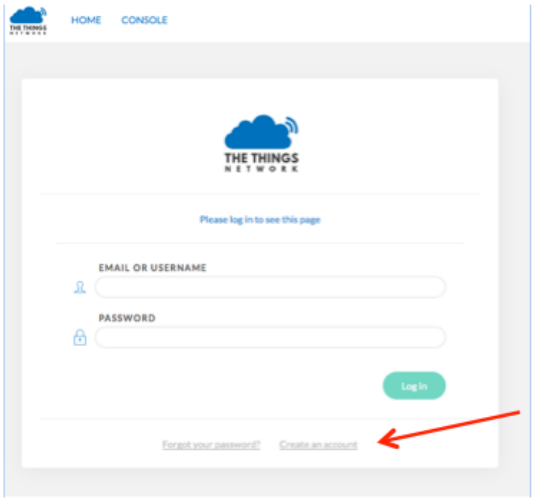
Objective

In this Lab, you will get familiar to The Things Network web site. You will first create your own account. Then you will create a unique Application and a new device that will allow you to connect the Arduino ExpLoRer board that you have used in Lab1.

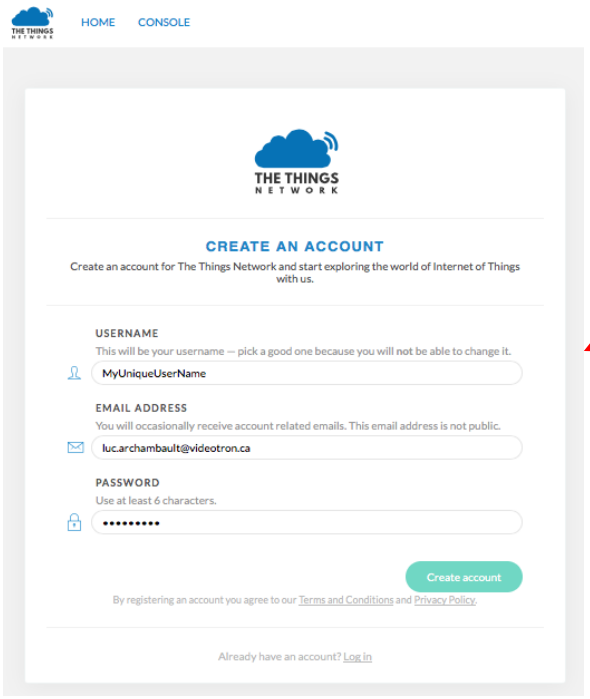
Create a TTN account

Step 1: Create a TTN account

a. **Go to** https://account.thethingsnetwork.org/users/login, and click on “Create an account”.

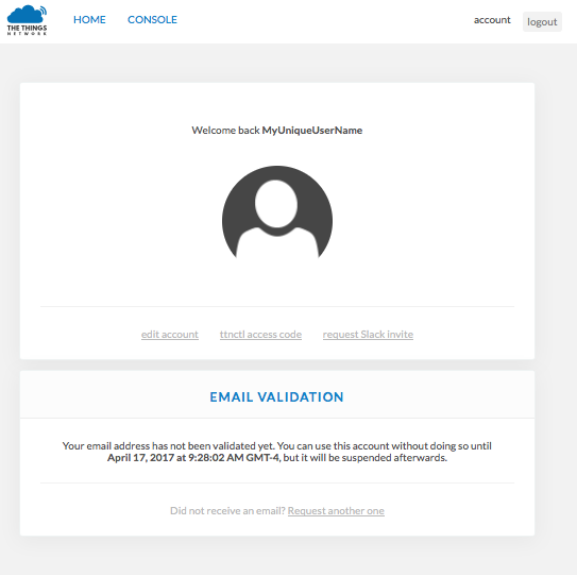


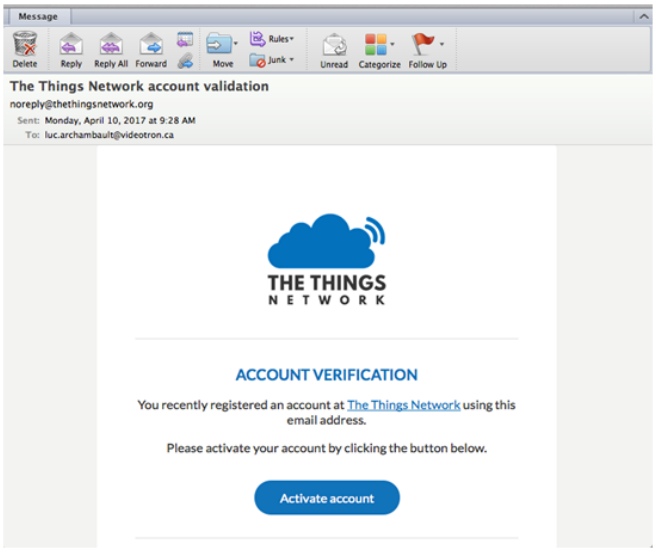
b. **Enter** your email address and password
c. **Enter** a unique “USERNAME”. You will not be able to change it after and it has to be unique within TTN network.



Unique USERNAME

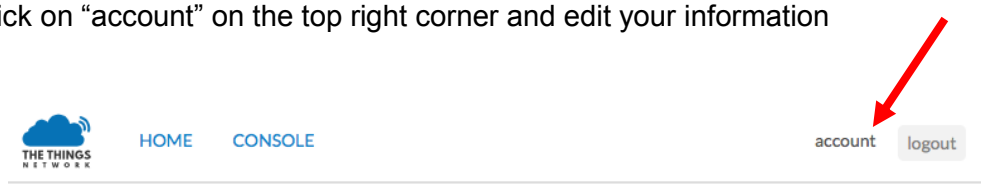
d. You will receive an email to validate your account. **You do NOT need to activate your account immediately**. It is available for 1 week without activation.

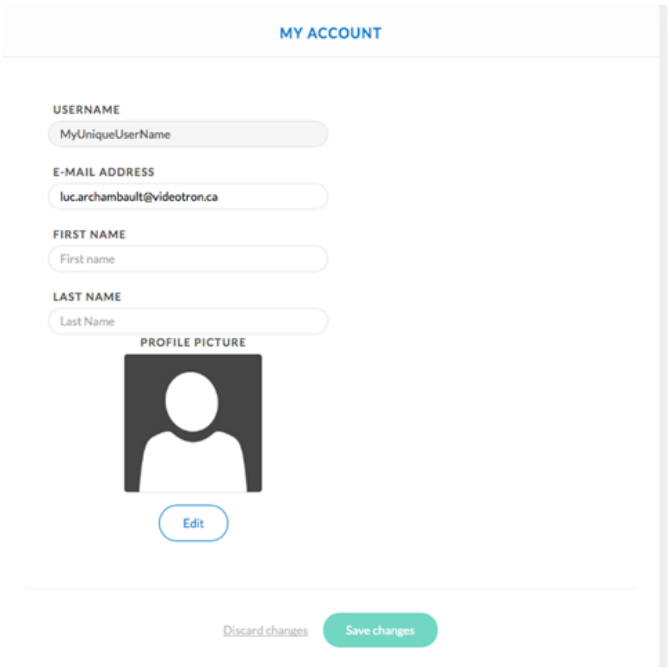




e. **Update your account information.**

1. Click on “account” on the top right corner and edit your information

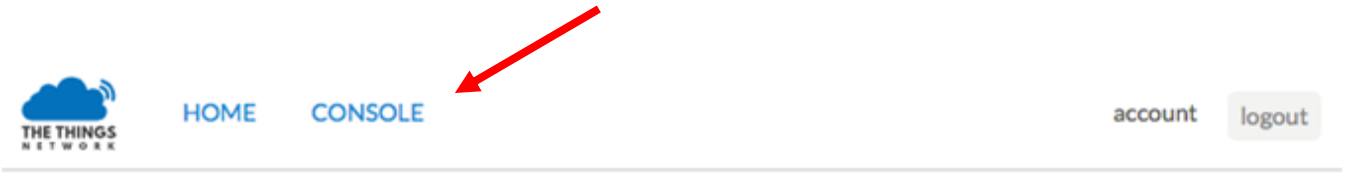




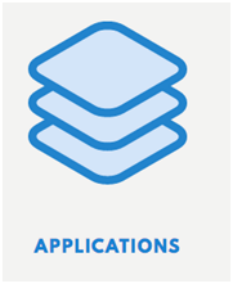
Create a new Application TTN account

Step 2: Create a new Application

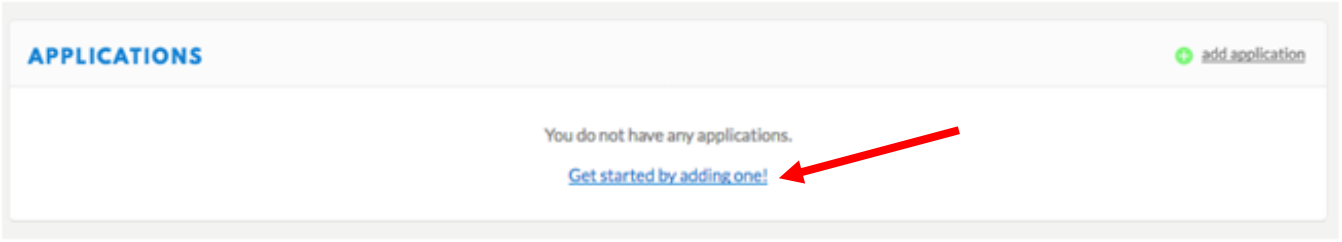
a. **Click** on the “Console” link.



b. **Click** on “Application” Icon.

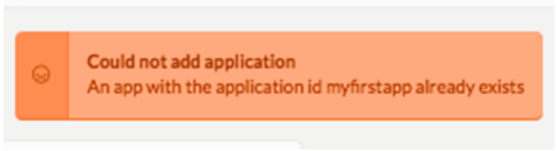


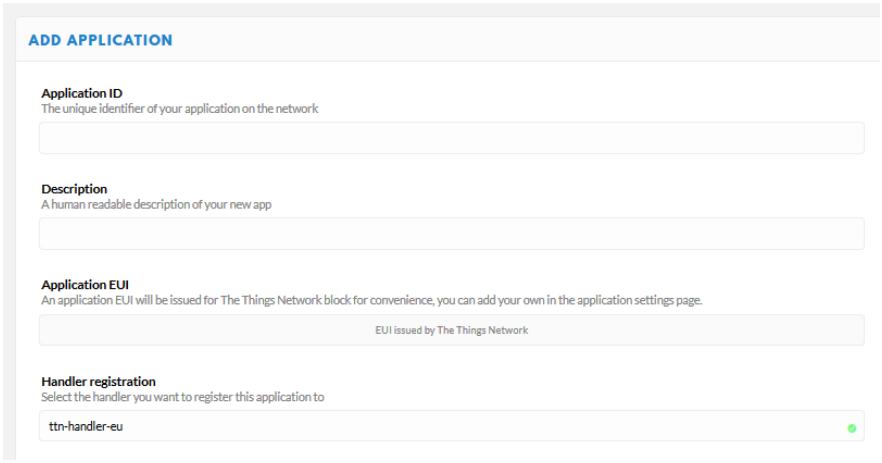
c. **Click** on the “Get stared by adding one!” link.



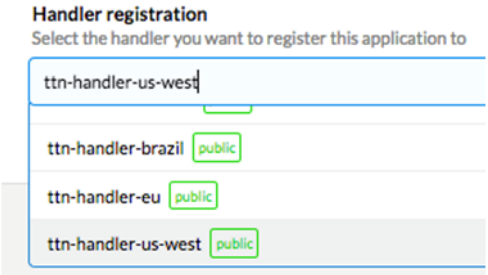
c. **Enter** information in the first three text boxes.

NOTE: For Application ID, name must be **unique** to TTN Server. So if you choose a name like "myfirstapp", most certainly someone else would have chosen the same name and it will fail. Be innovative ;-)

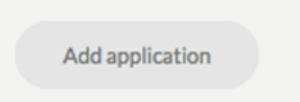




For "Handler registration", **select** => **ttn-handler–us-west**

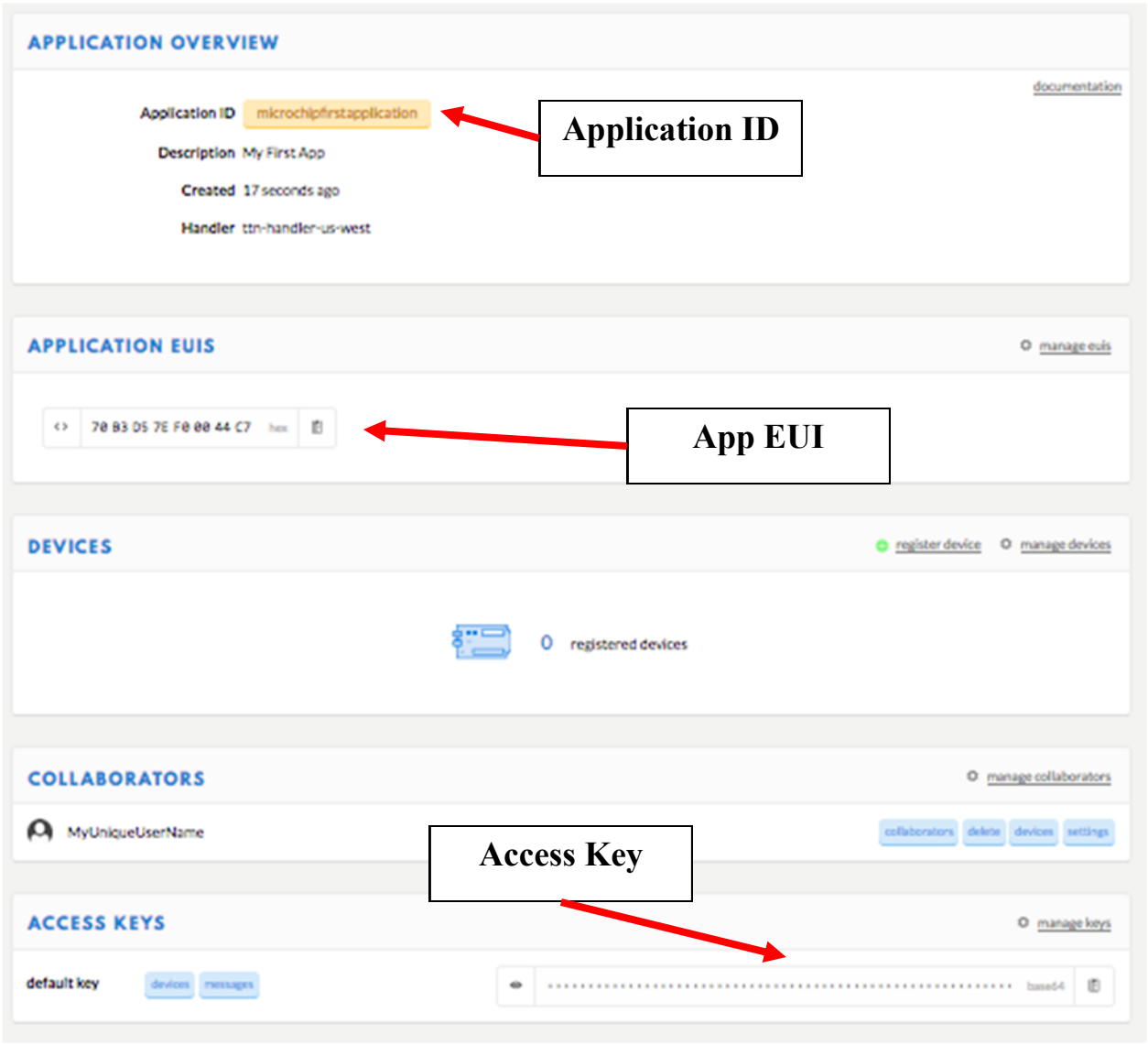


Click "Add Application"



Notice the information on the page following the newly created application. Click on the “ ” next to hidden keys to unveiled the information.

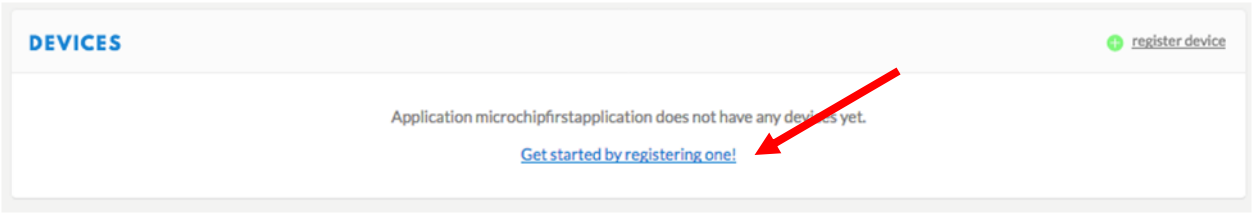
The “**Application ID**” and the “**Access Key**” are the credential required later in Labs 4 and 5 when using Node-RED.



Create a new Device

Step 3: Create a new Device

a. **Add** one device, by clicking “get started by registering one”:



Enter information:

For “**Device ID**”, readable name unique ONLY in this APP.

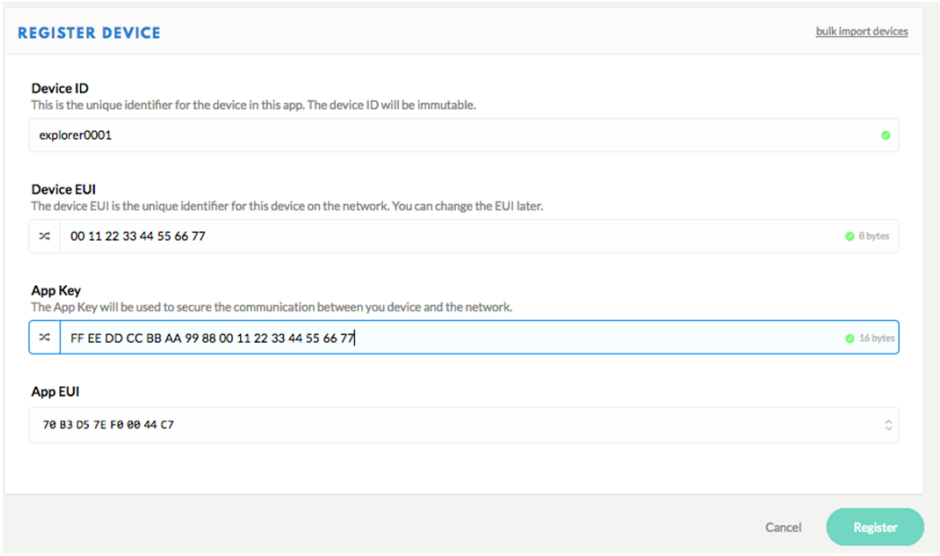
For “**Device EUI**”, enter devEUI from Lab 1, page 1-4

For “**AppKey**”, enter appKey from Lab 1, page 1-4 by clicking the little pencil on the left of the text box.

Note: For your own design, you could let TTN define an AppKey for you or define your own. You want to keep your AppKey secure, so it is good to use a hardware security chip (like the ECC508 included on the ExpLoRer board) to store keys like this.

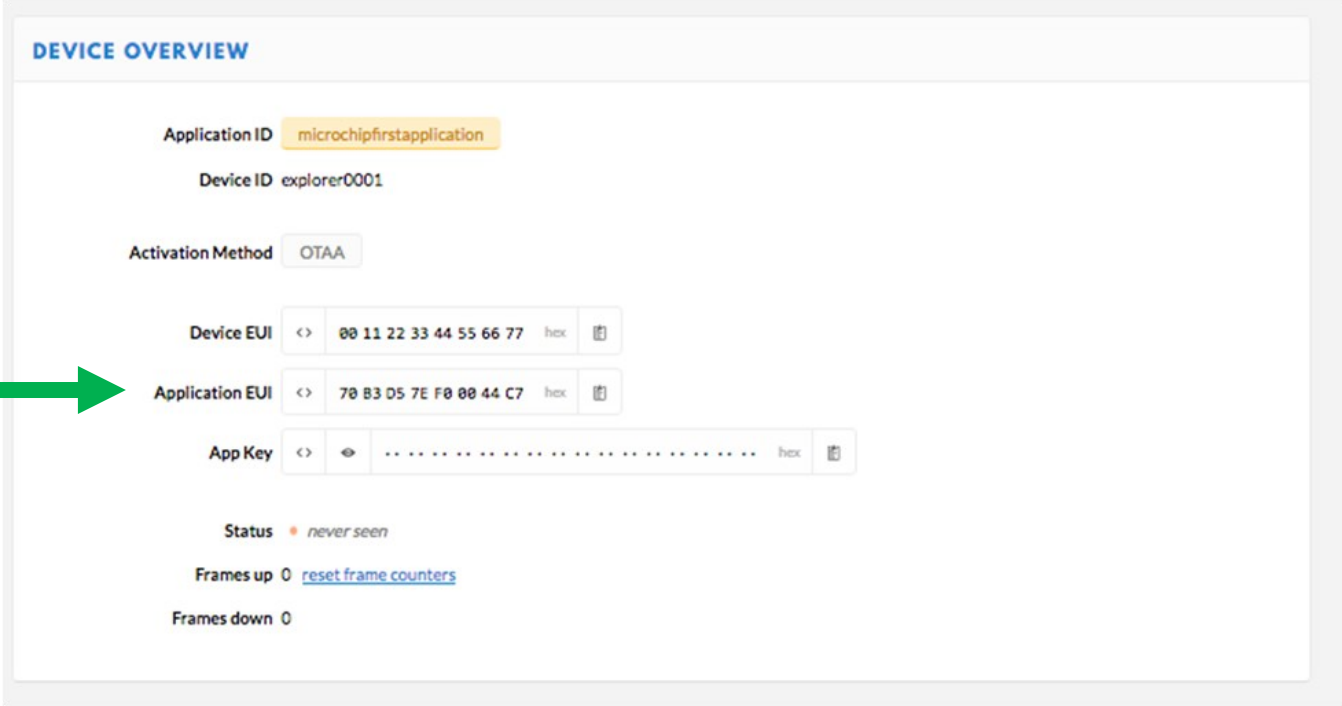
For the lab and to create unique AppKeys, we decided to use the first 8 bytes “FF EE DD CC BB AA 99 88” as our Unique ID and the last 8 bytes as the “**Device EUI**”.

“**Application EUI**” is already defined. This will also need to be copied in the Arduino sketch in LAB3.



Create a new Device

After clicking “**Register**”, you should see the information below with status “never seen”.
In LAB3 you will activate the device.



Collect Application EUI

Write here your unique Application EUI generated by TTN. These key will be needed for the **Lab 3.**

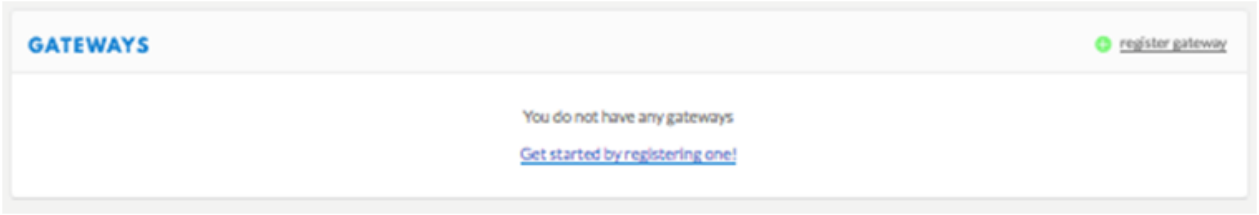
appEUI (8-bytes)

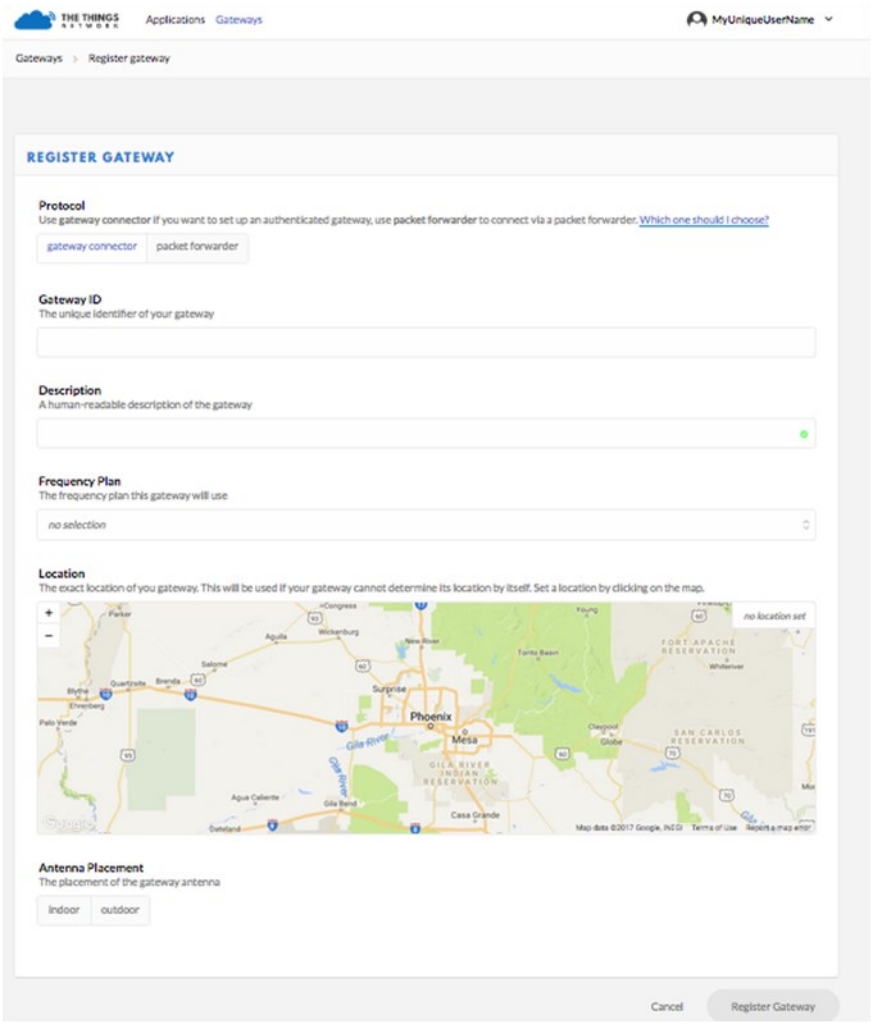
At this point, you have all the LoRa keys to join a network. So far, you have devEUI, appEUI and appKey.

Explore TTN Gateway

Go back to https://console.thethingsnetwork.org/

Click on the Gateway ICON. This is where you can register your own Gateway by entering the settings. For this class, you don't need to register a new gateway because we are connecting to the presenter's gateway. To have more information on how to register a gateway, go to Appendix

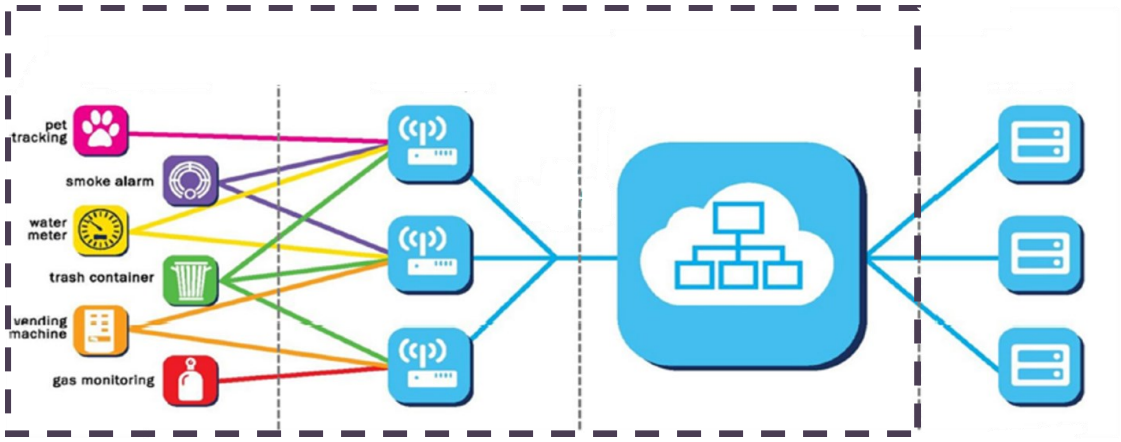




Conclusions

Now you should be familiar with provisioning your end node device within TTN.
What we've done so far:

⇒ Create a TTN account to connect your device to TTN network server
⇒ Create applications in the TTN console to collect data from your device
⇒ Each device must be registered within an application
⇒ Register your own gateway or connect to existing gateway



Lab 3

Connect ExpLoRer to Gateway and see data in the TTN Dashboard

Purpose

The purpose of this lab is to connect your ExpLoRer board to the gateway and see the temperature data in the TTN dashboard.

Requirements

Development Environment:	Arduino IDE 1.8.1 or above
Hardware Tools:	ExpLoRer Starter Kit
Lab files on class PC:	C:\MASTERS\21080\Lab3\...

Objective

During this lab, you will:

- Add your appEUI (from TTN console) to the existing Arduino Sketch
- Add code to existing Arduino Sketch in order to send temperature to TTN server
- View your connected device's data in TTN Data tab

On Your Own — Activate the device on the network

Step 1: Start Arduino IDE and open the sketch file

a. **Launch** Arduino IDE from the desktop if it's not already done
b. **Open** Lab3 by selecting from the menu: **File ▸ Open …** and selecting the Sketch file located at:

C:\MASTERS\21080\Lab3\Lab3.ino

Step 2: Add your unique appEUI

In the Lab 2, you collected the Application EUI generated from TTN console. It is now the time to **add** this unique key to the existing Arduino Sketch.

```
// Step 2 Replace here the 00's by your unique appEUI generated by TTN
const uint8_t appEUIfromTTN[8] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00} ;
```

At this point, you have now set the credentials information and be ready to join the network with the OTAA method.

Step 3: Activate your end device

To connect to the network, the end device must initiate the Over The Air Activation procedure by sending a JOIN_REQUEST message and the network server should reply with a JOIN_ACCEPT message if the end device is permitted to join the network. This can be done by using the following method provided by the library:

```
bool joinOTAALoRaNetwork(   uint8_t subBand,
                            const uint8_t devEUI[8],
                            const uint8_t appEUI[8],
                            const uint8_t appKey[16],
                            bool adr,
                            uint8_t dataRate ) ;
```

This method requires several parameters:
⇒ `uint8_t subBand` : channel that belong to the given frequency sub-band. For this lab, we will simply **enable channel #1** and disables the rest.
⇒ `const uint8_t devEUI, appEUI, appKey` : credentials information for OTA Activation
⇒ `bool adr` : Adaptive Data Rate status. Here we will **enable** this LoRaWAN features.
⇒ `uint8_t dataRate` : data rate value. For the lab, we will **set dataRate to 3**.
And returns TRUE if the end device has successfully join the network.

On Your Own — Activate the device on the network

Step 3 (continue): Activate your end device

The function `joinOTAALoRaNetwork` needs the cryptographic keys for OTAA passed as parameters and the function returns a boolean (true/false) which indicate the join process result.
In the Lab3.ino file, go to the "`Network Activation section`" and **replace** the "`// ### Your code here ###`" comments with the necessary code to join the network.

```
do
{
  setRgbColor(0x00, 0x00, 0xFF) ;
  debugSerial.println("Try to join the LoRa network through OTA Activation") ;
  // Step 3 Start the join procedure and put the result into joinRes
  // ### Your code here ###
  debugSerial.println(joinRes ? "Join Accepted." : "Join Failed! Trying again..
after 3 seconds.") ;
  if (!joinRes)
  {
    setRgbColor(0xFF, 0x00, 0x00) ;
    joinTentative ++ ;
    delay(3000) ;
  }
  if (joinTentative == 3)
  {
    debugSerial.println("Not able to join the network. Stay here forever!") ;
    while(1)
    {
      setRgbColor(0xFF, 0x00, 0x00) ;
      delay(250) ;
      setRgbColor(0x00, 0x99, 0xFF) ;
      delay(250) ;
      setRgbColor(0xFF, 0xFF, 0xFF) ;
      delay(250) ;
    }
  }
} while (joinRes == 0) ;
```

For guidance purpose and to fill the arguments of the function, the settings requested for this lab are:

- `subBand` = 1
- `adr` = true
- `dataRate` = 3

Notice in this portion of code, the activation procedure is tried 3 times with a help of the Boolean returned by the `joinOTAALoRaNetwork` function. This is a good practice to increase the chances of joining the network successfully. The actual line of code needed is shown below.

```
joinRes = LoRa.joinOTAALoRaNetwork(1, devEUI, appEUI, appKey, true, 3) ;
```

For your convenience, a full solution is provided in **C:\MASTERS\21080\Lab3_solution\Lab3_solution.ino** and in Appendix E, page E-4.

On Your Own — Activate the device on the network

- At this point, you have added code to existing Arduino Sketch in order to activate your end device within the TTN network server.

Step 4: Check the activation

Now it is time to verify the activation of your end device in the TTN network server.

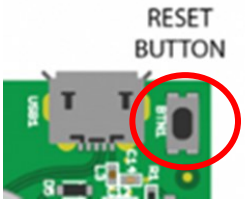
Step 4.1: Program ExpLoRer board with new sketch code

a. Make sure that your ExpLoRer board is **plugged** in to the PC.

b. **Verify/compile** the sketch by clicking the toolbar button

Verify/compile

c. **Press** the reset button (near the USB connector) twice within a second to place the ExpLoRer board into bootloader mode. Note that a new COM port dedicated for uploading the sketch is mounted.



d. **Select** the new mounted COM port from the menu:
Tools ▸ Port ▸ COMx (SODAQ ExpLoRer)

e. **Upload** the sketch to the board by clicking the toolbar button.

Upload

f. **Open** the serial monitor to **view data** sent by your working Arduino Software. You should observe a Join Accept from the serial monitor and the RGB LED should light **GREEN**.

Step 4.2: Confirm board is connecting to TTN

a. **Open** the TTN Console at https://console.thethingsnetwork.org/.
b. Once into your console, **click** on Applications and select the application you created in Lab 2.
c. **Click** on the device you have previously registered. You'll arrive to the *Device Overview* page.
d. **Observe** the status of your device. You should see an active status: **Status** ● 1 second ago

On Your Own — Send temperature to TTN Server

- At this point, your have now joined the LoRaWAN network. Sending data is the next step.

Step 5: Sending an uplink message

Once the device has been joined to the network, you can begin sending data to the network server. This can be made by using one of the following functions provided by the Arduino library.

Sending an unconfirmed message (without acknowledgement)

```
uint8_t send(uint8_t port,
             const uint8_t* payload,
             uint8_t size) ;
```

Sending a confirmed message (with acknowledgement)

```
uint8_t sendReqAck(
             uint8_t port,
             const uint8_t* payload,
             uint8_t size,
             uint8_t maxRetries) ;
```

To transmit a message, several parameters are requested:

⇒ `uint8_t port` : the port number allows multiplexing data streams on the same link. As an example, the end device may send measurements on one port number and some configuration data on another port number. The application server can then distinguish the two types of data based on the port number. The port number can be any number from 1 to 223.
⇒ `const uint8_t* payload` : the message to send
⇒ `uint8_t size` : size of the message to send
⇒ `uint8_t maxRetries` : applicable to confirmed message only to set the maximum retries value

The functions return the value 0 when transmission is successful or otherwise return one of the “Mac Transmit Error Codes” defined in the library.

In this lab, you will use the following settings:

- **Send a confirmed message by using the `sendReqAck()` method**
- **Using the port number 2, `port` = 2**
- **Set the maximum retries value to 3, `maxRetries` = 3**

On Your Own — Send temperature to TTN Server

Step 5: Sending an uplink message

We will construct the code **step by step** to send the temperature previously read in the Lab 1 to the TTN server.

Your code will be added in the loop() function.

Replace the "`// ### Your code here ###`" comments with the necessary code to complete the transmission of the temperature over the LoRaWAN network.

```
void loop()
{
  int sensorValue = analogRead(TEMP_SENSOR) ;
  float mVolts = (float)sensorValue * 3300 / 4096.0 ;
  float temp = (mVolts - 500) ;
  temp = temp / 10.0 ;
  debugSerial.println(temp) ;
  delay(3000) ;
  uint8_t res = NoResponse ;
  // Step 5.1 Create the payload variable
  // ### Your code here ###
  // Step 5.2 Fill the payload with temperature value
  // ### Your code here ###
  // Step 5.3 Send a confirmed uplink message
  // ### Your code here ###
  ...
  ..
  .

  // Step 5.4 Add a 20 sec delay
  // ### Your code here ###
}
```

Step 5.1: Create the payload variable

First, you need to declare an array with ten elements of type `char`. This array will contain the payload to transmit.

```
char payload[10] ;
```

On Your Own — Send temperature to TTN Server

Step 5: Sending an uplink message

Step 5.2: Fill the payload with temperature value

Then you have to use the `sprintf()` function to write formatted data to a string pointed to by payload. Here you will compose a string from a float (the temperature value stored in the `temp` variable) by using the format: `"%.2f"` which means a precision of 2 digits after the decimal point of the floating point value.

```
sprintf(payload, "%.2f", temp) ;
```

Step 5.3: Send a confirmed uplink message

The payload is formatted with the temperature value and ready to be sent. To transmit the payload, you should use the method `sendReqAck()` from the class named `LoRa` and provide the parameters previously described. This function returns an `uint8_t` datatype which should be stored into the `res` variable for a status of the transmission. Notice the `strlen()` function to compute the length of the payload but not including the terminating null character.

```
res = LoRa.sendReqAck(2,(const uint8_t*)payload, strlen(payload), 3);
```

Step 5.4: Add a 20 sec delay before restarting the loop()

You will **pause** the program for 20 seconds. The method `delay(ms)` should be use to pause the program for a certain amount of time in milliseconds.

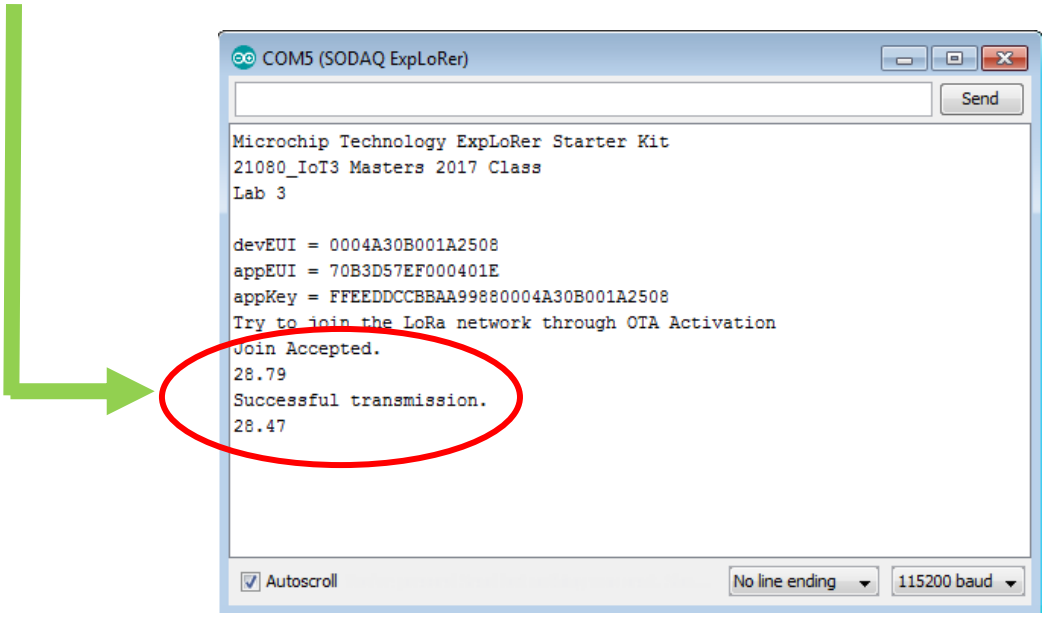
```
delay(20000) ;
```

Just like in step 4.1, **compile** the Sketch and **program** the ExpLoRer board. **Open** the serial monitor to **view data** sent by your working Arduino Software.

On Your Own — Send temperature to TTN Server

Results

You should observe a successful transmission message.



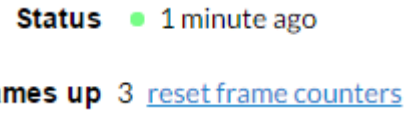
Step 6: View your connected device's data in TTN Data tab

a. **Open** the TTN Console at https://console.thethingsnetwork.org/.

b. Once into your console, **click** on Applications and select the application you created in Lab 2.

c. **Click** on the device you have previously registered. You'll arrive to the *Device Overview* page.

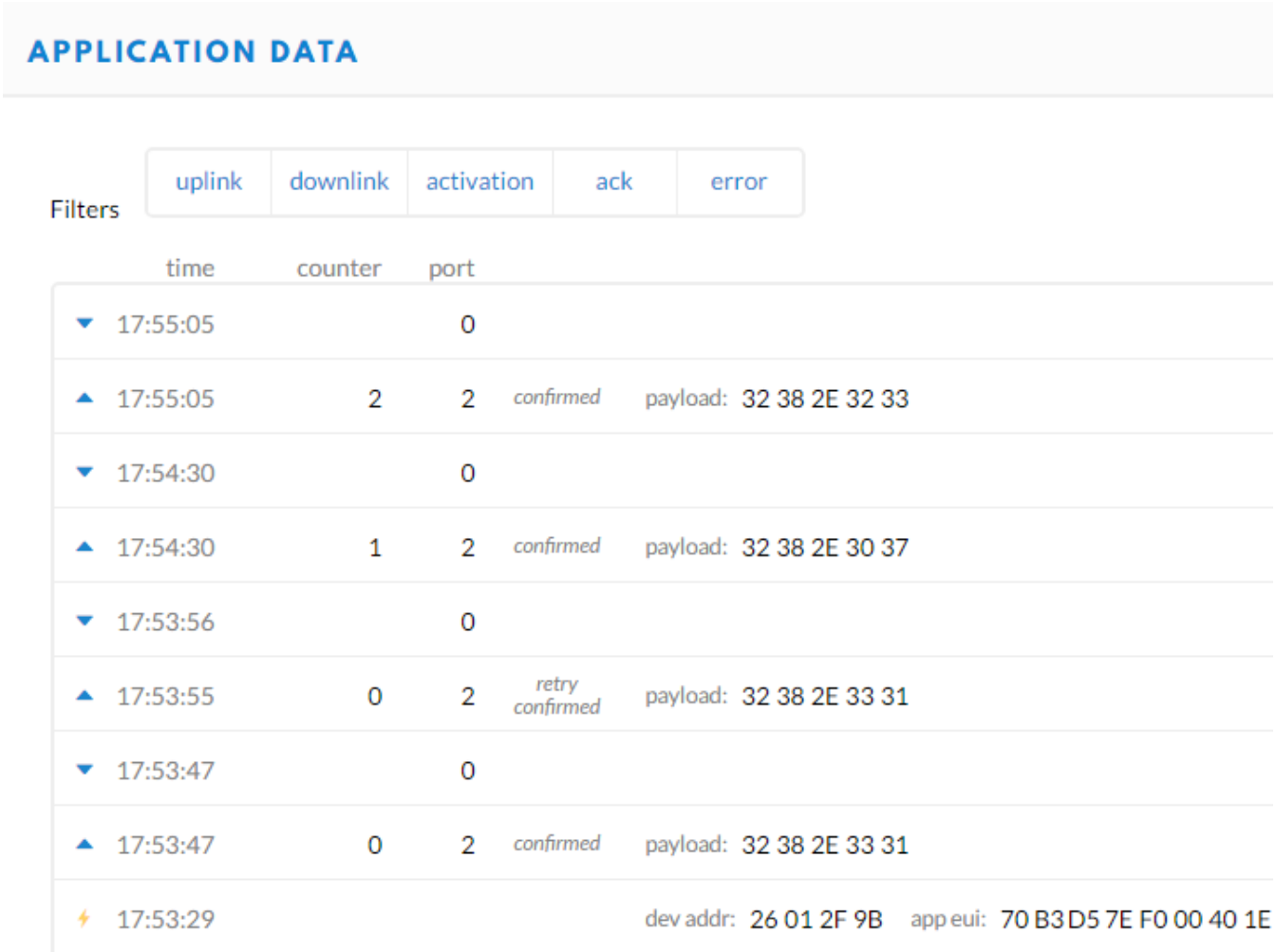
d. **Observe** the status of your device. You should see an active status: and the frame up counter value should increase at each successful transmission.



e. **Click** on the Data tab to see the Application Data.

Results

You should visualize the activity of your end device. Notice the ASCII format of the temperature. Click on the “Data” tab to see Application Data.

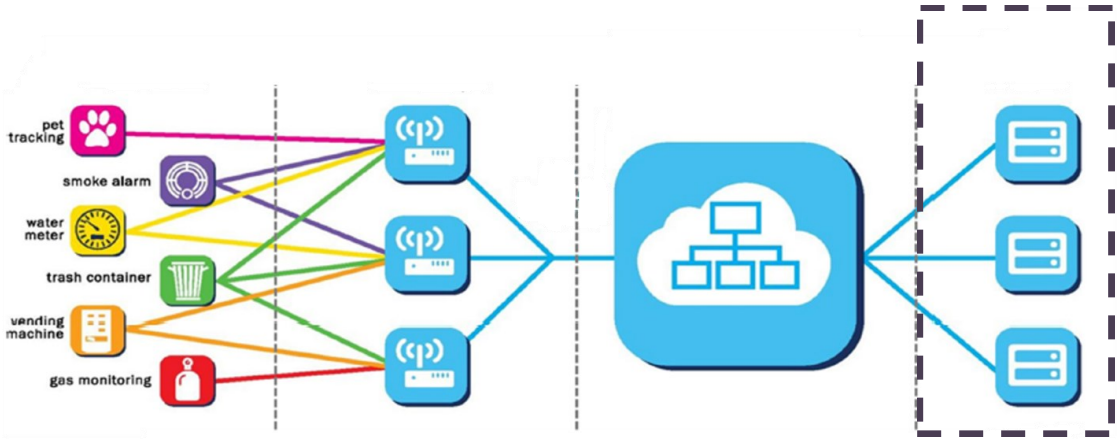


Conclusions

A network server will collect data from devices connected to gateway. The TTN console allows you to see activity from your devices and can also send downlink data to the device.

What you’ve done so far:

- Send uplink message to the TTN server
- Become familiar with the data visualization offered by the TTN console



Lab 4

Building a simple application with Node-RED

Purpose

The purpose of this lab is to guide you through the creation of a simple Node-RED application that graphs temperature data collected from the ExpLoRer board through the TTN network server.

Requirements

Development Environment:	Node.js v6.x+, NodeRED v0.16.2+
Browser:	Mainstream browser like Google Chrome or Mozilla Firefox
Hardware Tools:	ExpLoRer Starter Kit
Lab files on class PC:	C:\MASTERS\21080\Lab4\...

Your lab PC has node.js and Node-RED already installed. If you are following these steps on your own PC, you'll need to follow the steps in Appendix D to install the correct software.

Objective

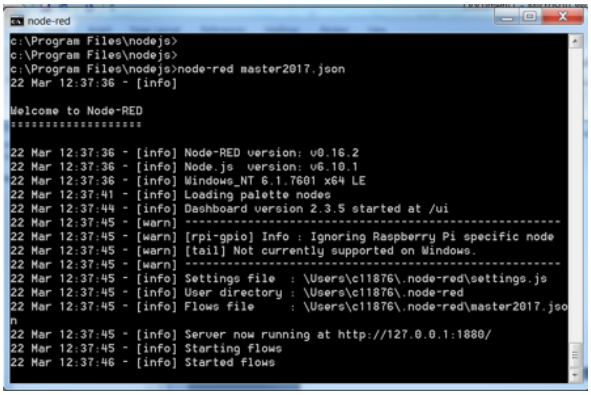
During this lab you will create a simple Node-RED application that

- Runs on your local machine in a browser
- Collects temperature data from the TTN server
- Graphs temperature data in Node-RED

On Your Own—Creating a Node-RED Application

Step 1: Start the Node-RED software from a CMD window.

- In Windows, **Open** a Windows Command Prompt
  Start Menu -> All Programs -> Accessories -> Command Prompt.
- Type the following command in the Command Prompt window to start Node-RED: **node-red**

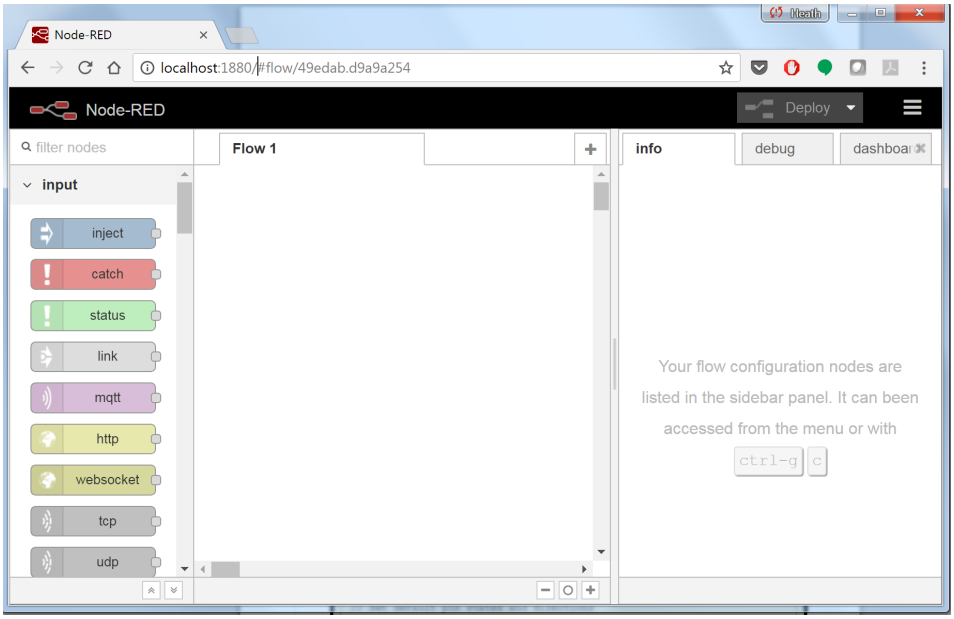


Node-RED should now be running on your PC. Make sure the Command Prompt window indicates that the "Server now running at http://127.0.0.1:1880/."

Step 2: Open the Node-RED GUI in the Chrome browser.

- **Open** the Google Chrome browser and type the following into the address bar: **http://localhost:1880**

- Your browser will show the Node-RED flow editor.



Step 3: Drag-and-drop the needed nodes into your Node-RED application.

Step 3.1: Add the "TTN Device" Node

Drag-and-drop the "ttn message" node into your Flow 1 workspace. Once configured, this node will collect data from the specified application running on the TTN server. The data is in json format.

Step 3.2: Add the "Function" Node

Drag-and-drop the "function" node into your Flow 1 workspace. Once configured and connected, this node can run JavaScript to do many different things. In this lab, you will be converting the temperature data from Celsius to Fahrenheit.

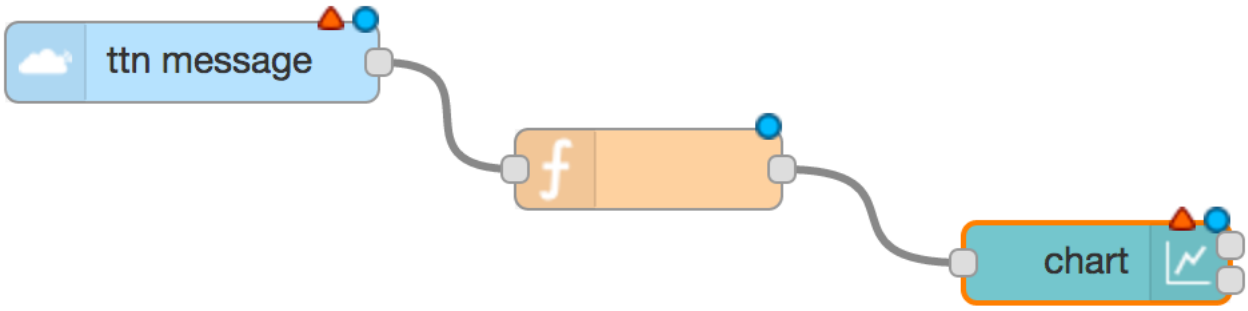
Step 3.3: Add the "Chart" Node

Drag-and-drop the "chart" node into your Flow 1 workspace. Once configured and connected, this node will graph data that is passed to it on a separate UI screen.

So far, your Flow 1 workspace should look like the picture below. Now it is time to connect the nodes together and configure them!

Step 4: Connect the nodes together.

Wire the nodes together in a logical flow by clicking on the wiring node to start the wiring, and dragging to the next node to complete the wiring. Complete the wiring by connecting the three nodes as shown below.



Step 5: Configure the “ttn message” node.

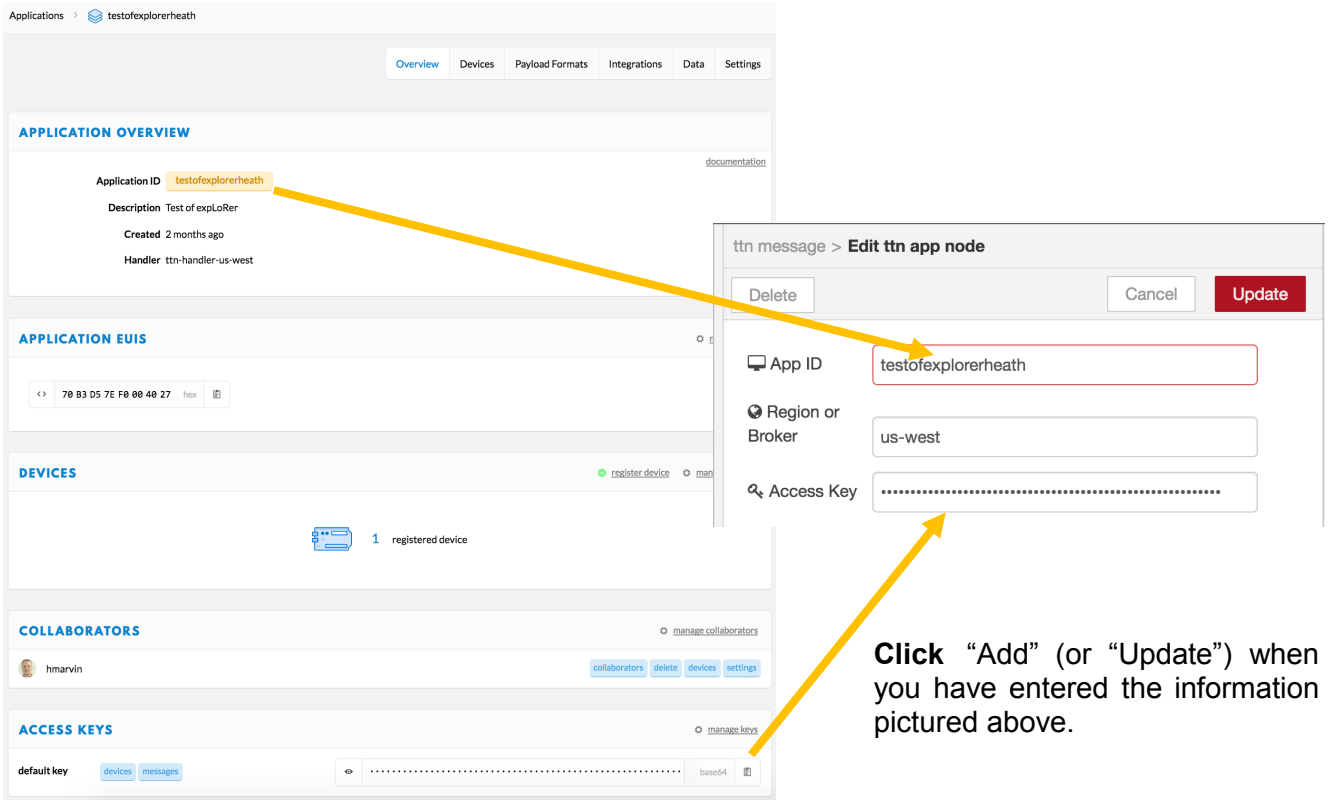
Step 5.1: Name the node

Double click on the “ttn message” node to open the node edit dialog box. In the Name box, enter the name of the node. For this exercise, you can name it “Temperature Input.”

Step 5.2: Add App information

In the App section, **click** the edit button to add a new ttn app. You will need to add the App ID, Region or Broker, and Access Key.

You will collect this information from the TTN console, so **open** the TTN Console at https://console.thethingsnetwork.org/.
Once into your console, **click on** Applications and select the application you created in Lab 2.
Copy the App ID and Access Key from the TTN Console into your NodeRED application.
Enter “us-west” for the Region or Broker.

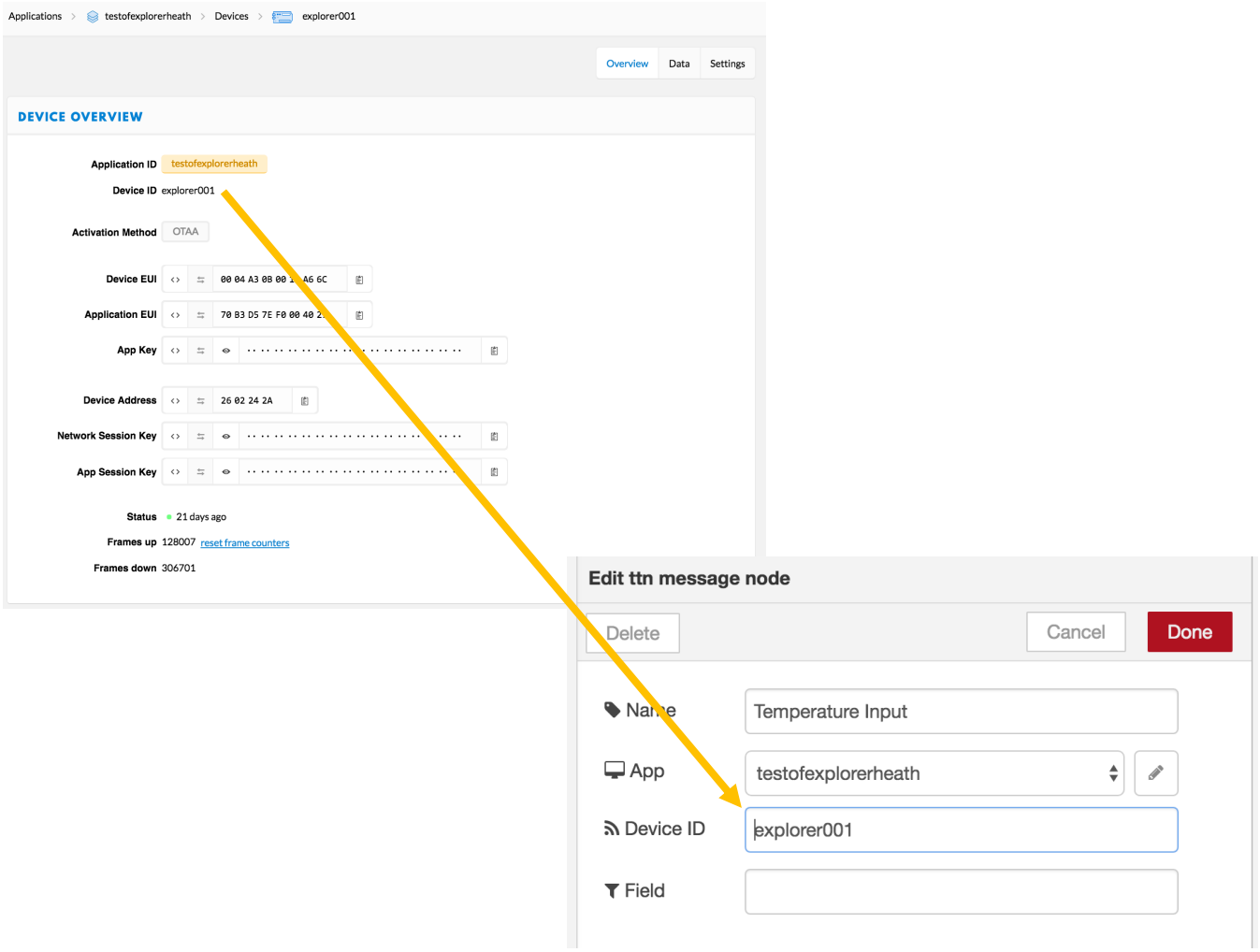


Click “Add” (or “Update”) when you have entered the information pictured above.

Step 5 (Continued):

Step 5.3: Name the node

Back in the “Edit ttn message node” dialog, add your device ID. This can be found in the Device Overview section of the TTN Console.
From the Application Overview section, click on the “Devices” tab.



Click “Done” when you have entered the information pictured above.

Step 6: Configure the “function” node.

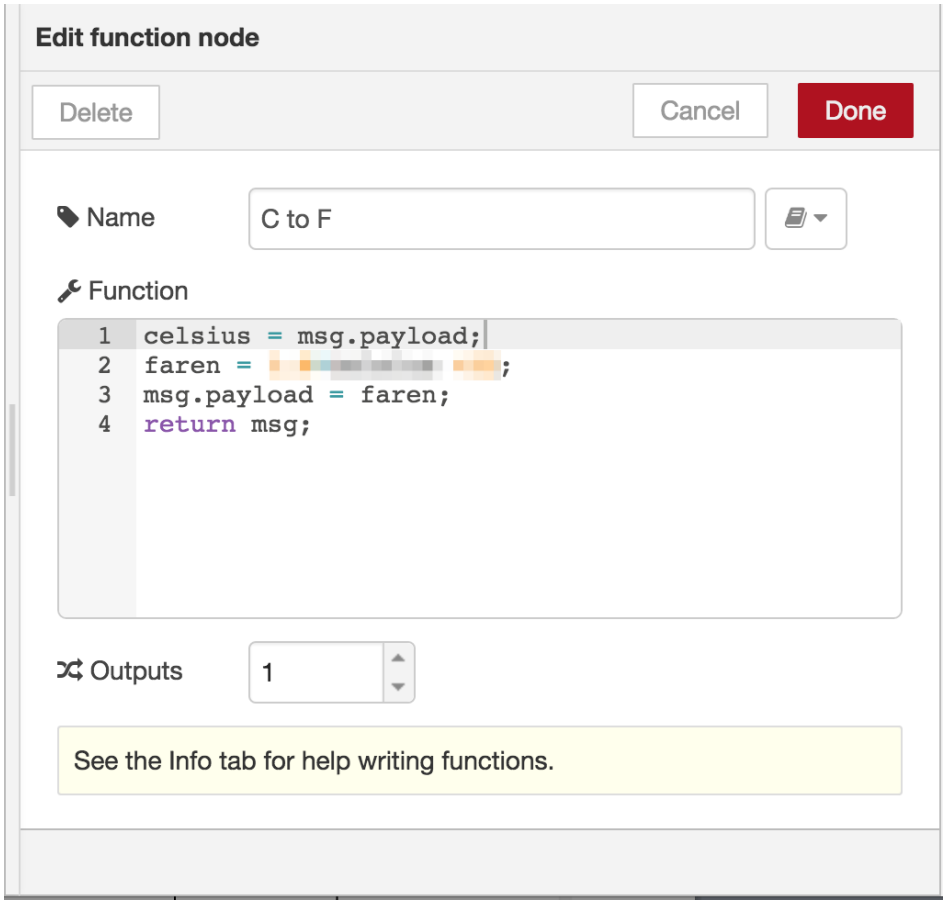
Step 6.1: Name the node

Double click on the “function” node to open the node edit dialog box. In the Name box, enter the name of the node. For this exercise, you can name it “C to F” since you will be converting temperature from Celsius to Fahrenheit.

Step 6.2: Add Function Code

In the function section, type the following code. We will let you figure out the Celsius to Fahrenheit calculation!

```
celsius = msg.payload;
faren = ******;
msg.payload = faren;
```



Click “Done” when you have entered the information pictured above, and added the conversion calculation.

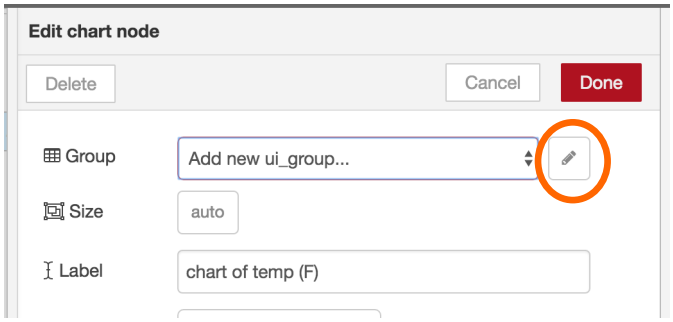
Step 7: Configure the “chart” node.

Step 7.1: Set the chart `label`

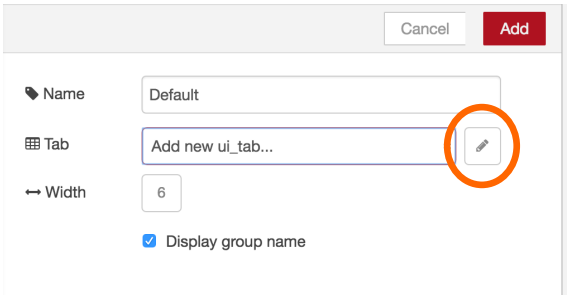
Double click on the “chart” node to open the node edit dialog box. In the Label box, enter the name of the chart. For this exercise, you can name it “chart of temp (F)” since you will be displaying temperature in Fahrenheit.

You also need to create a “Group” and “Tab” for your chart.

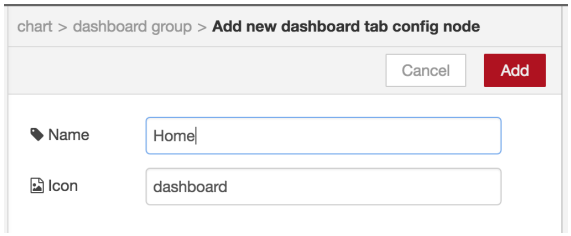
A. Click the pencil to “Add new ui_group”.



B. Leave the name at its default and click the pencil to “Add new ui_tab.”

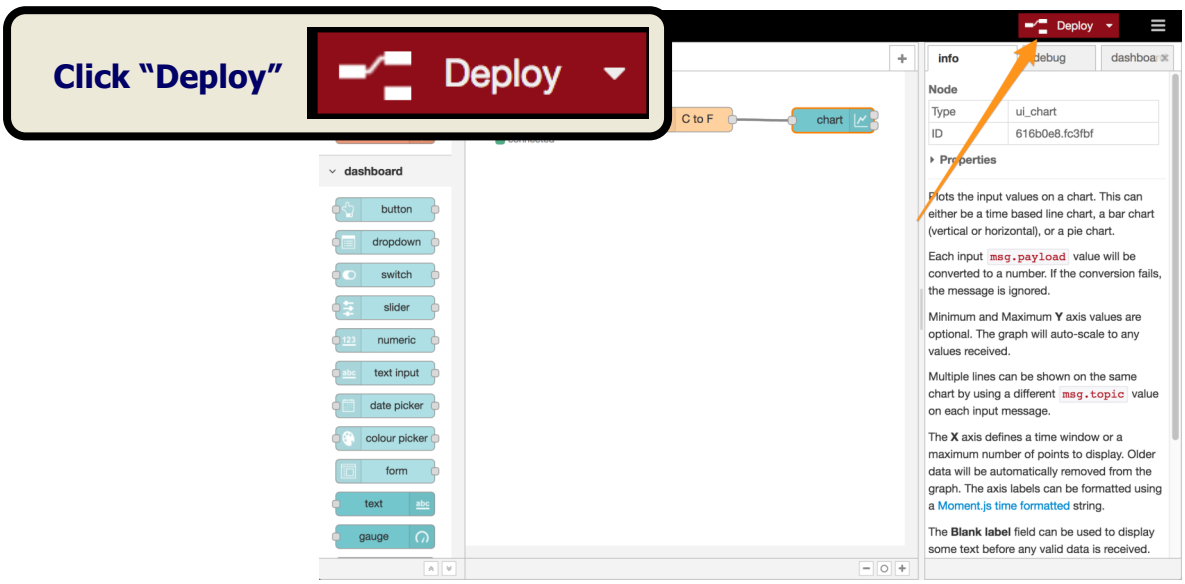


C. Leave the name at its default and click “Add.”
D. Click “Add” and “Add” to finish editing the Chart node.



Click “Done” when you have entered the information pictured above.

Step 8: Deploy your NodeRED application.



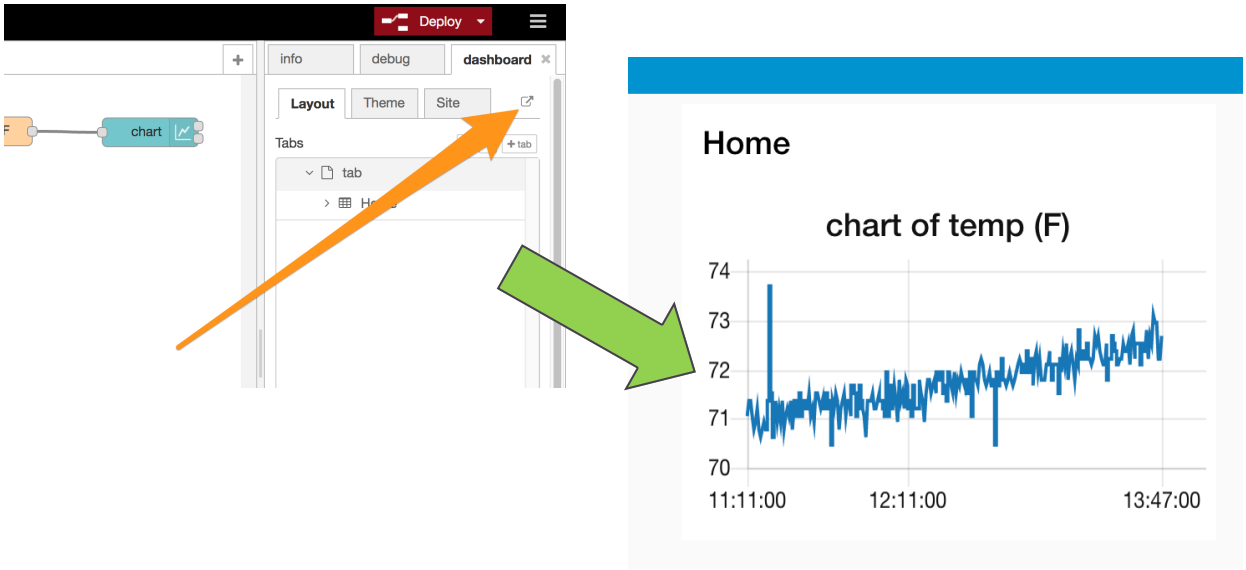
Step 9: View your data in the UI dashboard.

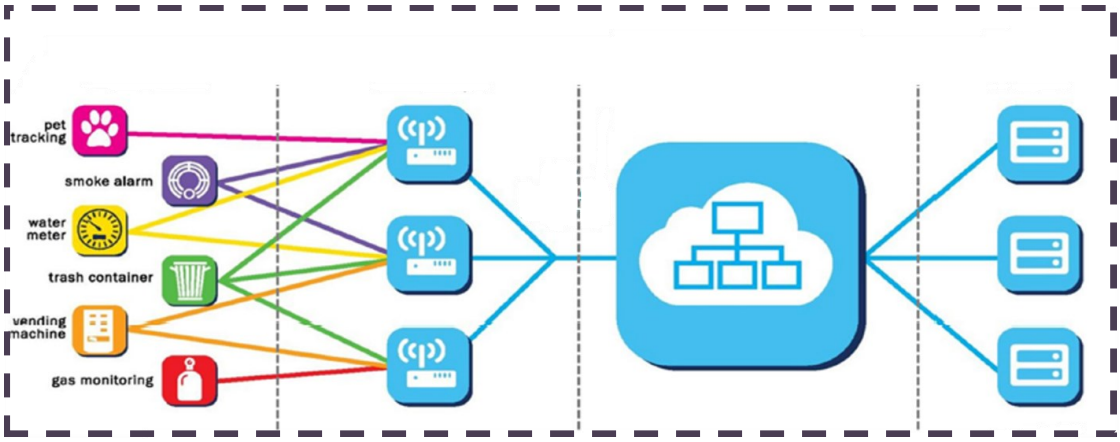
Step 9.1: View data in the graphing dashboard.

There are two ways to open the dashboard.

1. **Navigate** to the URL (http://localhost:1880/ui).
2. **Launch** the dashboard from the dashboard tap in the NodeRED editor (see picture to the right).

This will launch your dashboard that is graphing your current temperature data.





Lab 5

Bonus Lab: Two-Way Communication

Purpose

- Bonus: Change color of RGB LED on SODAQ through a Node-RED application.

Requirements

Development Environment:	Arduino IDE 1.8.1 or above, Node.js v6.x+, NodeRED v0.16.2+
Browser:	Mainstream browser like Google Chrome
Hardware Tools:	ExpLoRer Starter Kit
Lab files on class PC:	C:\MASTERS\21080\Lab5\...

Your lab PC has the Arduino libraries required for this lab. If you are following these steps on your own PC, you'll need to refer to Appendix B for the process for installing the following libraries.

1. RN4871 Bluetooth: Found here on the SODAQ ExpLoRer website (http://support.sodaq.com/sodaq-one/explorer/)
2. RTC: Found here on github (https://github.com/arduino-libraries/RTCZero)

Objective

Bonus: In this bonus lab, you will use an existing Node-RED application to change the color of the RGB LED on the SODAQ ExpLoRer board.
This lab can also be used as a reference for the following additional features:

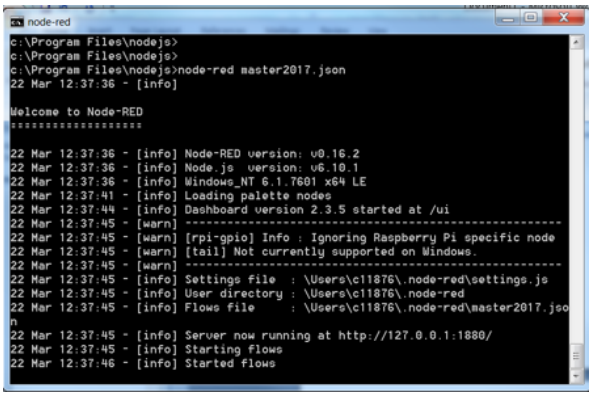
- Using the RTC on the ExpLoRer board.
- Using the Bluetooth module on the ExpLoRer board (this lab just puts the module to sleep).
- Using the MQTT node in Node-RED.

On Your Own—Opening an Existing Node-RED Application

Step 1: Start the Lab 5 Node-RED flow from the CMD Window

- In Windows, **Open** a Windows Command Prompt
  Start Menu -> All Programs -> Accessories -> Command Prompt.
- Navigate to the Lab 5 directory: C:\MASTERS\21080\Lab5\

```
> cd C:\MASTERS\21080\Lab5
```



- Type the following command in the Command Prompt window to start Node-RED and run the Lab 5 flow: **node-red Lab5-flow**

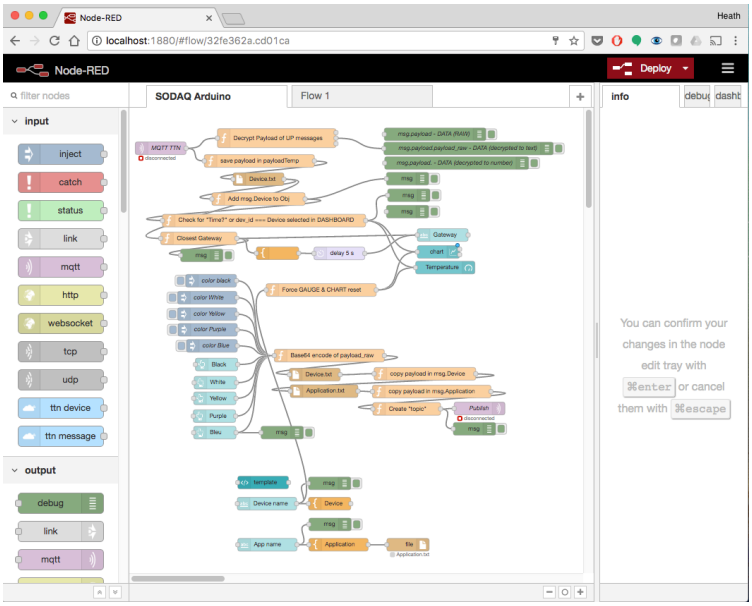
```
> node-red Lab5-flow
```

The Lab 5 flow and Node-RED should now be running on your PC. Make sure the Command Prompt window indicates that the "Server now running at http://127.0.0.1:1880/."

Step 2: Open the Node-RED GUI in the Chrome browser.

- **Open** the Google Chrome browser and type the following into the address bar: **http://localhost:1880**

```
http://localhost:1880
```

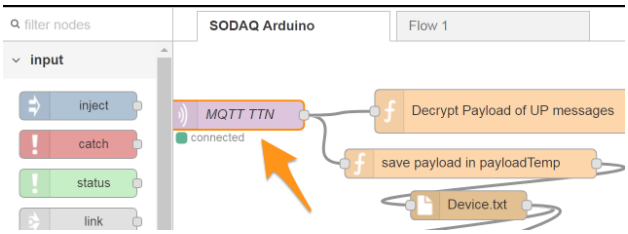


- This view shows the completed flow for Lab 5. This flow uses an MQTT node to collect and graph temperature data (like in Lab 4). In addition this flow allows the user to change the color of the RGB LED on the SODAQ ExploRer board.

Step 2 (Continued): Configure the “MQTT TTN” node.

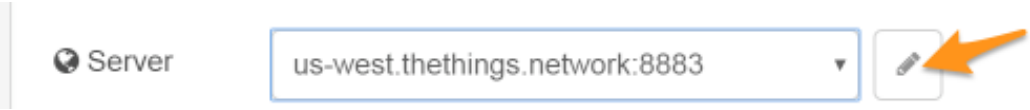
Step 2.1: Verify server

Double click on the “MQTT TTN” node in the top left of the Node-RED flow. Once the dialog box on the right of the screen opens, verify that the server is set to “us-west.thethings.network:8883.

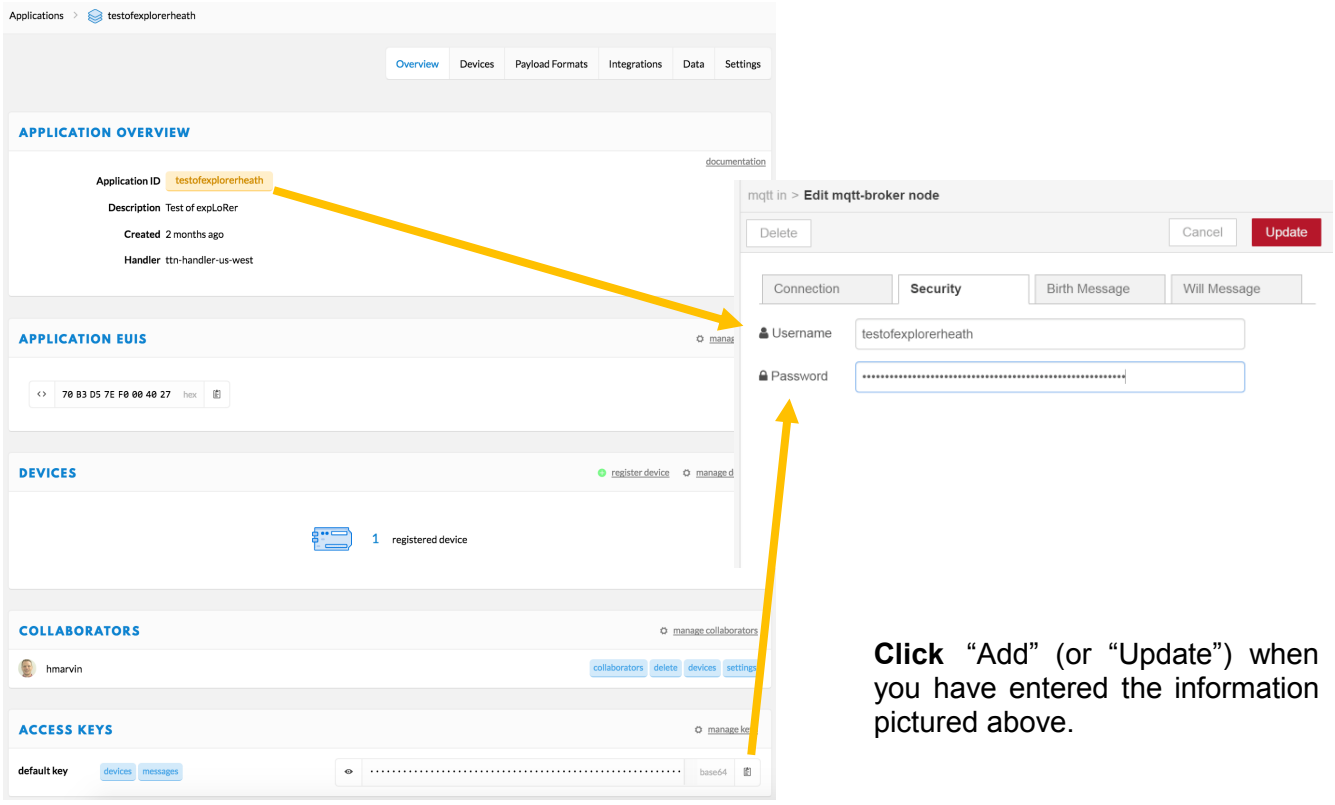


Step 2.2: Add Security/App information

After verifying the server, **click** the edit button and navigate to the “Security” tab to add the TTN connection information. You will need to add the App ID and App Key from TTN.



You will collect this information from the TTN console, so **open** the TTN Console at https://console.thethingsnetwork.org/.
Once into your console, **click on** Applications and select the application you created in Lab 2.
Copy the App ID and Access Key from the TTN Console into your NodeRED application.



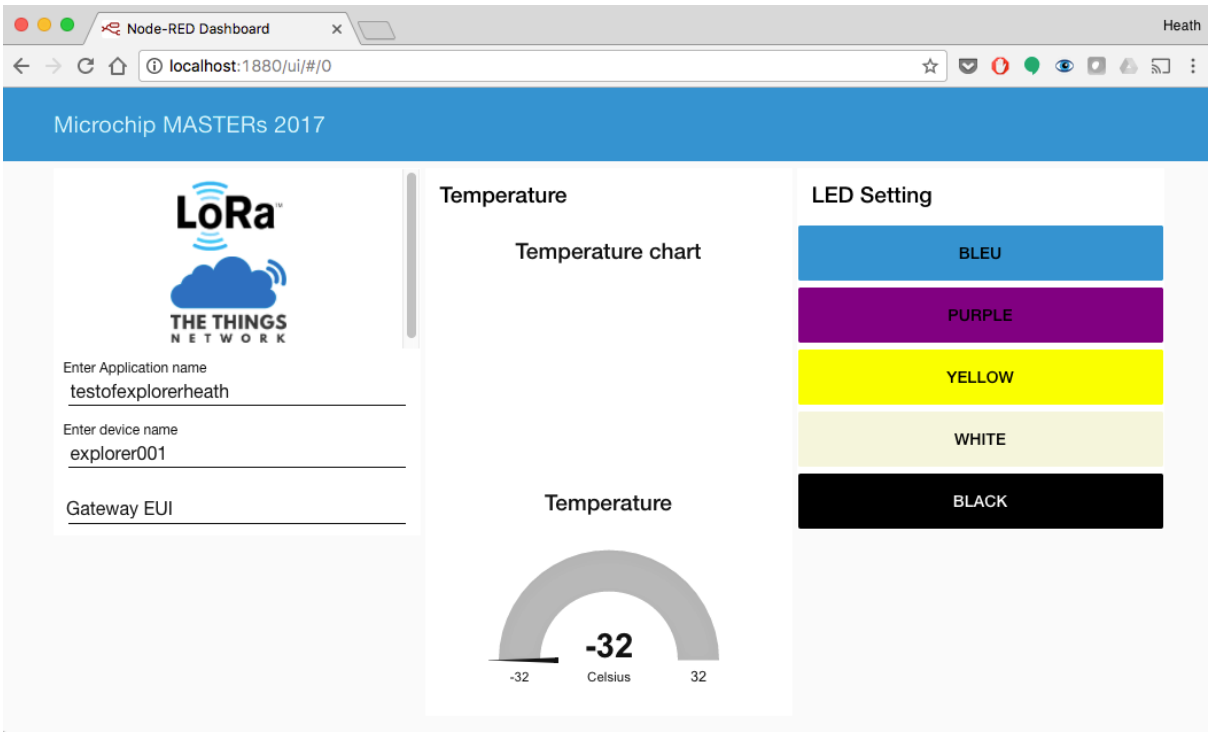
Click “Add” (or “Update”) when you have entered the information pictured above.

Step 3: Open the Node-RED UI in the Chrome browser.

- **Open** the Google Chrome browser and type the following into the address bar: **http://localhost:1880/ui**

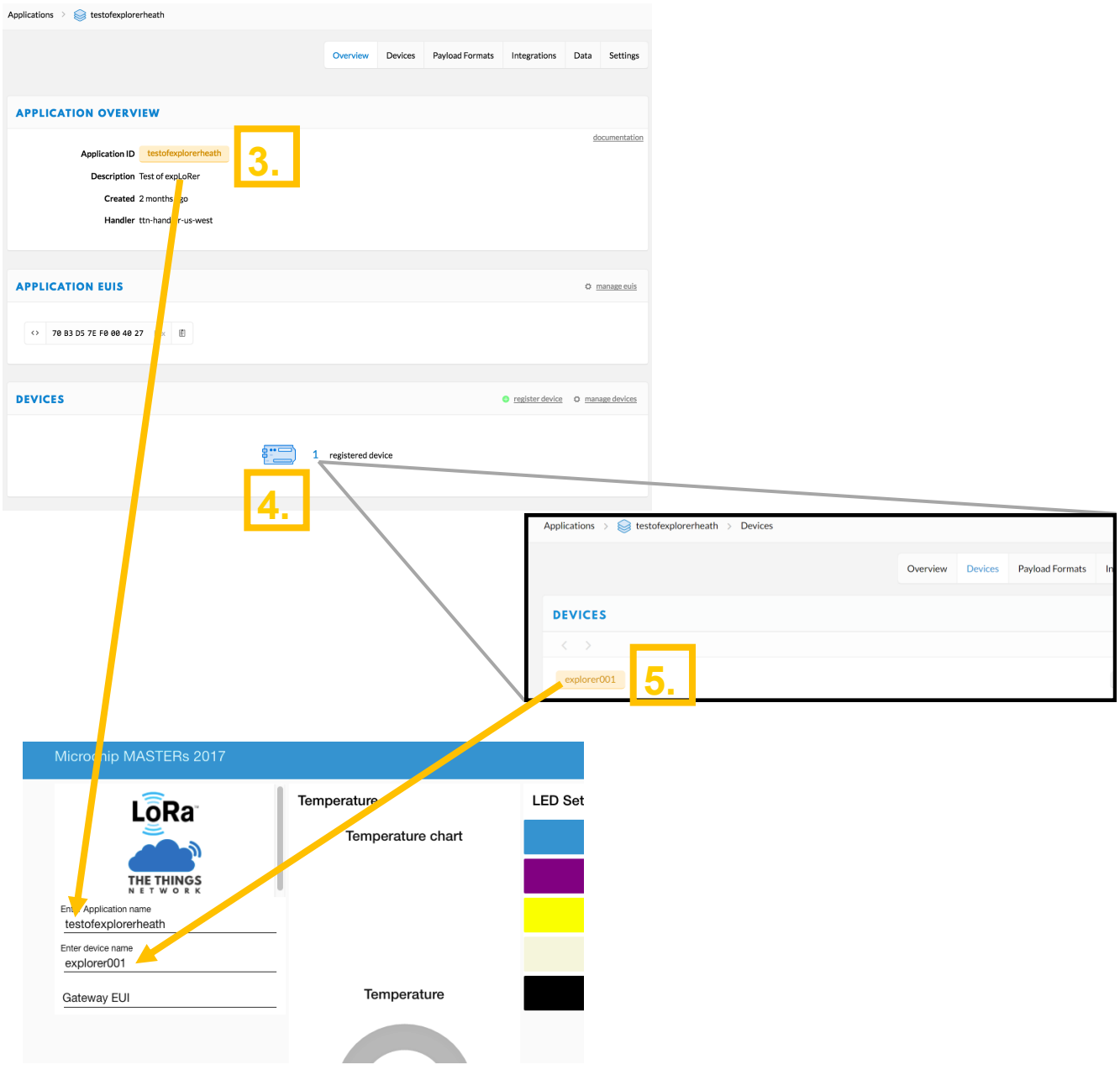
http://localhost:1880/ui

- In this lab, you will be using the Node-RED dashboard UI to configure the application name and device name. This will allow you to connect to your own SODAQ ExpLoRer board and read the temperature and set the color of the LED.



Step 4: Enter the correct Application and Device names in the Node-RED UI

1. You will collect this information from the TTN console, so open the TTN Console at https://console.thethingsnetwork.org/.
2. Once into your console, click on Applications and select the application you created in Lab 2.
3. Copy the Application ID into the NodeRED UI.
4. In the TTN Console, click on Devices.
5. Copy the Device name into the NodeRED UI.



Step 5: Program ExpLoRer board with new sketch code.

a. **Launch** Arduino IDE and make sure that your ExpLoRer board is plugged in to the PC.
b. **Open** the Lab5 sketch by selecting from the menu: **File ▸ Open …** and selecting the Sketch file located at:

C:\MASTERS\21080\Lab5\Lab5.ino

As already done in the Lab 3, **add** your unique Application EUI key generated from your TTN account to the current Arduino Sketch.

```
// Step 5 Replace here the 00's by your unique appEUI generated by TTN
const uint8_t appEUIfromTTN[8] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00} ;
```

c. **Verify/compile** the sketch by clicking the toolbar button.

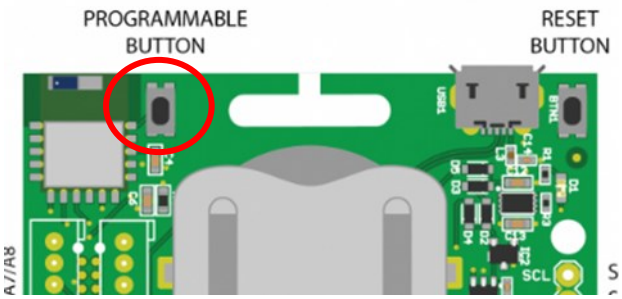
Verify/compile

d. **Upload** the sketch to the board by clicking the toolbar button.

Upload

Step 6: Confirm board is connecting to TTN.

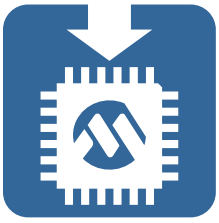
In this firmware, the SODAQ ExpLoRer is in low power mode. It will sleep and only wake every 60 seconds. This allows the board to last about 12 hours on the rechargeable coin cell. To wake the board and immediately connect to TTN, press the button next to the BLE module.



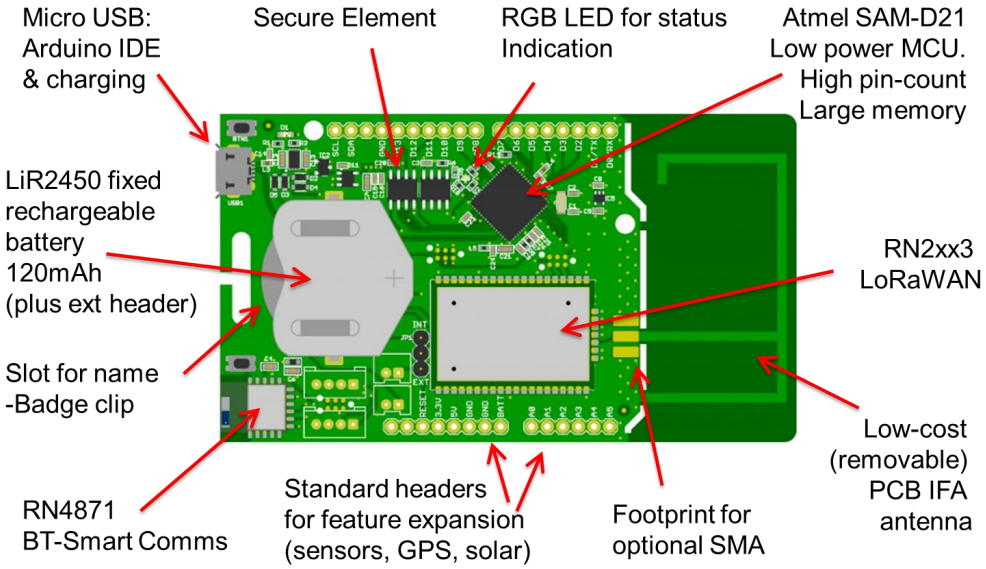
Once the board wakes up and sends temperature data, you should see the data graphing in the Node-RED UI. You should also be able to select a color for the RGB LED from within the UI.

Appendix A

Hardware Information



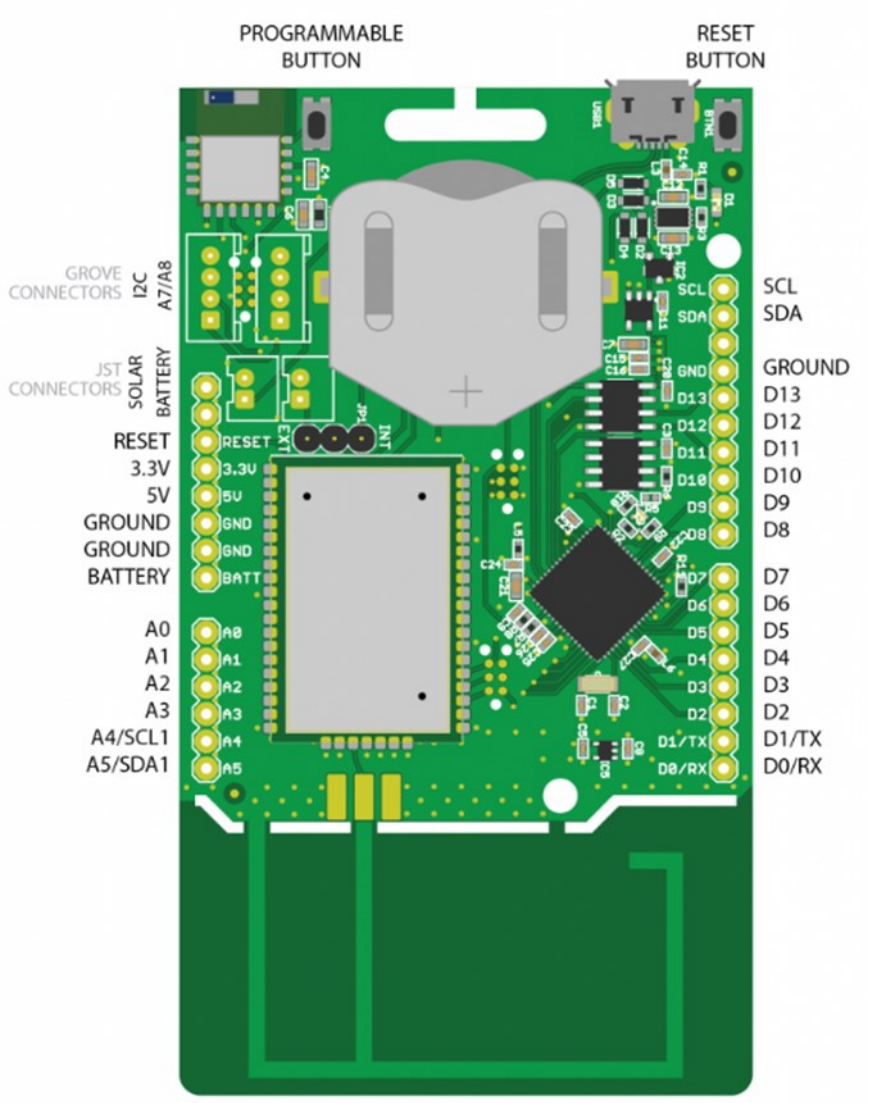
ExpLoRer Starter Kit



Specifications	
Microcontroller	Microchip ATSAMD21J18 32-bit ARM Cortex M0+
Compatibility	Arduino M0 Compatible
Size	94 x 53 mm
Operating Voltage	3.3V
I/O Pins	20
Flash Memory	256 kB (internal) and 4MB (external SST25PF040C)
SRAM	32 kB
EEPROM	Up to 16 kB by emulation
Clock Speed	48 MHz
Power	5V USB power and/or 3.7V Lithium battery
Charging	Solar charge controller, up to 500mA charge current
LED	RGB LED and Blue LED
LoRa	Microchip RNxx3A LoRaWAN 1.01 Certified Module
Bluetooth	Microchip RN4871 BLE 4.2 Certified Module
CryptoAuthentication	Microchip ATECC508A Secure Element
Temperature Sensor	Microchip MCP9700AT
USB	Micro USB Port

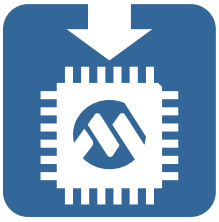
ExpLoRer Starter Kit

Pinout Definition	
RGB LED	LED_RED, LED_GREEN, LED_BLUE
Bluetooth Wake up	BLUETOOTH_WAKE
LoRa Reset	LORA_RESET
Bluetooth Reset	BT_RESET
Programmable Button	BUTTON
Temperature Sensor	TEMP_SENSOR
Grove Header I2C	PIN_WIRE_SDA, PIN_WIRE_SCL
Grove Header	A7, A8



Appendix B

Get Started with Arduino IDE and ExpLoRer



Purpose

To get familiarized with the Arduino Software (IDE).
To be ready to use the ExpLoRer Starter Kit by installing Board support and the library.

Objective

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the ExpLoRer Starter Kit. It runs Windows, Mac OS X and Linux. The environment is written in java and based on Processing and other open-source software.
Here, we will install and configure the environment to be ready for the Microchip ExpLoRer Starter Kit.

On Your Own — Arduino IDE Setup

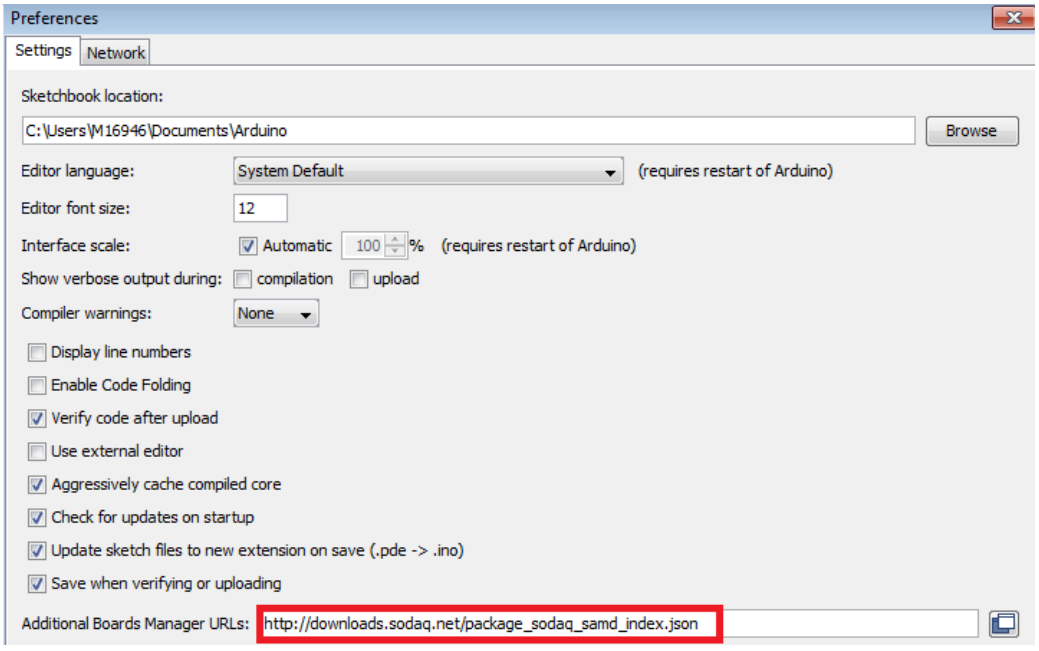
Step 1: Download and install the latest Arduino Software IDE

a.

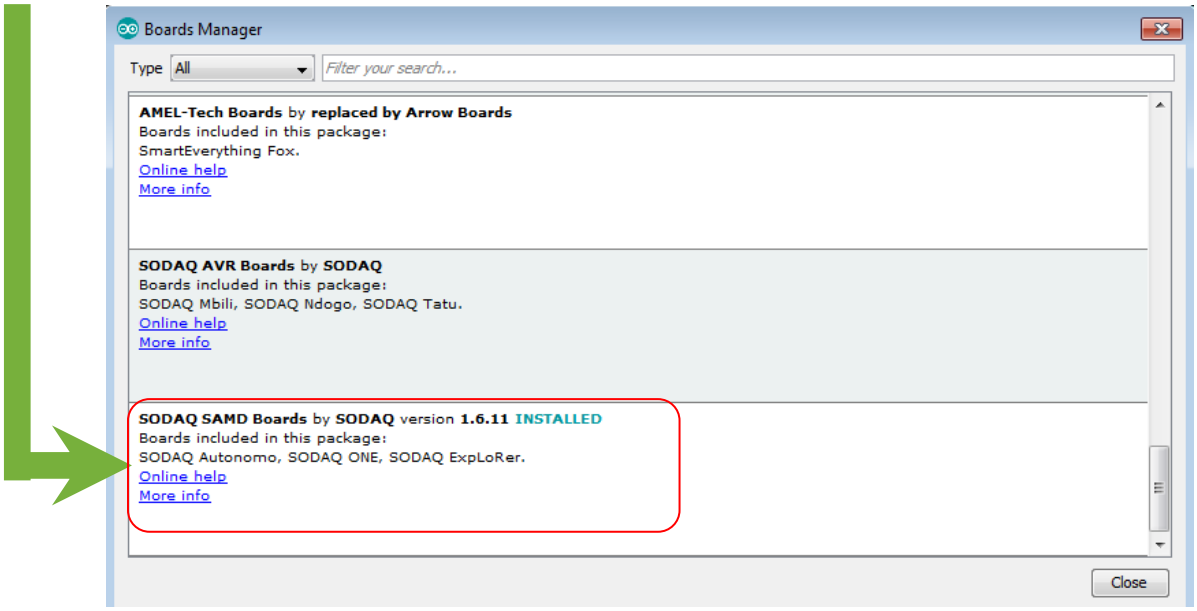
Download from the following link https://www.arduino.cc/en/Main/Software
b. **Start** the installation procedure
c. **Launch** the Arduino Software

Step 2: Configure the Board Manager

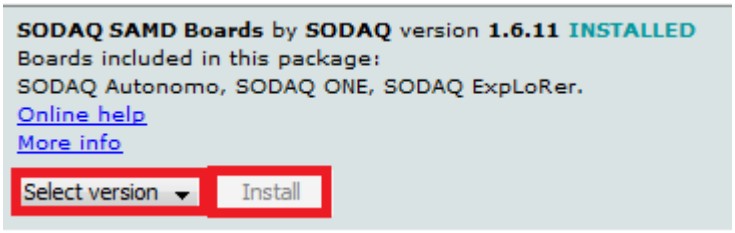
a. In order to install the board to the environment, you will need to **add** the Sodaq Board Manager URL: **http://downloads.sodaq.net/package_sodaq_samd_index.json**
to the menu *File -> Preferences -> Additional Board Manager URLS*



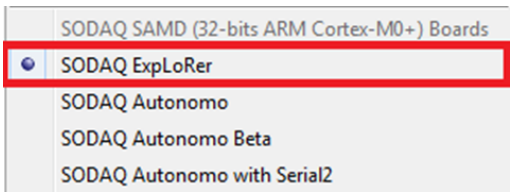
b. **Open** the menu *Tools -> Board:"<Selected Board>" -> Boards Manager,* the SODAQ SAMD Boards package should appear in the list



c. **Install** the latest version of the SODAQ SAMD Boards package and close the Boards Manager.



d. Now the board should be visible from the list menu. **Select** the SODAQ ExpLoRer board from the menu: *Tools -> Board*

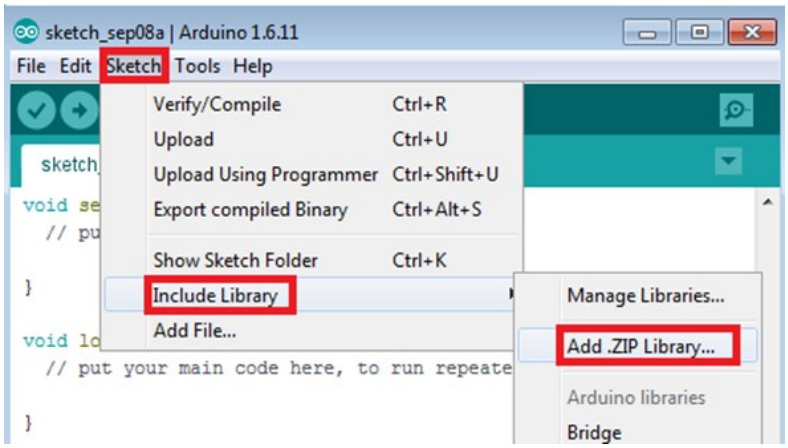


Step 3: Adding library to the Arduino environment

In order to use the features of the ExpLoRer Starter Kit, for instance pinout definition and LoRa features through API methods, we shall add the library to the Arduino environment.

a. **Include** the two required library files listed below by using the menu *Sketch -> Include Library -> Add .ZIP Library*:

*C:\MASTERS\21080\Lib***IoT3_LoRa_RN2xx3-Masters2017.zip**
*C:\MASTERS\21080\Lib***IoT3_Microchip_RN487x-Masters2017.zip**

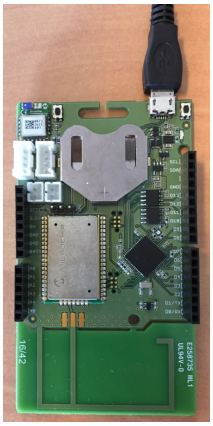


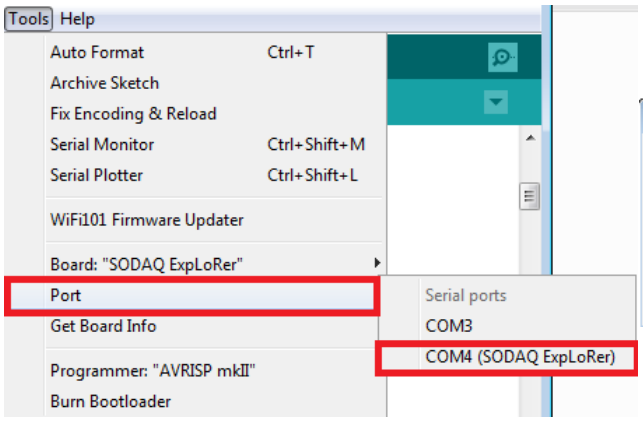
Step 4: Connect the ExpLoRer Starter Kit

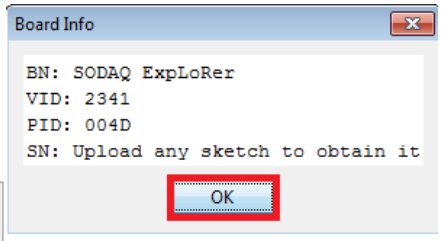
a. **Connect** the Starter Kit to your computer using the micro USB cable and wait for the driver installation and the COM port mounting. The USB port powers the board and enables the user to communicate with the kit.

b. **Set** the corresponding COM port within the Arduino IDE: *Tools -> Port COM*

c. **Check** the communication by selecting *Tools -> Get Board Info*







a b c

Helpful

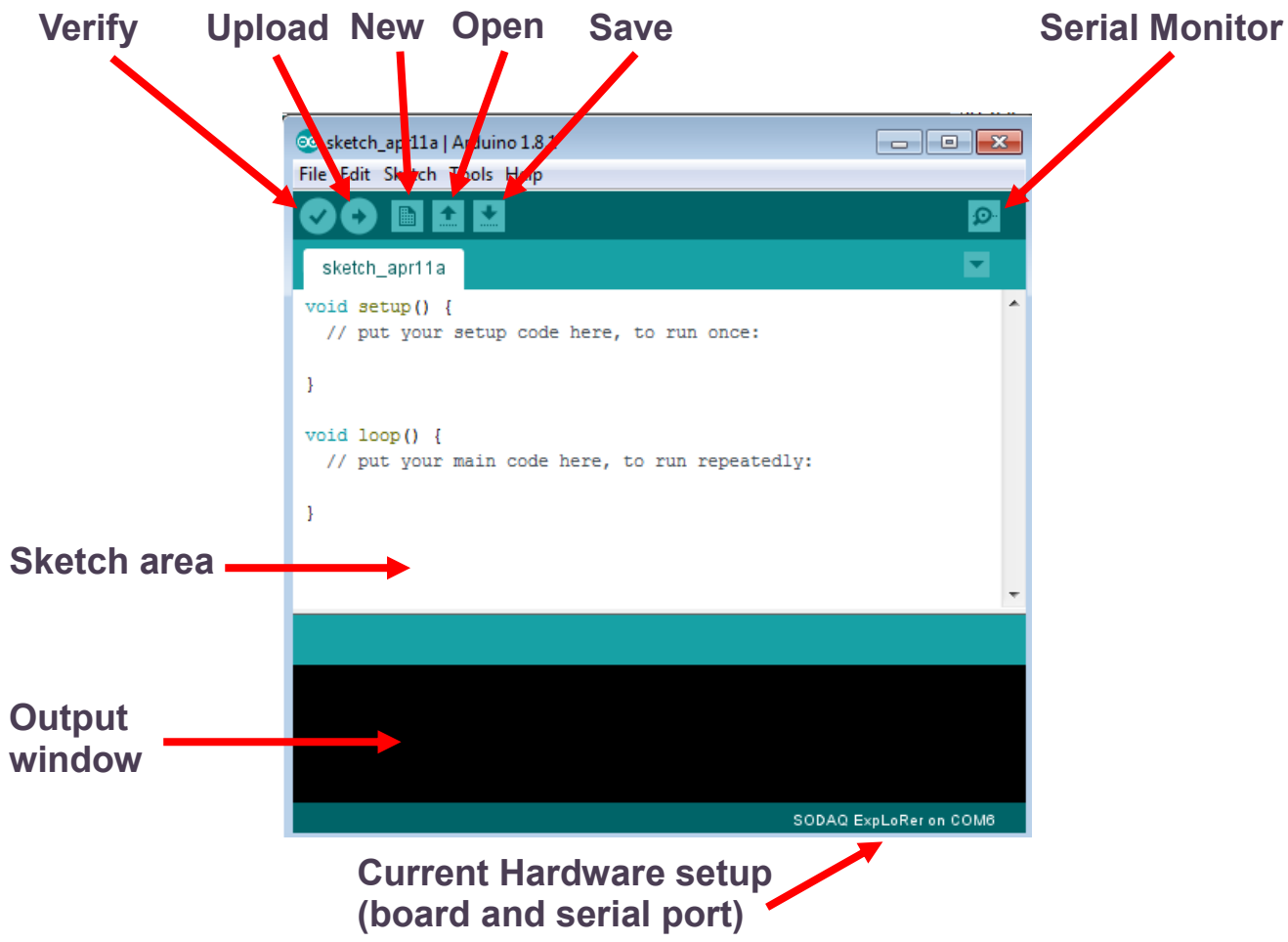
A sketch is the name that Arduino uses for a program. It is the unit of code that is uploaded to and run on an Arduino board. The programming language used in the Arduino IDE system is C++. Familiarity with C or C++ will, therefore, be very helpful in learning to write sketches using Arduino IDE.

There are two basic functions required by all sketches: setup() and loop().

setup() is called once at the beginning of execution of the sketch. This function performs any initialization that is needed to set up the execution environment for the sketch. This could include setting pin direction on I/O pins, initializing libraries and so on.

loop() is called repeatedly during the execution of the sketch. This is where the body of the sketch is located.

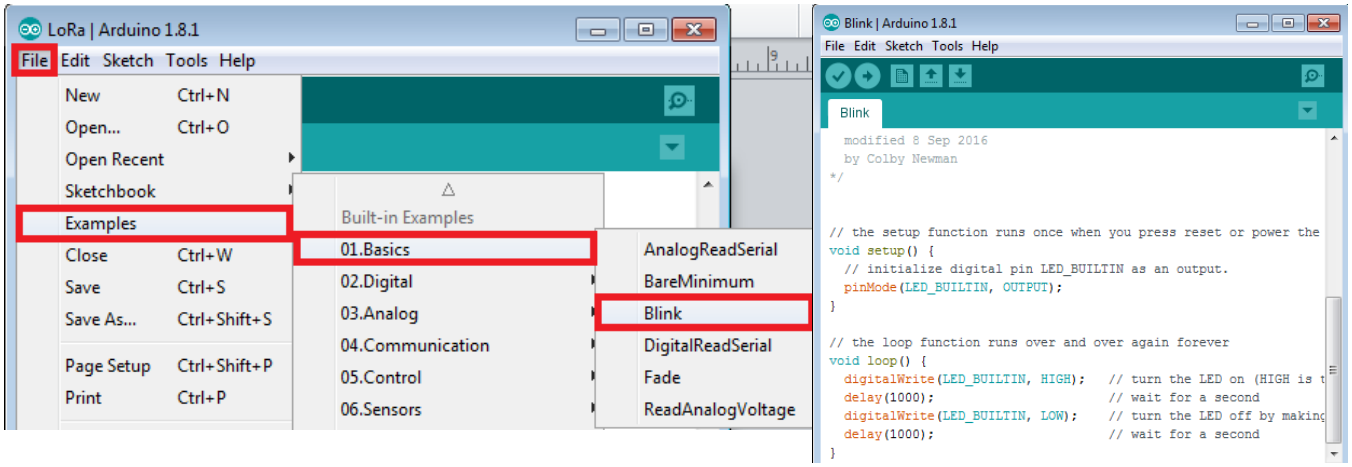
For your reference and if you want to get some knowledge on the Arduino Foundations, you can visit the Arduino Getting Started page https://www.arduino.cc/en/Guide/HomePage



Step 5: Explore a basic sketch and let's blink an LED

The Arduino IDE has some examples built in.

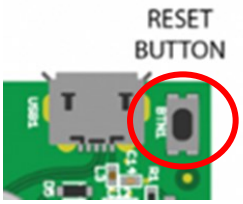
a. **Open** the basic blink sketch. *File -> Examples -> 01. Basics -> Blink*



b. **Verify/Compile** the basic blink sketch
The first step to getting a sketch ready for transfer over to the Arduino is to Verify/Compile it. That means check it over for mistakes and then translate it into an application that is compatible with the Arduino hardware.

Verify/Compile

c. **Press** the reset button (near the USB connector) twice within a second to place the ExpLoRer board into bootloader mode and to mount a new COM port dedicated for uploading the sketch.



d. **Select** the new COM port within the Arduino IDE by going to *Tools -> Port COM*

e. **Upload** the sketch to the board. The built-in LED starting to blink every second.

Upload

Step 6: Use the hardware serials embedded on the ExpLoRer Starter Kit

The ExpLoRer board has 4 hardware serials defined like this:

- **SerialUSB** is for debugging over the USB cable
- **Serial** is attached to pin D1/TX and D0/RX of the internal microcontroller
- **Serial1** is connected to the RN4871 Bluetooth Low Energy module
- **Serial2** is connected to the RN2xx3 LoRaWAN module

The sketch starts directly after uploading new code or when connected to a power source. After opening a Serial Monitor, the code will not reset. The following code may be added to your sketch if you want to wait for a Serial Monitor before pursuing the execution of the program.

```
void setup()
{
   // put your setup code here, to run once:
   // wait for SerialUSB or start after 10 seconds
   while ((!SerialUSB) && (millis() < 10000));
   SerialUSB.begin(57600) ;
   Serial.begin(57600) ;
   Serial1.begin(115200) ;
   Serial2.begin(57600) ;
}

void loop()
{
   // put your main code here, to run repeatedly:
}
```

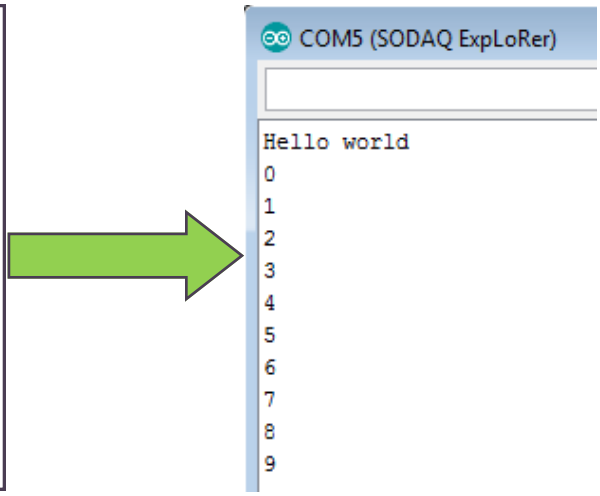
Step 7: Use the Serial Monitor

The Arduino IDE has a feature that can be great help in debugging sketches or controlling Arduino from your computer's keyboard. The Serial Monitor is a separate pop-up window that acts as a separate terminal that communicates by receiving and sending Serial data over USB cable in our case. The Serial Monitor can be open by clicking the icon on the far right side of the bar menu.

An example of sketch to use the Serial Monitor for debugging purpose:

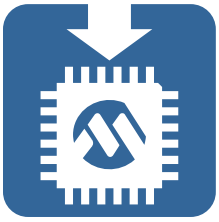
```
void setup()
{
  // put your setup code here, to run once:
  // wait for SerialUSB or start after 10 seconds
  while((!SerialUSB) && (millis() < 10000)) ;
  SerialUSB.begin(115200) ;  // open a communication @115200bps
  SerialUSB.println("Hello world") ;  // print a single line
  for (int myCounter = 0; myCounter < 10; myCounter++)
  {
    SerialUSB.println(myCounter) ;  // print myCounter variable
  }
}

void loop()
{
  // put your main code here, to run repeatedly:
}
```



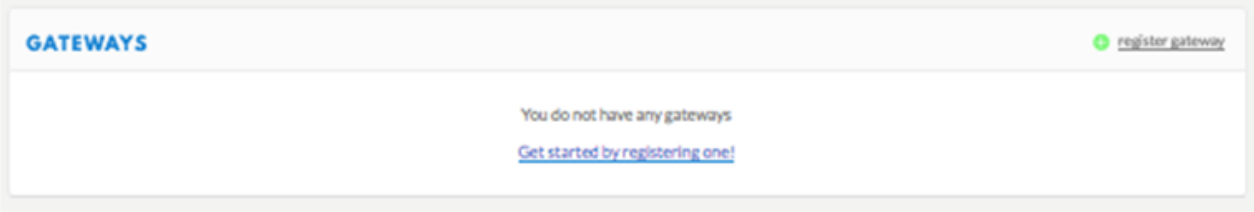
Appendix C

Register a gateway to your TTN account



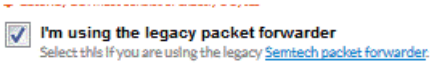
Explore TTN Gateway

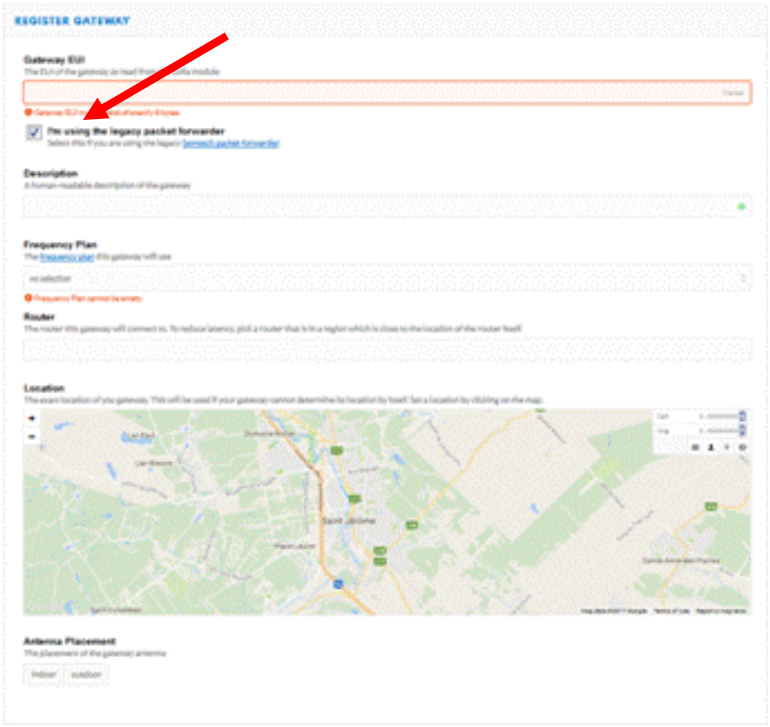
Go back to https://console.thethingsnetwork.org/
Click on the Gateway ICON, and **click** on “Get started by registering one”. This is where you will enter your gateway information



Explore TTN Gateway

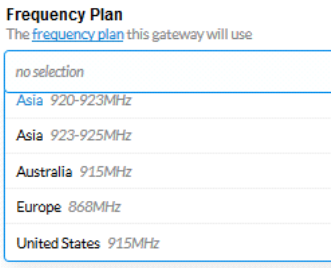
For the Microchip gateway, click on “I’m using the legacy packet forwarder”
Enter the “Gateway ID”. This is typically the unique MAC address of the Ethernet chip of the Gateway





Add a description of the Gateway and the Frequency plan.

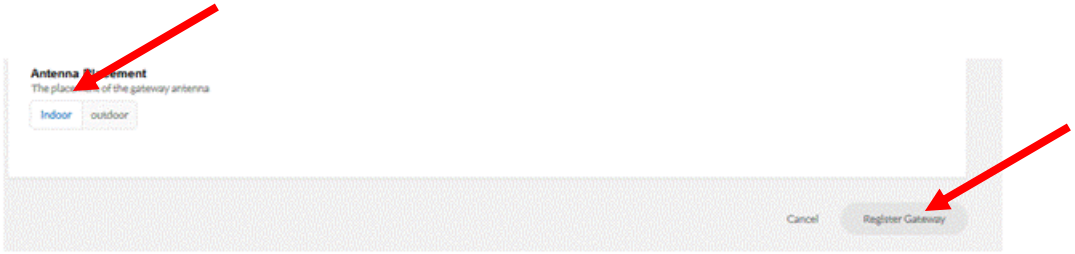
And finally select the location on the map of your Gateway.



Explore TTN Gateway

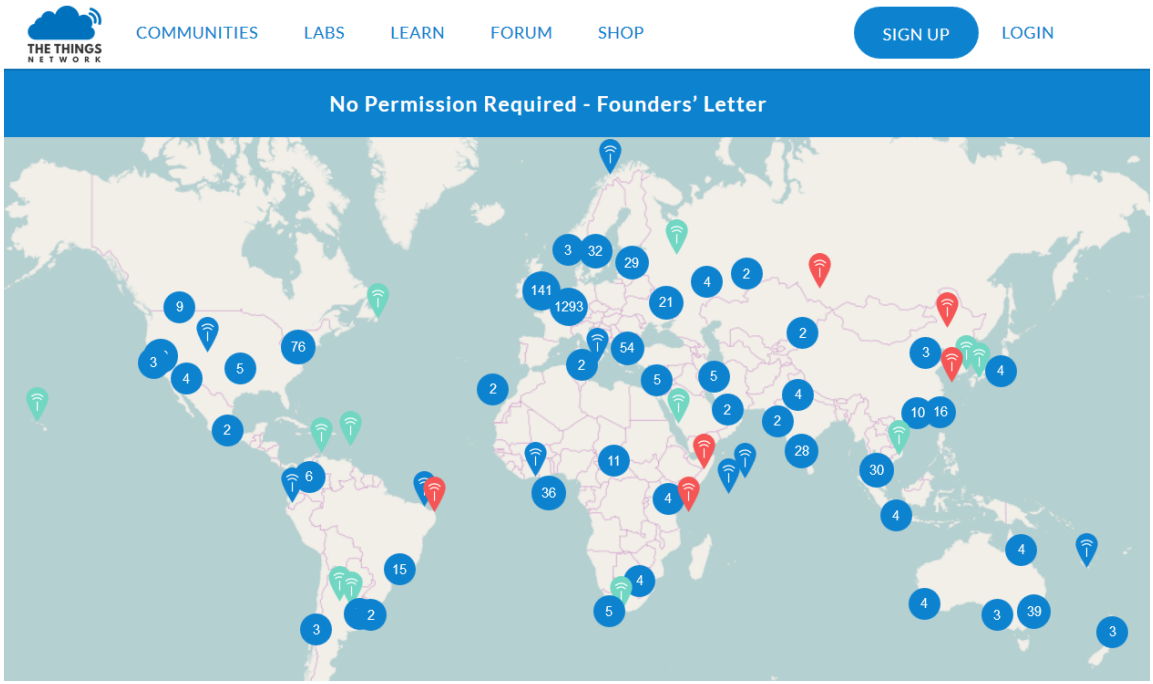
Select “Indoor” or “outdoor”

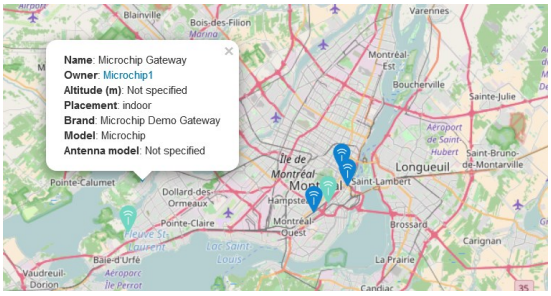
Click “register gateway”



Go on “https://www.thethingsnetwork.org/map” to view you Gateway.

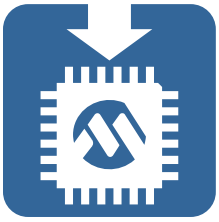
Note: I will take several minutes to show up!





Appendix D

Download and install Node-RED



Purpose

To install Node-RED on your Windows personal computer.
To be ready to create and run a Node-RED application on your personal computer.

Objective

Node-RED is a programming tool for wiring together a web application. It provides a browser-based editor that makes it easy to wire program flows together. This appendix will help you install it on your personal computer so that you can run it locally and follow the included labs.
An official online Node-Red installation guide can be found here:
http://nodered.org/docs/getting-started/installation.html

If you are using an operating system other than Windows (Linux or Mac OS), please follow the instructions in the link above to install Node-RED.

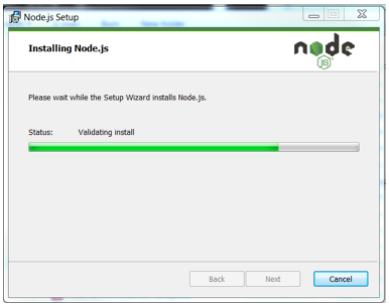
On Your Own — Node-RED Installation

Step 1: Download and install node.js

a. Download and install the latest version of node.js.

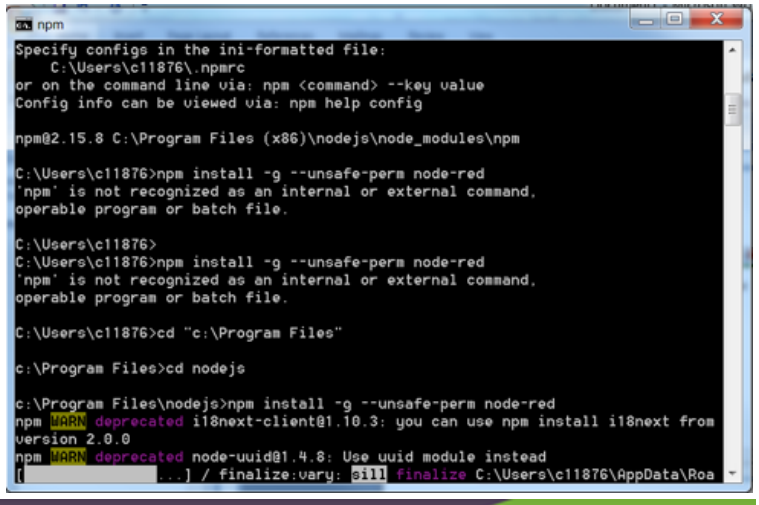
https://nodejs.org/en/download/

Node.js is required in order to run Node-RED.



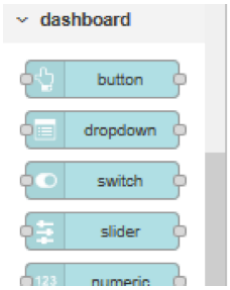
Step 2: Install Node-RED using node's package manager, npm.

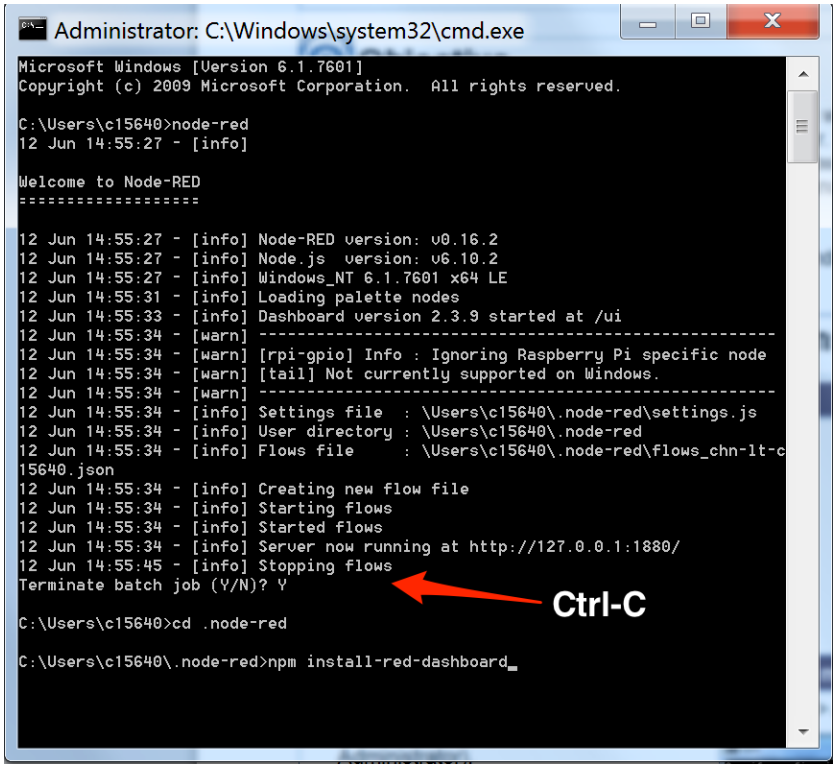
a. **Open a Windows Command Prompt** (Start Menu -> All Programs -> Accessories -> (right click) Command Prompt (Run as Administrator).
b. In CMD prompt, **go to** nodejs installation directory (C:\Program Files\nodejs).
c. **Run** the following command:
npm install -g --unsafe-perm node-red
d. Reboot Windows.



Step 3: Install Node-RED Dashboard Module

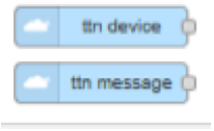
a. After rebooting Windows, open a CMD prompt again.
b. In CMD prompt, run Node-RED with the following command (this step just creates the Node-RED directories needed for the following steps): **node-red**
c. In CMD prompt, hit Ctrl-C to end the Node-RED process. You may need to type **"Y"** to confirm that you want to end the process.
d. In CMD prompt, go to Node-RED directory (C:\Users\<user>\.node-red).
e. Install the Node-RED Dashboard with the following command: **npm install node-red-dashboard**
f. This command will install nodes to Node-RED that allow graphing, etc. These nodes are shown below.

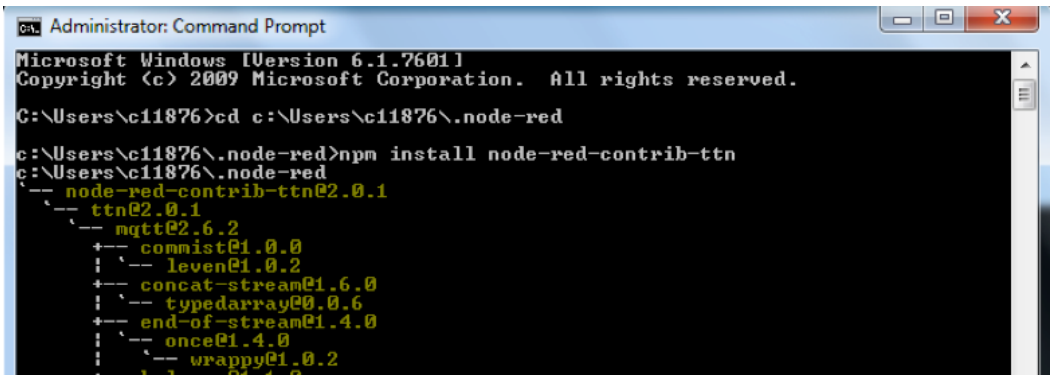




Step 4: Install Node-RED TTN Module

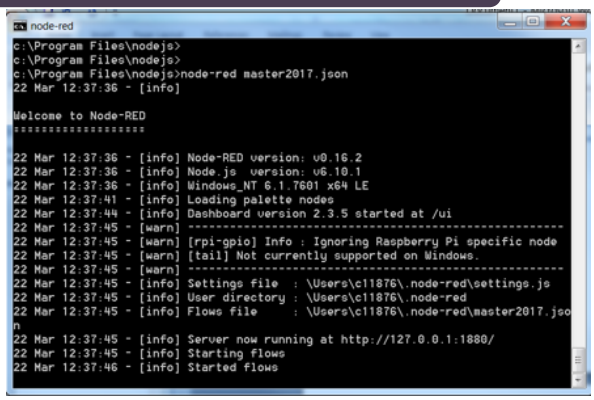
a. In CMD prompt, go to Node-RED directory (C:\Users\<user>\.node-red).
b. Install the Node-RED TTN modules with the following command: **npm install node-red-contrib-ttn**
c. This command will install nodes to Node-RED that allow graphing, etc. These nodes are shown on the left.





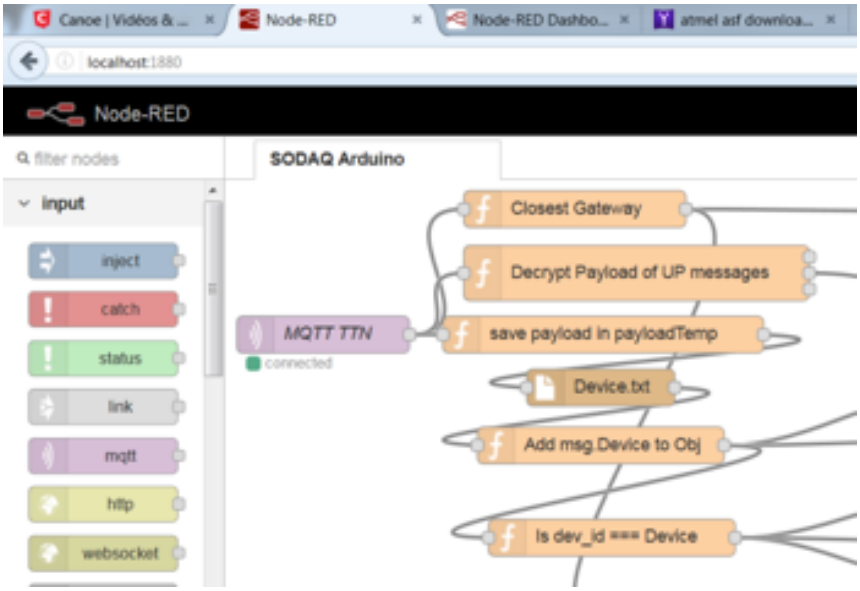
Step 5: Run Node-RED

a. In CMD prompt, go to Node-RED directory (C:\Users\<user>\.node-red).
b. Run Node-RED with the following command: **node-red**
c. You can run a specific flow with the following command:
node-red <flow-name.json>

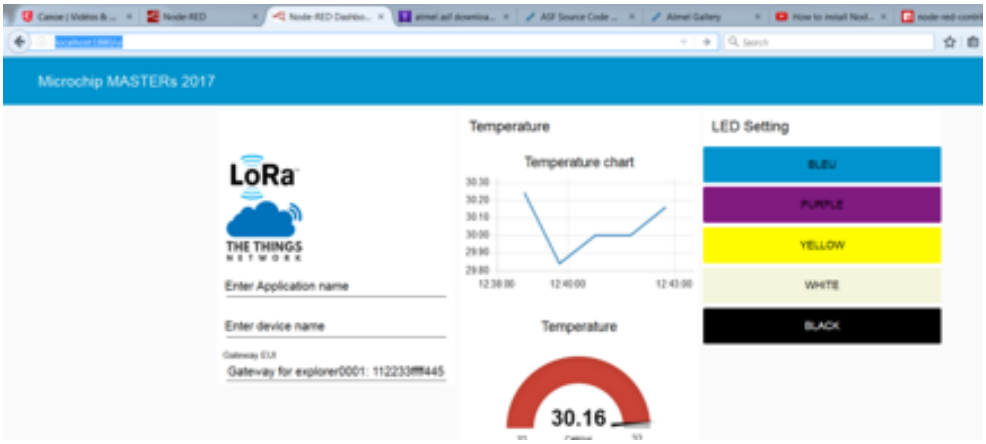


Step 6: View and Edit Node-RED Flow

a. After running Node-Red (see Step 5), **open** a browser.
b. **Type** the following URL: htt**p://localhost:1880**

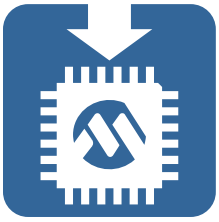


c. To view the data dashboard, **type** the following URL in your browser:
http://localhost:1880/ui



Appendix E

Labs solutions



Lab 1

```
#include <LoRa.h>
#define debugSerial SerialUSB
#define loraSerial  Serial2

// Variables will contain your personal OTAA Activation Keys
uint8_t devEUI[8] ;   // Device EUI
uint8_t appEUI[8] ;   // App EUI
uint8_t appKey[16] ;  // App Key
const uint8_t appKeyPrefix[8] = {0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88} ;

const uint8_t appEUIfromTTN[8] = {0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00} ;

void setup()
{
  while (!debugSerial && (millis() < 10000)) ;
  debugSerial.begin(115200) ;
  debugSerial.println("Microchip Technology ExpLoRer Starter Kit") ;
  debugSerial.println("21080_IoT3 Masters 2017 Class") ;
  debugSerial.println("Lab 1") ;

  // ------------------------------------------------------------------------
  // Init section
  // ------------------------------------------------------------------------
  // Temperature sensor
  pinMode(TEMP_SENSOR, INPUT) ;
  analogReadResolution(12) ;

  // LoRa
  LoRa.hwInit() ;
  loraSerial.begin(LoRa.getDefaultBaudRate()) ;
  LoRa.initLoRaStream(loraSerial) ;
  LoRa.swInit() ;

  // ------------------------------------------------------------------------
  // Activation keys section
  // ------------------------------------------------------------------------
  // Get the internal Hardware EUI of the LoRaWAN module
  uint8_t hwEUI[8] ;
  uint8_t len = LoRa.getHWEUI(hwEUI, sizeof(hwEUI)) ;
  if (len == 0) { debugSerial.println("Error to get HwEUI") ; while(1) ; }

  // Device EUI is the unique identifier for this device on the network
  // Assign the Hardware EUI as the devEUI key
  memcpy(devEUI, hwEUI, sizeof(hwEUI)) ;
  // App EUI is generated by the TTN Application Server
  memcpy(appEUI, appEUIfromTTN, sizeof(appEUIfromTTN)) ;
  // App Key can be a default key for all devices or can be unique per device
  // Here the key is computed by the help of the Device EUI
  // The 16-Bytes format will start by: FF EE DD CC BB AA 99 88 and
  // will finish by: the 8 Bytes of the devEUI
  memcpy(appKey, appKeyPrefix, sizeof(appKeyPrefix)) ;
  memcpy(appKey + 8, devEUI, sizeof(devEUI)) ;
}
```

Lab 1 (continue)

```
void loop()
{
  debugSerial.println("") ;
  debugSerial.print("devEUI = ") ;
  displayArrayInOneLine(devEUI, sizeof(devEUI)) ;
  debugSerial.print("appEUI = ") ;
  displayArrayInOneLine(appEUI, sizeof(appEUI)) ;
  debugSerial.print("appKey = ") ;
  displayArrayInOneLine(appKey, sizeof(appKey)) ;

  // Step 9.1 Read value from analog input
  int sensorValue = analogRead(TEMP_SENSOR) ;
  // Step 9.2 Convert the value to voltage
  float mVolts = (float)sensorValue * 3300 / 4096.0 ;
  // Step 9.3 Calibrate to 0°C
  float temp = (mVolts - 500) ;
  // Step 9.4 Convert to temperature value
  temp = temp / 10.0 ;
  // Step 9.5 Display the temperature on the serial monitor
  debugSerial.println(temp) ;
  // Step 9.6 Add a 3 sec delay before reading a new value
  delay(3000) ;
}

// ------------------------------------------------------------------------
// Display array in HEX format routine
// ------------------------------------------------------------------------
void displayArrayInOneLine(const uint8_t tab[], uint8_t tabSize)
{
  char c[2] ;
  for (uint8_t i = 0; i < tabSize; i++)
  {
    sprintf(c, "%02X", tab[i]) ;
    debugSerial.print(c) ;
  }
  debugSerial.println("") ;
}
```

Lab 3

```
#include <LoRa.h>
#define debugSerial SerialUSB
#define loraSerial  Serial2

// Variables will contain your personal OTAA Activation Keys
uint8_t devEUI[8] ;   // Device EUI
uint8_t appEUI[8] ;   // App EUI
uint8_t appKey[16] ;  // App Key
const uint8_t appKeyPrefix[8] = {0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88} ;

const uint8_t appEUIfromTTN[8] = {0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00} ;

void setup()
{
  while (!debugSerial && (millis() < 10000)) ;
  debugSerial.begin(115200) ;
  debugSerial.println("Microchip Technology ExpLoRer Starter Kit") ;
  debugSerial.println("21080_IoT3 Masters 2017 Class") ;
  debugSerial.println("Lab 3") ;

  // ------------------------------------------------------------------------
  // Init section
  // ------------------------------------------------------------------------
  // Temperature sensor
  pinMode(TEMP_SENSOR, INPUT) ;
  analogReadResolution(12) ;

  // LoRa
  LoRa.hwInit() ;
  loraSerial.begin(LoRa.getDefaultBaudRate()) ;
  LoRa.initLoRaStream(loraSerial) ;
  LoRa.swInit() ;

  // ------------------------------------------------------------------------
  // Activation keys section
  // ------------------------------------------------------------------------
  // Get the internal Hardware EUI of the LoRaWAN module
  uint8_t hwEUI[8] ;
  uint8_t len = LoRa.getHWEUI(hwEUI, sizeof(hwEUI)) ;
  if (len == 0) { debugSerial.println("Error to get HwEUI") ; while(1) ; }

  // Device EUI is the unique identifier for this device on the network
  // Assign the Hardware EUI as the devEUI key
  memcpy(devEUI, hwEUI, sizeof(hwEUI)) ;
  // App EUI is generated by the TTN Application Server
  memcpy(appEUI, appEUIfromTTN, sizeof(appEUIfromTTN)) ;
  // App Key can be a default key for all devices or can be unique per device
  // Here the key is computed by the help of the Device EUI
  // The 16-Bytes format will start by: FF EE DD CC BB AA 99 88 and
  // will finish by: the 8 Bytes of the devEUI
  memcpy(appKey, appKeyPrefix, sizeof(appKeyPrefix)) ;
  memcpy(appKey + 8, devEUI, sizeof(devEUI)) ;
```

Lab 3 (continue)

```
  // ----------------------------------------------------------------------------
  // Display activation keys section
  // ----------------------------------------------------------------------------
  debugSerial.println("") ;
  debugSerial.print("devEUI = ") ;
  displayArrayInOneLine(devEUI, sizeof(devEUI)) ;
  debugSerial.print("appEUI = ") ;
  displayArrayInOneLine(appEUI, sizeof(appEUI)) ;
  debugSerial.print("appKey = ") ;
  displayArrayInOneLine(appKey, sizeof(appKey)) ;

  // ----------------------------------------------------------------------------
  // Network Activation section
  // ----------------------------------------------------------------------------
  bool joinRes = 0 ;
  uint8_t joinTentative = 0 ;
  do
  {
    setRgbColor(0x00, 0x00, 0xFF) ;
    debugSerial.println("Try to join the LoRa network through OTA Activation") ;
    // Step 3 Start the join procedure and put the result into joinRes
    joinRes = LoRa.joinOTAALoRaNetwork(1, devEUI, appEUI, appKey, true, 3) ;
    debugSerial.println(joinRes ? "Join Accepted." : "Join Failed! Trying again
after 3 seconds.") ;
    if (!joinRes)
    {
      setRgbColor(0xFF, 0x00, 0x00) ;
      joinTentative ++ ;
      delay(3000) ;
    }
    if (joinTentative == 3)
    {
      debugSerial.println("Not able to join the network. Stay here forever!") ;
      while(1)
      {
        setRgbColor(0xFF, 0x00, 0x00) ;
        delay(250) ;
        setRgbColor(0x00, 0x99, 0xFF) ;
        delay(250) ;
        setRgbColor(0xFF, 0xFF, 0xFF) ;
        delay(250) ;
      }
    }
  } while (joinRes == 0) ;
  setRgbColor(0x00, 0xFF, 0x00) ;
  delay(3000) ;
}
```

Lab 3 (continue)

```
void loop()
{
  int sensorValue = analogRead(TEMP_SENSOR) ;
  float mVolts = (float)sensorValue * 3300 / 4096.0 ;
  float temp = (mVolts - 500) ;
  temp = temp / 10.0 ;
  debugSerial.println(temp) ;
  delay(3000) ;
  uint8_t res = NoResponse ;
  // Step 5.1 Create the payload variable
  char payload[10] ;
  // Step 5.2 Fill the payload with temperature value
  sprintf(payload, "%.2f", temp) ;
  // Step 5.3 Send a confirmed uplink message
  res = LoRa.sendReqAck(2, (const uint8_t*)payload, strlen(payload), 3) ;
  switch (res)
  {
    case NoError:
      debugSerial.println("Successful transmission.") ;
      setRgbColor(0x00, 0xFF, 0x00) ;
      delay(2000) ;
      setRgbColor(0x00, 0x00, 0x00) ;
      break;
    case NoResponse:
      debugSerial.println("There was no response from the device.") ;
      setRgbColor(0xFF, 0x00, 0x00) ;
      break ;
    case Timeout:
      debugSerial.println("Connection timed-out. Check your serial connection to
the device! Sleeping for 20sec.") ;
      setRgbColor(0xFF, 0x00, 0x00) ;
      delay(20000) ;
      break ;
    case PayloadSizeError:
      debugSerial.println("The size of the payload is greater than allowed.
Transmission failed!") ;
      setRgbColor(0xFF, 0x00, 0x00) ;
      break ;
    case InternalError:
      debugSerial.println("Oh No! This shouldn't happen. Something is really
wrong! Try restarting the device!\r\nThe program will now halt.") ;
      setRgbColor(0xFF, 0x00, 0x00) ;
      while (1) {} ;
      break ;
    case Busy:
      debugSerial.println("The device is busy. Sleeping for 10 extra seconds.");
      delay(10000) ;
      break ;
    case NetworkFatalError:
      debugSerial.println("There is a non-recoverable error with the network
connection. You should re-connect.\r\nThe program will now halt.");
      setRgbColor(0xFF, 0x00, 0x00) ;
      while (1) {} ; break ;
```

Lab 3 (continue)

```
    case NotConnected:
      debugSerial.println("The device is not connected to the network. Please
connect to the network before attempting to send data.\r\nThe program will now
halt.");
      setRgbColor(0xFF, 0x00, 0x00) ;
      while (1) {} ;
      break ;
    case NoAcknowledgment:
      debugSerial.println("There was no acknowledgment sent back!");
      setRgbColor(0xFF, 0x00, 0x00) ;
      break ;
    default:
      break ;
  }
  // Step 5.4 Add a 20 sec delay before restarting the loop()
  delay(20000) ;
}

// ------------------------------------------------------------------------
// Display array in HEX format routine
// ------------------------------------------------------------------------
void displayArrayInOneLine(const uint8_t tab[], uint8_t tabSize)
{
  char c[2] ;
  for (uint8_t i = 0; i < tabSize; i++)
  {
    sprintf(c, "%02X", tab[i]) ;
    debugSerial.print(c) ;
  }
  debugSerial.println("") ;
}

// ------------------------------------------------------------------------
// LED routines
// ------------------------------------------------------------------------
#define COMMON_ANODE  // LED driving method
void setRgbColor(uint8_t red, uint8_t green, uint8_t blue)
{
  #ifdef COMMON_ANODE
    red = 255 - red ;
    green = 255 - green ;
    blue = 255 - blue ;
  #endif
  analogWrite(LED_RED, red) ;
  analogWrite(LED_GREEN, green) ;
  analogWrite(LED_BLUE, blue) ;
}
void turnBlueLedOn(){
  digitalWrite(LED_BUILTIN, HIGH) ;
}
void turnBlueLedOff() {
digitalWrite(LED_BUILTIN, LOW) ;
}
```