# Building a LoRa®-based Device End-to-End with Arduino

Semtech

August, 2020

# Introduction

In this guide we will walk you through building your own LoRa®-based node from end-to-end using an Arduino board and Dragino shield. We are using a moisture sensor in this tutorial, but there are many potential uses for LoRa devices; alternative sensors for your node could include temperature, humidity, motion, air quality, light, accelerometers, and more.

## What are LoRa and LoRaWAN®?

LoRa, is an RF modulation technique which allows for extremely low powered wireless communication systems. Since the frequencies used are orders of magnitude smaller than other common wireless communication methods, such as Wi-Fi, the range of transmission is much larger; up to 10 miles or more in rural areas with a clear line of sight. Coupling this long-range transmission with ultra-low power requirements makes LoRa an ideal choice for battery-powered Internet of Things (IoT) devices with applications that necessitate communication with widely-distributed systems.

LoRaWAN® is a protocol based on an open-source specification. In short, LoRaWAN is an example of a low power wide area network that is based on LoRa radio modulation. This standard allows the wireless connection of LoRa-based devices (nodes) to the internet and helps ensure compatibility of devices among manufacturers around the world. It is even possible, as we will outline in this guide, to build your own custom LoRa end node which can transmit information to other LoRa devices and the internet via a LoRaWAN network server (LNS).

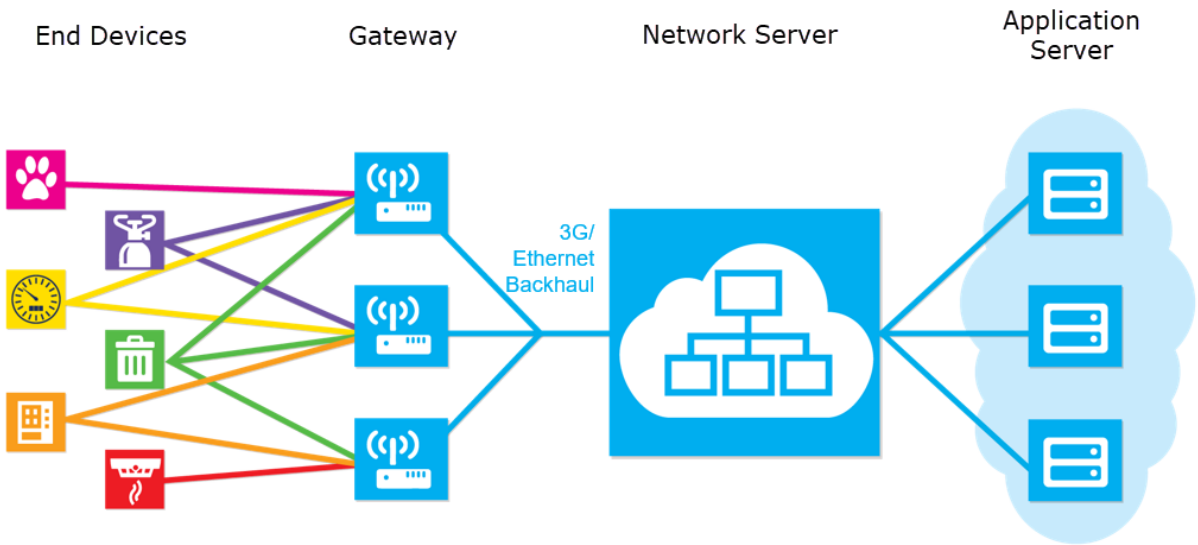


**Figure 1: LoRaWAN network high level architecture.**

The architecture of a LoRaWAN network is shown in Figure 1. The end devices – or nodes – transmit their payloads in what is known as an *uplink message,* to be picked up by a LoRaWAN gateway. A key aspect of LoRaWAN is that a device is not associated with a single gateway and its payloads may be picked up by any number of LoRaWAN gateways within the device's range. In theory an entire city could be covered by a just a handful of gateways and messages from devices belonging to any user could be detected anywhere in the city.

The gateway then passes messages to its associated network server, usually carried out over Wi-Fi or Ethernet. However, cellular and satellite gateways are available to enable very remote solutions without relying on an internet connection. There are various options for network servers, for example third-party public network servers are provided by The Things Network (TTN) and Helium, among others. For more information about choosing your network server refer to this guide.

The LNS receives the messages from the gateway and checks whether the device is registered on its network; the LNS also carries out *deduping*, since the message may be received and uploaded by multiple gateways. The message is then forwarded to the application server on which the device is registered.

Communication is also possible in the other direction, where a message is sent as a downlink to the node. The downlink message is sent from the application server to the network server. The network server then queues the downlink, and when the device next sends an uplink message, the network server will pass the downlink message to the nearest gateway which received the uplink. The gateway then broadcasts the downlink message for the device to receive.

Common uses for a downlink message are to update a device's broadcast settings, for example, to trigger updates more or less frequently or trigger some other action, such as causing a device to open, closing a water valve or turning an A/C unit on or off.

For a detailed look at LoRa and LoRaWAN including RF modulation, more information on architecture including the technical detail on how devices join the network using a Join Server, and different classes of devices, refer to our technical paper on 'What are LoRa and LoRaWAN?'

## What Will We Build?

This guide is split into three stages:

1. We will build a LoRa-based end node to measure and communicate the moisture level in soil. This is a great example of the power of LoRa-based communication, since detecting soil moisture levels is key to many agricultural processes and is often challenging given the wide geographical areas where wireless monitoring is needed. We will set up a detector node which

will connect to a moisture sensor and then send a LoRa payload containing the moisture level in the soil at regular intervals.

2. We will build a LoRa receiver node which will receive the payload and log the moisture level to the console.

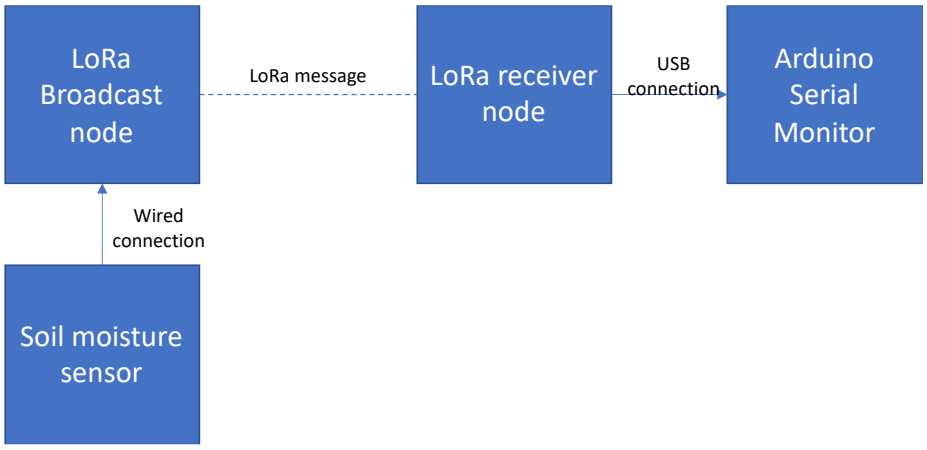


**Figure 2: Schematic of the system for stages 1 and 2.**
**The moisture level is read by the LoRa-based broadcast node which broadcasts a message. The receiver node receives the message, logging it to the Serial Monitor in the Arduino IDE**

3. We will then use the LoRaWAN specification and connect our moisture detection node to The Things Network, where we will be able to see the moisture level of the soil.

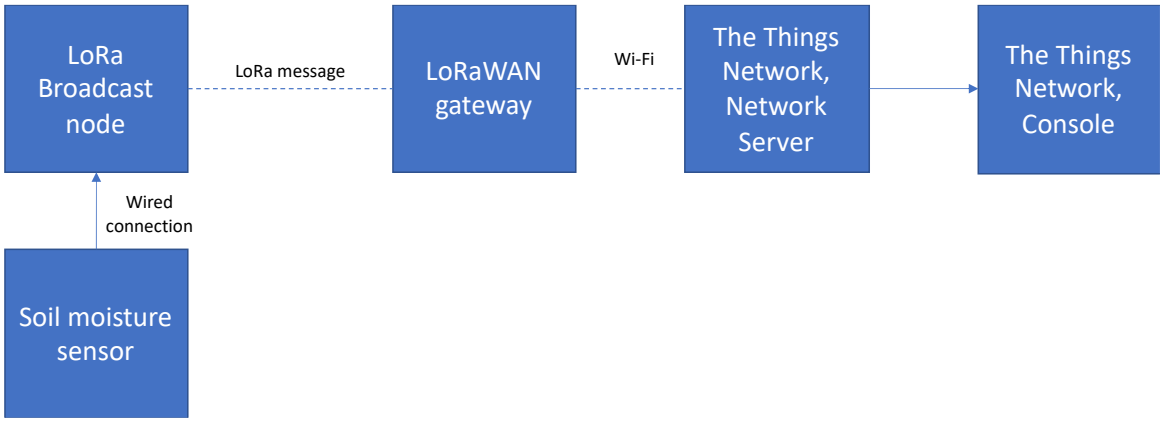


**Figure 3: Schematic of the system for stage 3.**
**The moisture level is read by the LoRa broadcast node which broadcasts a LoRa uplink. The uplink is received by the LoRaWAN gateway. The gateway sends the message over Wi-Fi to the TTN network server. The node uplink is then visible in the TTN Console.**

If you are limited by time, budget or hardware availability, it is possible to complete stage 1 and then either one of;

a. Stage 2, explore LoRa in a local setting, node-to-node
b. Stage 3, explore LoRaWAN and send messages via the TTN network server

## Objectives

By the end of this tutorial we will know how to:

- Build a LoRa-based node using the Dragino shield for LoRa devices and an Arduino Uno
- Connect a moisture sensor and configure the node to send moisture readings on a set interval
- Build a LoRa-based node to receive and log messages
- Determine a sensible uplink frequency for our device
- Build a LoRa-based end device connected to TTN via a LoRaWAN gateway

## Hardware Recommendations

There are many manufacturers of certified hardware which can be used to create your solutions. The Semtech catalog is a great reference for hardware, and contains listings from companies offering LoRa-based hardware and software products, solutions and services.

For this tutorial we recommend the following reference hardware:

### Stage 1: Hardware Recommendations for Building a Broadcasting LoRa-based Node (Required)

A. 1 x Arduino Uno (or other Arduino board compatible with the shield)
   - US: Amazon.com, Arrow, Mouser
   - UK: CPC, Amazon.co.uk, RS Components

B. 1 x Dragino's Arduino Shield featuring LoRa technology, local variant
   - US (915Mhz model): Amazon.com, Robotshop.com
   - UK (868Mhz model): CPC, Amazon.co.uk

C. 1 x Grove Moisture Sensor by Seeed Studio (other sensors are available but may require soldering)
   - US: Amazon.com, Arrow, Mouser
   - UK: CPC, RS Components

D. 1 x Grove 4 pin JST to male jumper cable (to connect the moisture sensor to the Arduino board)
   - US: Amazon.com, Mouser
   - UK: CPC, Amazon.co.uk

E. 1 x USB-B Male to USB-A Male cable (to connect the Arduinos to the computer)
   - US: Amazon.com
   - UK: CPC

F. 1 x 9V battery
   - US: Amazon.com

- UK: CPC

G. 1 x Battery clip to barrel jack connector (or other compatible portable power supply for the broadcasting Arduino)
   - US: Amazon.com, Arrow
   - UK: Amazon.co.uk, Robotshop

## Stage 2: Hardware Recommendations for Building a Receiver LoRa-based Node to Enable Node-to-Node Communication

This stage is optional, although if you choose not to complete it, you should still complete stage 3.

A. 1 x Arduino Uno (or other Arduino board compatible with the shield)
   - US: Amazon.com, Arrow, Mouser
   - UK and EU: CPC, Amazon.co.uk, RS Components

B. 1 x Dragino's Arduino Shield featuring LoRa technology, local variant
   - US (915Mhz model): Amazon.com, Robotshop.com
   - UK (868Mhz model): CPC, Amazon.co.uk

## Stage 3: Hardware Requirements for Connecting the LoRa-based Device to a Network Server (Optional)

This stage is optional, although if you choose not to complete it you should still complete stage 2.

A. 1 x The Things Indoor LoRaWAN gateway
   - US (915Mhz model): Adafruit
   - UK and EU (868Mhz model): Connected Things Store

## Alternatives

If you have a different **Ardunio board**, you may use that instead of the Arduino Uno. However, you must ensure the shield is compatible with your board. The Dragino shield is listed as compatible with Ardunio Leonardo, Uno, Mega, and DUE.

If you are unable to acquire the Dragino shield listed above, or would prefer to choose your own, you can do so. The shield needs to be listed as compatible with your Arduino board. LoRa uses license-free bandwidth, and the frequencies are different depending on the region. To use LoRa-based radios legally, the shield must operate on, and only be used with, the frequencies designated for your region.

If you are within range of a **LoRaWAN gateway** already, you could use that instead of purchasing your own. You can find worldwide coverage and links to operator maps on the LoRa Alliance® home page.

We recommend the Wi-Fi-connected indoor LoRaWAN gateway from The Things Industries for its low cost, but there are other options:

A. More expensive gateways which can be used outdoors and/or have a greater operating range due to a more sophisticated antenna, or that can be connected via Ethernet.
B. Available single-channel gateways which are not fully LoRaWAN-compliant, but could provide a cheaper alternative to a multichannel gateway for initial development or hobbyist purposes.
C. Building your own LoRaWAN-compliant gateway using a Raspberry Pi and a Gateway HAT. TTN has published a guide on how to do this; Helium has a similar guide.

It is essential to ensure the gateway operates on the same frequency as the shield you have selected, designated for your region, and that will work with your chosen network server. TTN has a list of tested compatible gateways; Helium has a similar list.

## Software

In order to configure the Arduino nodes we will be using the Arduino IDE, which you can download for Windows, Mac OS X or Linux from the official Arduino website.

## Putting It All Together

### Stage 1: Build a broadcasting LoRa-based node

In this first stage we will build the LoRa-based moisture-sensing broadcasting node and configure it to send the moisture reading every 30 seconds.

#### Assemble the Arduino, Dragino shield and moisture sensor

If you happen to have the Grove base shield, you can add the shield to your Arduino and plug the Grove Moisture sensor directly into the A0 analog sensor port; otherwise follow along with these steps.

1. Place the shield next to the Arduino so that the both the shield and the Arduino have the same number of pins at the top as each other, and the same number of pins at the bottom (as seen in Figure 4).

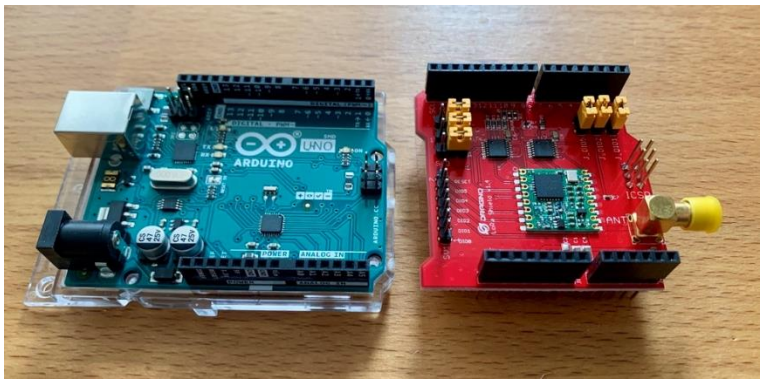


**Figure 4: Arduino and shield side-by-side, ready to mount**

2. Without turning the shield, place it directly on top of the Arduino so that the pins on the shield go into the pin headers on the Arduino and push firmly down (see Figure 5).

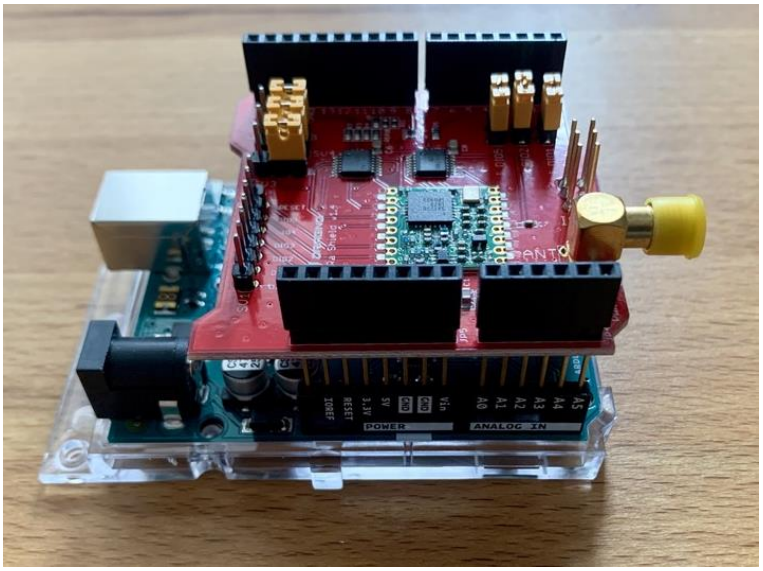


**Figure 5: Arduino and shield, mounted**

3. Attach the antenna to the shield, screwing it in clockwise (Figure 6).

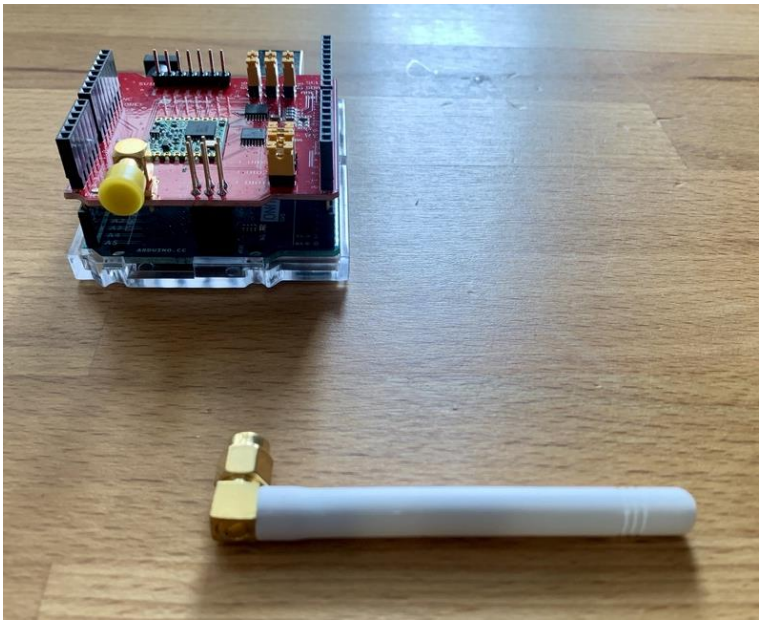

Figure 6: Assembled Arduino and shield, with antenna

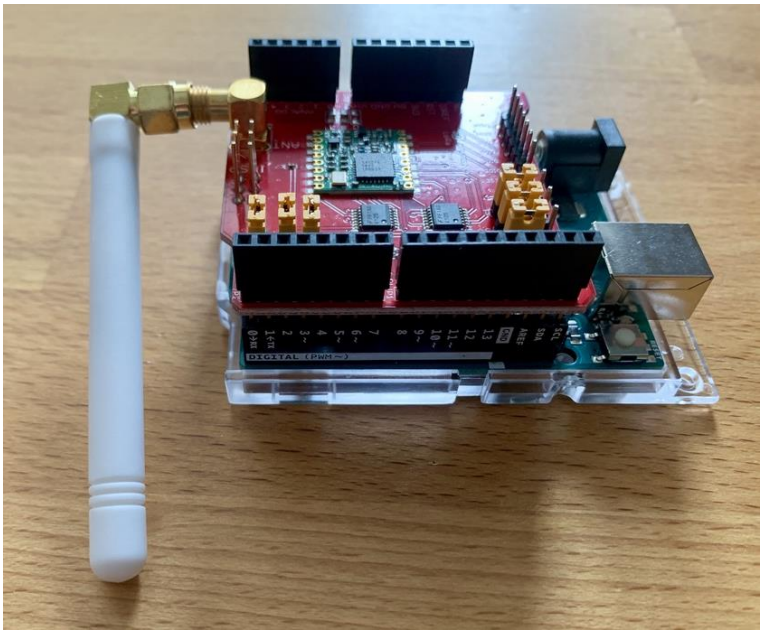

Figure 7: Screwing the antenna onto the shield

4. Locate the JST-to-male jumper cable, and the Grove Moisture Sensor.

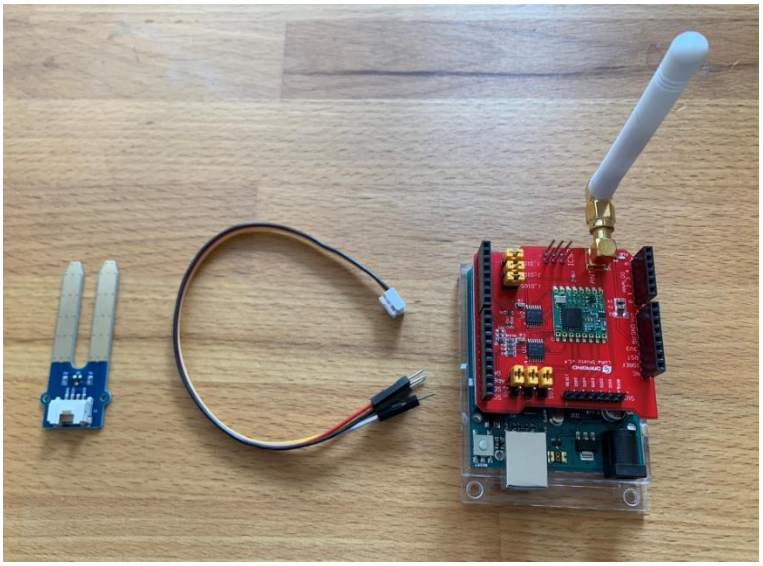

Figure 8: (left to right) Grove Moisture Sensor, JST to male jumper cable, assembled Arduino and shield

5. Connect the male end of the jumper cable to the Grove Moisture Sensor, pushing firmly to clip it into place (Figure 9).

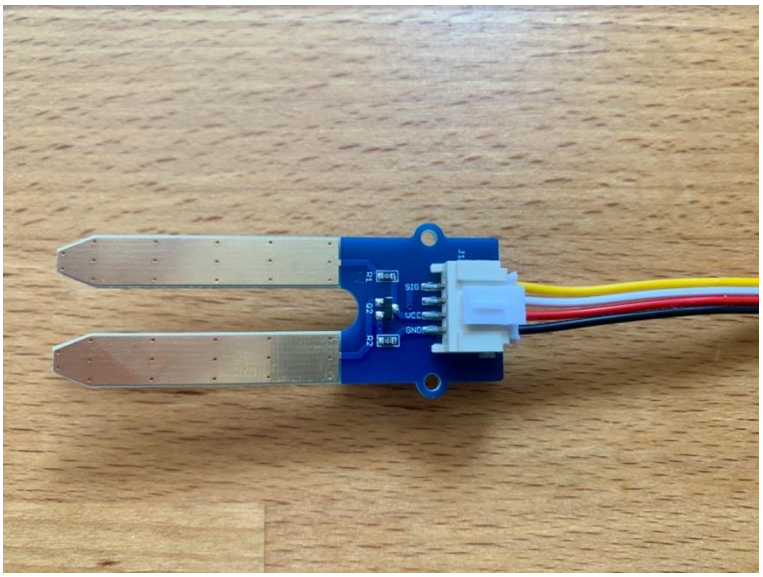


Figure 9: The Grove Moisture Sensor with male jumper connected

6. At the other end of the JST-to-male jumper cable are 4 colored wires. We need to plug these into the terminals on the shield.
    a. VCC (red wire) to the 5V terminal (Figure 10)
    b. GND (black wire) to GND terminal ( Figure 11)
    c. SIG (yellow wire) to an Analog terminal, A0 ( Figure 12)
    d. The white wire is not used, we can connect it to the second GND terminal to keep it out of the way (Figure 13)

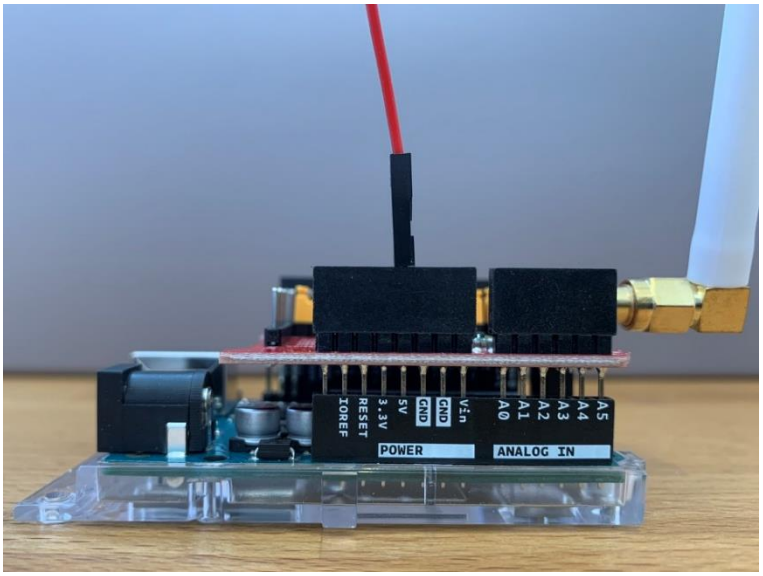


**Figure 10: Red wire connected to the 5V terminal**

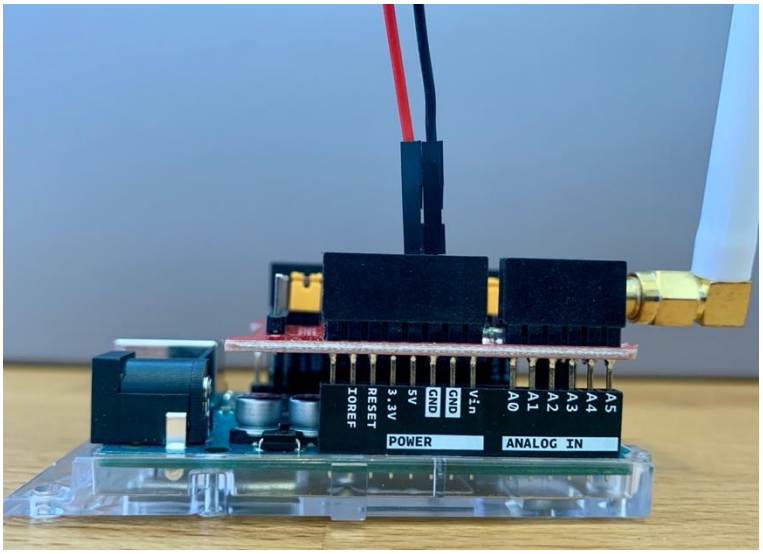


**Figure 11: Black wire, connected to a GND terminal**

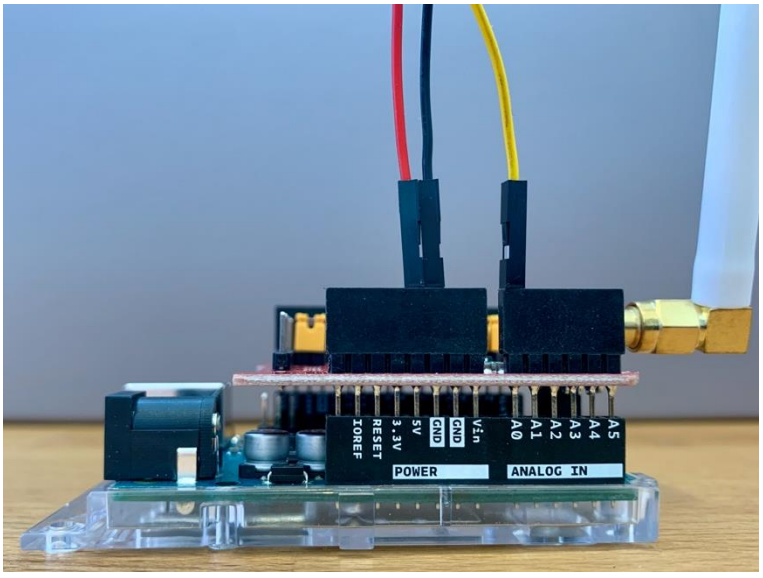


**Figure 12: Yellow wire, connected to the A0 terminal**

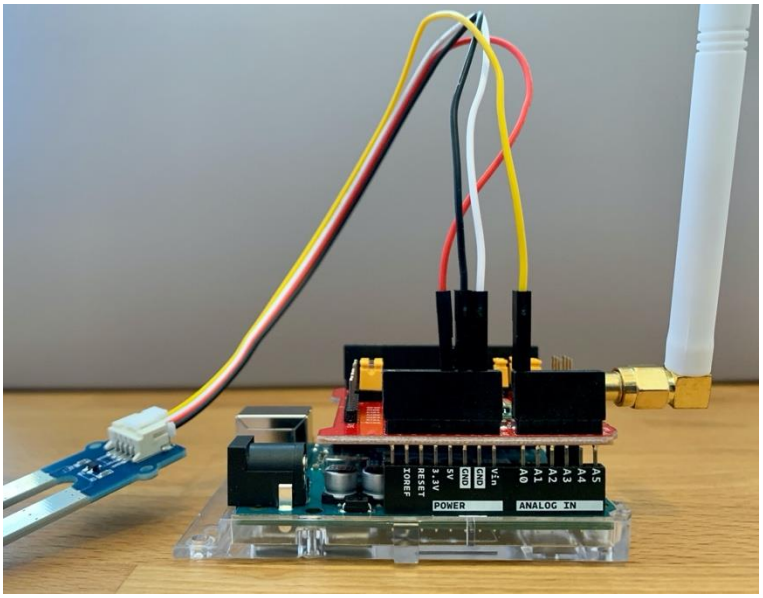


**Figure 13: All wires connected, including unused white wire**

The end node is now built.

## Setup the Arduino IDE

We need to install the open-source library, *Arduino LoRa by Sandeep Mistry*. This will allow us to send and receive data using LoRa-based radios. Then, we will select our board.

1. Download and install the latest Arduino IDE (1.8.12 at time of writing).
2. Open the Arduino IDE.
3. At the Arduino IDE menu, select **Sketch > Include Library > Manage Libraries.**

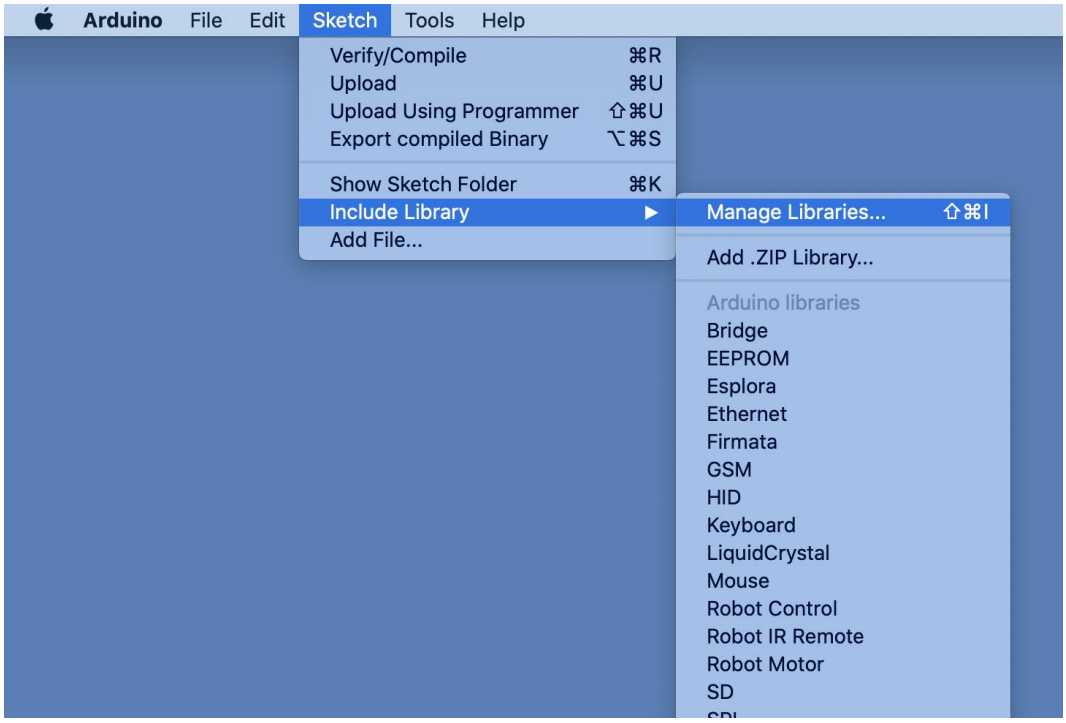


**Figure 14: Selecting 'Manage Libraries' in the Arduino IDE**

4. In the **Library Manager** window that opens, search for "LoRa."
5. Select the latest version of the LoRa library, (0.7.2 at time of writing), and select **Install.**

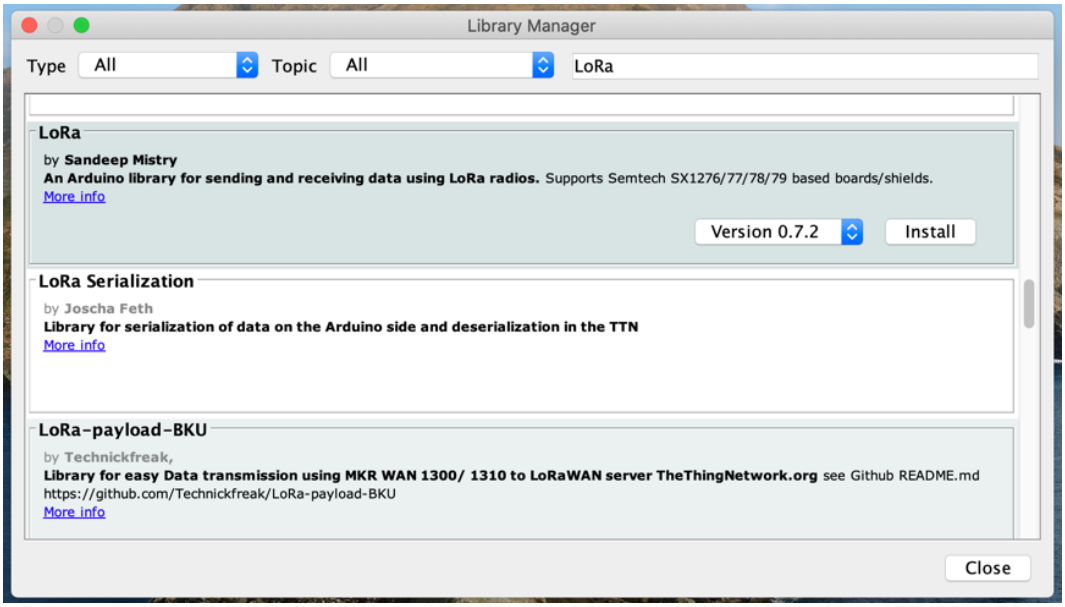


**Figure 15: Installing the LoRa library in Arduino Library Manager**

6. At the Arduino IDE menu, select **Tools > Board > Arduino Uno** (or whichever board you are using for this tutorial).

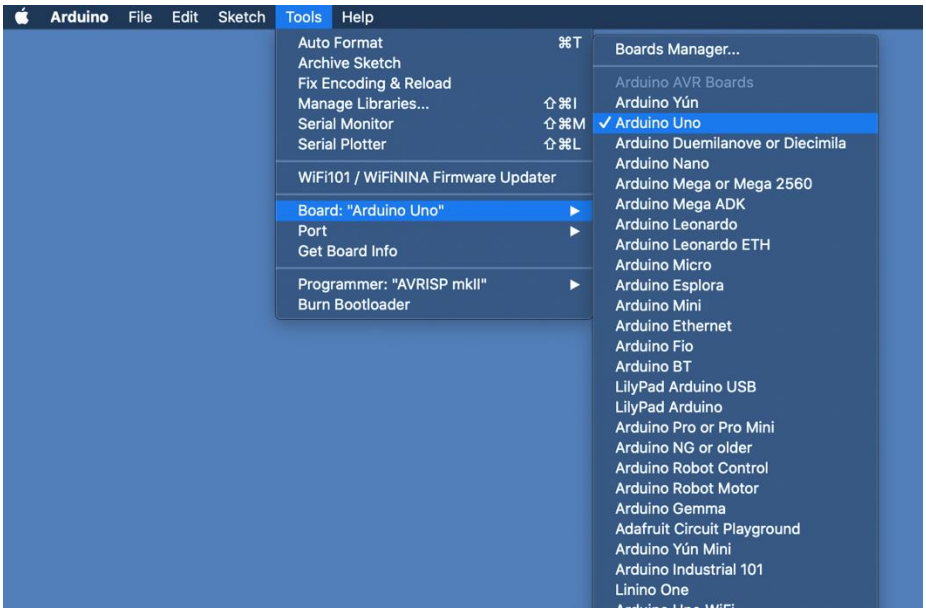


**Figure 16: Selecting your board in the Arduino IDE**

## Configure the LoRa-based Node Moisture Sensor

We are going to demonstrate sending a moisture reading at a regular set interval. If you did not purchase the second Arduino and Dragino shield, proceed to section 3.

1. At the Arduino IDE menu, select **File > New.**
2. Copy the following code snippet:

```
#include <SPI.h>

void setup() {

  Serial.begin(9600);

  while (!Serial);

  Serial.println("LoRa Sender Test");

}

void loop() {

  Serial.println("Hello.");

  delay(30000);

}
```

This is a very simple starting sketch. We will build on this in the following steps.

The `setup` function will be executed first; this function sets up the console logging so that we can see the readings logged in the Arduino IDE Serial Monitor.

The `loop` function will log a message every 30 seconds.

3. Connect the Arduino to the computer's USB port using the USB-B cable. We should see a power LED illuminate on the Arduino and the shield.
4. At the Arduino IDE menu, select **Tools > Port.** We need to select the Serial Port that the Arduino is connected to. Select the port which has the model of Arduino named alongside. If none of the ports are labelled, disconnect the Arduino board and reopen the menu; the entry which disappears should be your board. Reconnect the board, then select the entry which had disappeared.
5. At the top left of the sketch window, select the checkmark **Verify** icon. This will verify the sketch, which checks it for errors.


**Figure 17: Verifying the sketch**

6. At the top of the sketch window, select the right arrow **Upload** icon, to the right of the **Verify** icon (Figure 18). This will upload the sketch to the device.

We will see a message *Done uploading* at the end of the sketch file window, confirming the upload has taken place.


**Figure 18: Uploading the sketch**

7. At the menu, select **Tools > Serial Monitor**. The Serial Monitor window will open.
8. Select the **9600 baud** option from the drop-down menu at the bottom-right of the Serial Monitor (Figure 19).

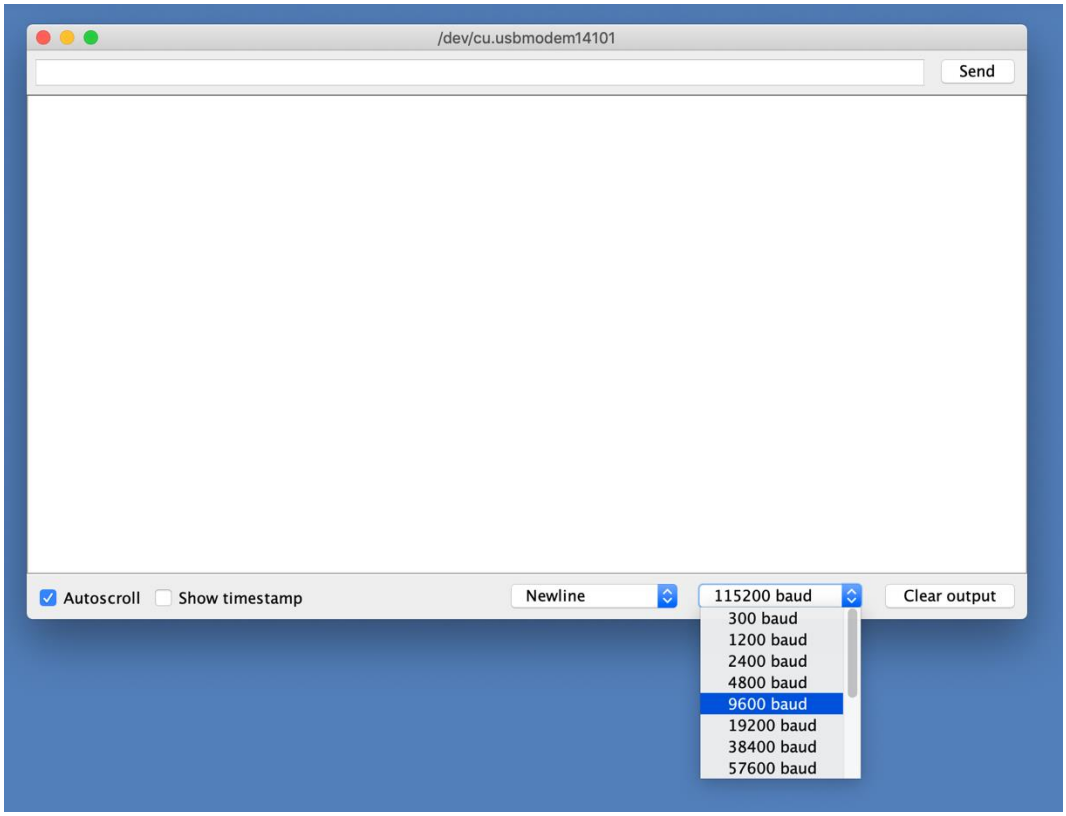


**Figure 19: Changing the baud rate in the Serial Monitor**

9. If the code uploaded successfully, we will see the *Hello* message logged to the Serial Monitor every 30 seconds.


**Figure 20: Arduino Serial Monitor showing message logging on 9600 baud.**

10. Now we will extend this sketch to send the current moisture reading from the sensor connected to the A0 analog terminal. Add the lines marked in bold to our sketch.

```
#include <SPI.h>

#define SensorPin A0

int sensorValue = -1;

void setup() {

  Serial.begin(9600);

  while (!Serial);

  Serial.println("LoRa Sender Test");

}

void loop() {

  sensorValue = analogRead(SensorPin);

  Serial.print("Reading is: ");

  Serial.println(sensorValue);

  delay(30000);

}
```

The declarations at the top of the sketch define which analog pin the sensor is connected to and set an initial value of -1.

Within the loop we then use the `analogRead` function to get the current reading from the moisture sensor via the sensor pin. The moisture sensor reading is then printed out to the Serial Monitor.

11. Verify and upload the new sketch to the Arduino. Open the Serial Monitor. You should see the real-time sensor value logged every 30 seconds, which for now will read '0'.

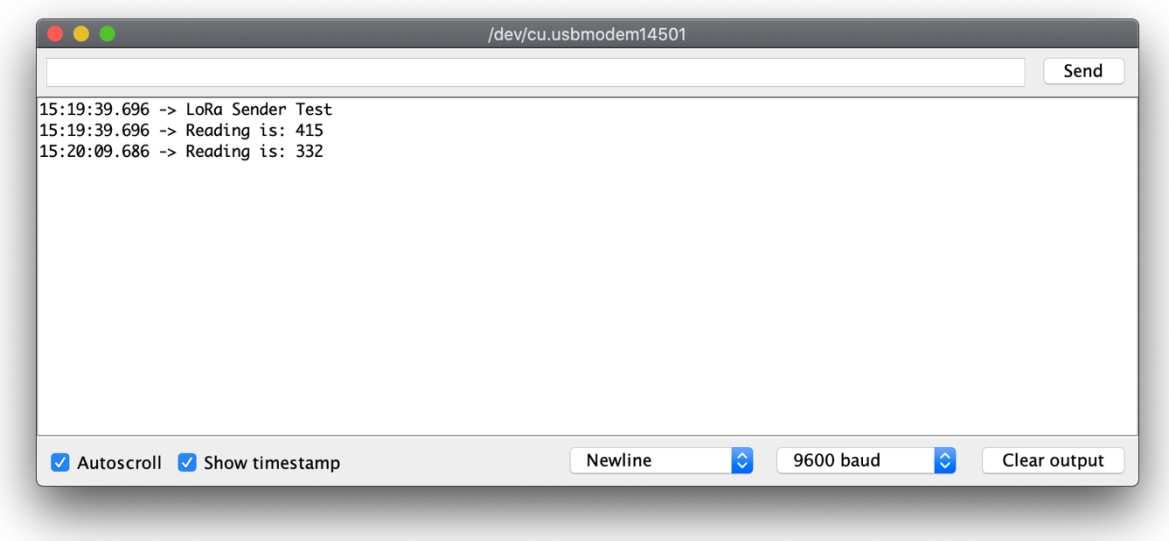


**Figure 21: Arduino Serial Monitor showing sensor readings**

12. We can test the Soil Moisture sensor by locating two pots of soil, making one pot of soil damp and keeping one dry. If you do not have a pot of soil, use a shallow glass of water, but take care to only insert the metal coated probes into the water and keep liquids away from the top of the sensor above the prongs, as well as away from the Arduino to avoid potential short circuits.

    Insert the sensor into the pot of moist soil. The sensor measures across the whole length of the probes, so make sure you push the majority of the length of the probe into the soil to get a good reading (see Figure 22).

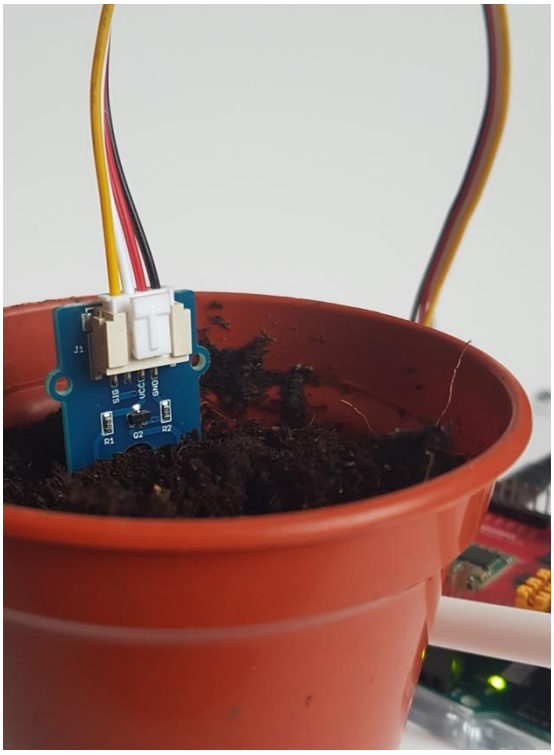

**Figure 22: Testing the assembled moisture sensor**

13. Inspect the Serial Monitor to see the readings. The higher the number, the more moisture has been detected in in the soil.

    Suggested mappings for the Grove Moisture Sensor measurement are:
    - 0: the sensor is in the open air
    - 1-299: dry soil
    - 300-699: moist soil
    - 700-949: waterlogged soil
    - 950: the sensor is immersed in water, e.g. the glass of water
14. Move the sensor to a drier pot of soil and see the numbers decrease in the Serial Monitor.
15. We will put this reading into a LoRa packet that the node will send every 30 seconds. Add the following bolded lines of code to the sketch:

```
#include <SPI.h>

// Add the LoRa library

#include <LoRa.h>

#define SensorPin A0

int sensorValue = -1;

void setup() {

  Serial.begin(9600);

  while (!Serial);

  Serial.println("LoRa Sender Test");

  // Initalize the LoRa radio on the shield.

  // NB: Replace the operating frequency of your shield here.

  // 915E6 for North America, 868E6 for Europe, 433E6 for Asia

  if (!LoRa.begin(868E6)) {

    Serial.println("Starting LoRa failed!");

    while (1);

  }

}

void loop() {

  sensorValue = analogRead(SensorPin);

  Serial.print("Reading is: ");

  Serial.println(sensorValue);

  // Transmit the LoRa packet

  LoRa.beginPacket();

  LoRa.print(sensorValue);

  LoRa.endPacket();

  delay(30000);
}
```

We first add the reference to the LoRa library.

Next, we initialize the LoRa radio on the Dragino shield with `LoRa.begin`.

We transmit the LoRa packet containing the sensor value using `LoRa.beginPacket`, `LoRa.print` and `LoRa.endPacket`.

16. Edit the code in bold to insert the operating frequency for your region. For the U.S. this is '915E6', for Asia '433E6'.

```
…
  Serial.println("LoRa Sender Test");
  // Initalize the LoRa radio on the shield.
  // NB: Replace the operating frequency of your shield here.
  // 915E6 for North America, 868E6 for Europe, 433E6 for Asia
  if (!LoRa.begin(868E6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
…
```

17. Verify and upload the sketch to the Arduino.
18. Check the Serial Monitor to verify the device is still logging readings every 30 seconds.

Well done! – We now have a working wireless LoRa-based node which broadcasts a soil moisture reading at regular intervals.

If you purchased a second Arduino and shield, continue to Stage 2: Build a LoRa-based Receiver to Enable LoRa-based Node-to-Node Communication

, to build a LoRa-based node to receive the message locally.

If you did not purchase a second Arduino and shield, continue to Stage 3: Connecting the LoRa-based Node to a Network Server

, to setup your gateway and receive the message via the cloud.

# Stage 2: Build a LoRa-based Receiver to Enable LoRa-based Node-to-Node Communication

In this stage, we will use another end node to receive messages from our broadcasting node and print the contents out to the Serial Monitor. If you did not purchase a second Arduino and shield, you can skip this step and proceed to Stage 3: Connecting the LoRa-based Node to a Network Server

.

## Assemble the Receiver Arduino

We will assemble a second Arduino and Dragino shield, which we will use to receive the message.

1. Place the Dragino shield next to the Arduino, so that the both the shield and the Arduino have the same number of pins at the top as each other, and the same number of pins at the bottom (as seen in Figure 4).
2. Without turning the shield, place it directly on top of the Arduino, so that the pins on the shield go into the pin headers on the Arduino. Push firmly down. (see Figure 5).
3. Attach the antenna to the shield, screwing it in clockwise.

## Configure the LoRa-based Receiver

Now that we have confirmed that the sensor is making readings and the code is being looped over every 30 seconds. Now, it is time to use the second node to receive the moisture readings being sent out.

1. Connect the receiver Arduino to the computer's USB port using the USB-B cable. We should see a power LED illuminate on both the Arduino and the shield.

2. Start a new sketch (**File > New**) and paste in the following code:

```
#include <SPI.h>

#include <LoRa.h>

void setup() {

  Serial.begin(9600);

  while (!Serial);

  Serial.println("LoRa Receiver Test");

  // Replace the operating frequency of your shield here.

  // 915E6 for North America, 868E6 for Europe, 433E6 for Asia

  if (!LoRa.begin(868E6)) {

    Serial.println("Starting LoRa failed!");

    while (1);

  }

}
```

We first add the reference to the LoRa library.
We initialize the LoRa radio on the shield with `LoRa.begin`.
We transmit the LoRa packet containing the sensor value using `LoRa.beginPacket`, `LoRa.print` and `LoRa.endPacket`.

3. Edit the code in bold to insert the operating frequency for your region. For the U.S. this is '915E6', for Asia '433E6'.

```
…

  Serial.println("LoRa Receiver Test");

  // Replace the operating frequency of your shield here.

  // 915E6 for North America, 868E6 for Europe, 433E6 for Asia

  if (!LoRa.begin(868E6)) {

    Serial.println("Starting LoRa failed!");

    while (1);

…
```

4. Update the sketch to add the following bolded lines, which will parse and LoRa packets the node receives:

```
…
    while (1);
  }
}
void loop() {
  // see if a packet was received
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
    // if a packet size is defined, a packet was received
    Serial.println("");
    Serial.println("Received packet!");
   }
}
```

The `loop` continually attempts to parse any LoRa packets. If a message is received, the `packetSize` will be returned and a message will be printed to the Serial Monitor.

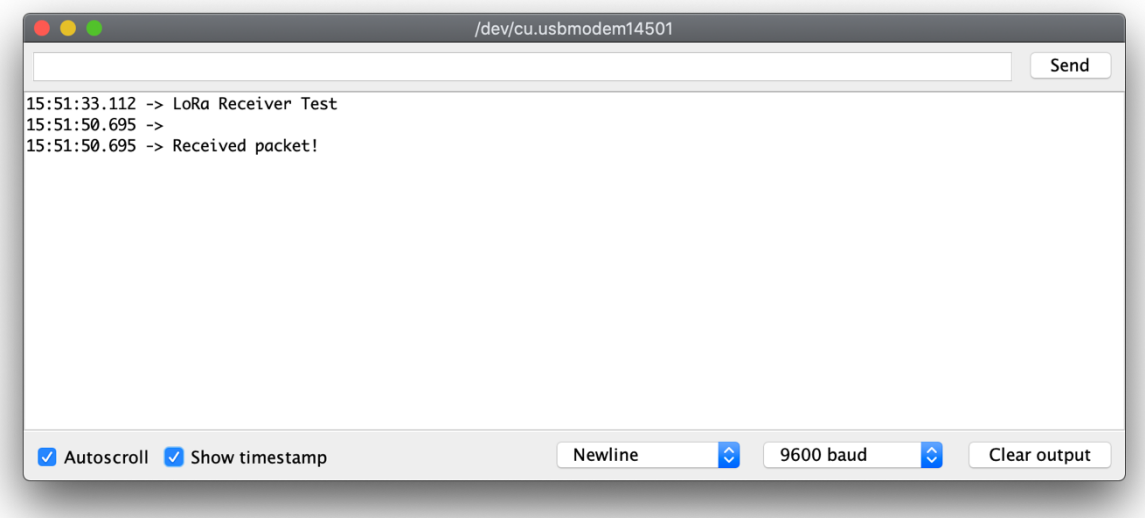


**Figure 23: Arduino Serial Monitor showing the receipt of LoRa packets**

5. To read out the payload sent in the packet, update the sketch to add the following bolded lines:

```
…
void loop() {
  // see if a packet was received
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
    // received packet
    Serial.println("");
    Serial.println("Received packet: ");
    // read the packet
    String message = "";
    while (LoRa.available()) {
      message += (char)LoRa.read();
    }
    // print the Packet and RSSI
    Serial.println(message);
    Serial.print("RSSI: ");
    Serial.println(LoRa.packetRssi());
  }
}
```

We use `LoRa.available` and `LoRa.read` to read each character of the packet, printing them to the Serial Monitor.

We use `LoRa.packetRssi` to print the Received Signal Strength Indicator (RSSI). RSSI is measured in dBm and is the received signal power in milliwatts. The closer the measurement is to 0, the better, indicating a strong signal.

6. Plug the Arduino into the computer using the USB-B cable.
7. Verify and upload the code to the Arduino.
8. Locate the broadcasting node we built in Stage 1.
9. Connect the 9-volt battery to the battery clip, and the barrel jack into the barrel jack of the broadcasting node. For now, keep the broadcasting node close to the receiving node.

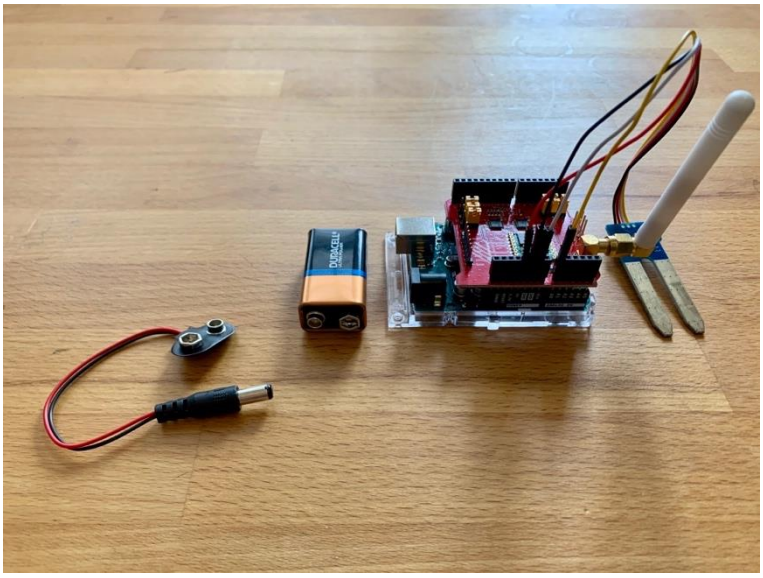

Figure 24: Left to right, battery clip, 9-volt battery, broadcasting node

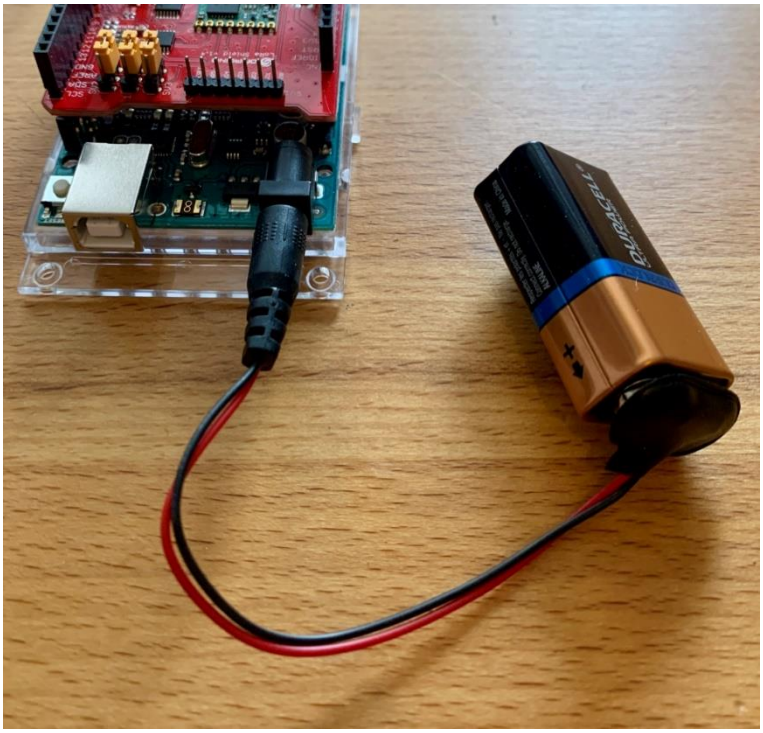

Figure 25: Battery clip connected to 9-volt battery, barrel jack connected to Arduino

10. Check the Serial Monitor. If all is well, we will see received packets being logged out every 30 seconds. This means the receiving node is receiving the packets from the broadcasting node (Figure 26).

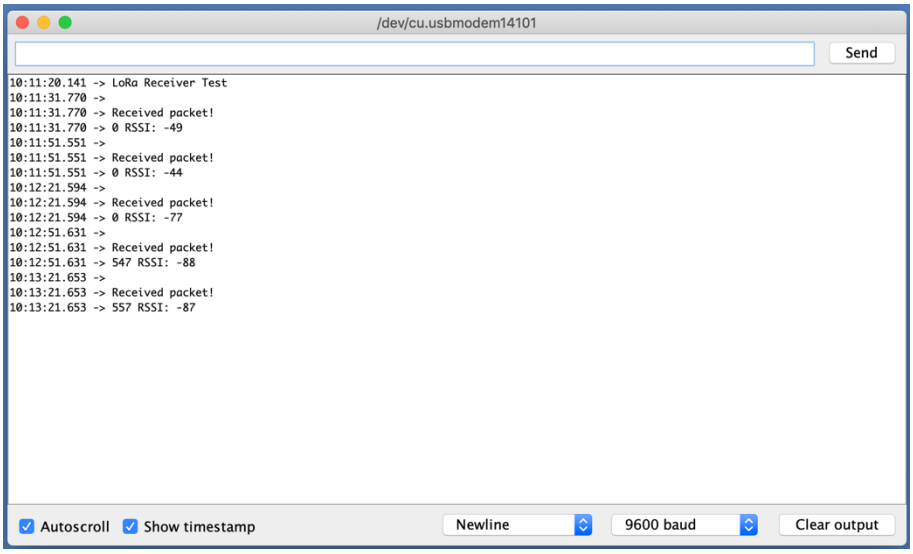


**Figure 26: Arduino Serial Monitor showing received packet values**

11. Move the broadcasting node away from the receiving node. Return to the Serial Monitor and observe the message and the change in RSSI.
12. See how far you can maintain a signal by moving the nodes farther apart. Using a more powerful radio, a much wider range can be achieved, but even with the shield, we can observe the range is much greater than other protocols, such as Wi-Fi.
13. Unplug the battery pack from the broadcasting node when you are finished to preserve battery life. In a real-world device, you would make use of sleep functionality to minimize battery consumption during broadcasts.

We have now successfully set up the broadcasting node and seen the LoRa packets being received by a second receiving node.

In the next stage of this tutorial we will move on to reconfigure the broadcasting node to connect to a TTN server to demonstrate a full end-to-end solution.

## Stage 3: Connecting the LoRa-based Node to a Network Server

In the first stage, we built the LoRa-based moisture-sensing broadcasting node and configured it to send the moisture reading every 30 seconds, Now, we can reconfigure the device to allow connection to a public network server. In this case we'll be using The Things Network (TTN), but as discussed in the introduction to this guide, you can use another public network server such as Helium, or even a private network if you have access to one. Connecting to a network server is the next step to building a fully LoRaWAN-compliant IoT solution, and doing so allows messages from our LoRa-based nodes to be viewed through an application server to present data to end users.

## Setup the Gateway

If you are using The Things Network's indoor gateway, follow the official documentation from TTN to set it up. If you are using another gateway to connect to TTN then you should be able to find instructions on the manufacturer's website, TTN gateway documentation pages, or you can search and ask for help on The Things Network forum. If you are using Helium and want to connect a LoRaWAN gateway or build your own hotspot, rather than purchasing a Helium hotspot, more information can be found on the **Hotspot** section of Helium's developer site.

## Add the Device to The Things Network

1. If you do not have an account already, register a new account with TTN, or login to your existing account.
2. On the top menu click on Console and select **Applications.**

   An application allows devices to communicate outside of the network server, for example, into AWS IoT, Azure IoT or private servers.

   Each device registered to TTN must be added to an application, so we will need to add at least one to associate with your LoRa-based node.

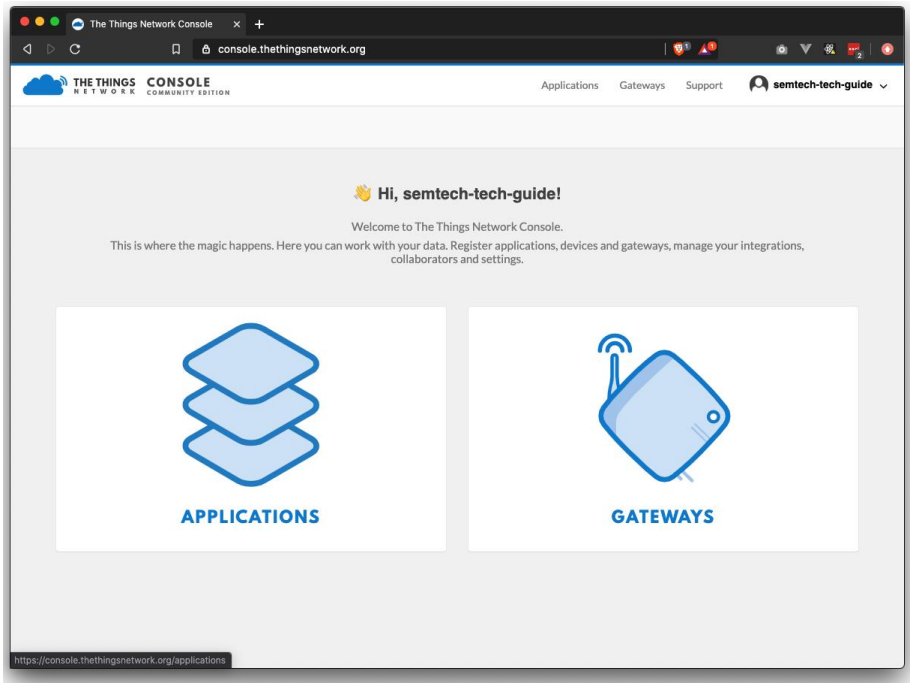


**Figure 27: TTN Console homepage**

3. Select **add application.**

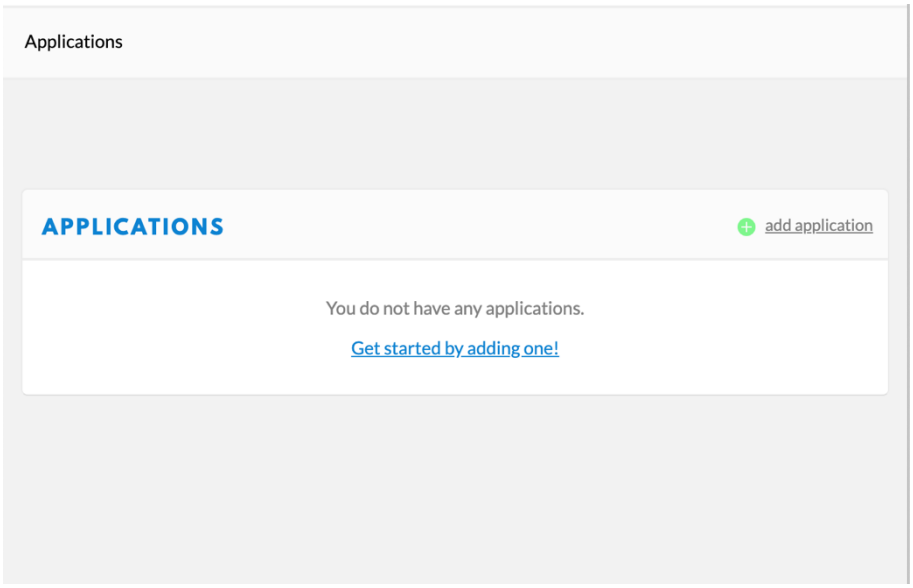


**Figure 28: TTN applications page**

4. Fill out the form as follows:
    a. **Application ID:** supply a unique ID using only lower case, alphanumerical characters and nonconsecutive '-' and '_' symbols. E.g. 'moisture-sensor-application-q123' where q123 is random. The ID must be unique throughout TTN, so you may need to try a few times.
    b. **Description:** choose your own description, e.g. 'Moisture sensor application as part of Semtech tutorial'
    c. **Application EUI:** you can use the auto-generated EUI. The application EUI (or AppEUI) globally identifies the application.
    d. **Handler registration: t**his determines the region of TTN where the packet will be transmitted. You should select the handler that matches the region you are in, 'ttn-handler-eu' in the EU, 'ttn-handler-us-west' in the U.S., 'ttn-handler-asia-se' in Asia, etc. This should be automatically detected for you.

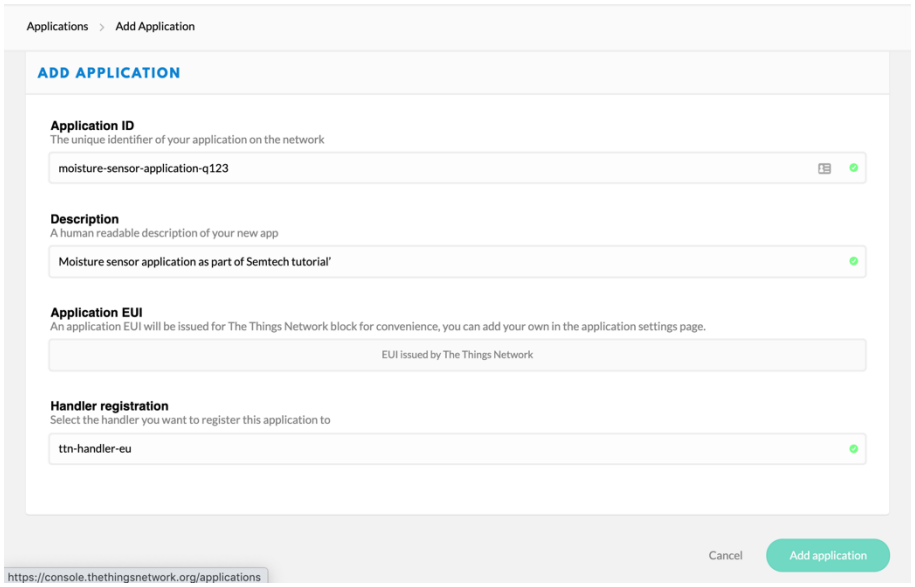


**Figure 29: Adding an application to TTN**

5. Click **Add application.**
6. If any errors are shown, for example, if the Application ID was not unique, update and click **Add application** again.
7. We are taken to the **Application Overview** page where we can see our application and its EUI. Locate the **Devices** section of the page, and click **register device.**

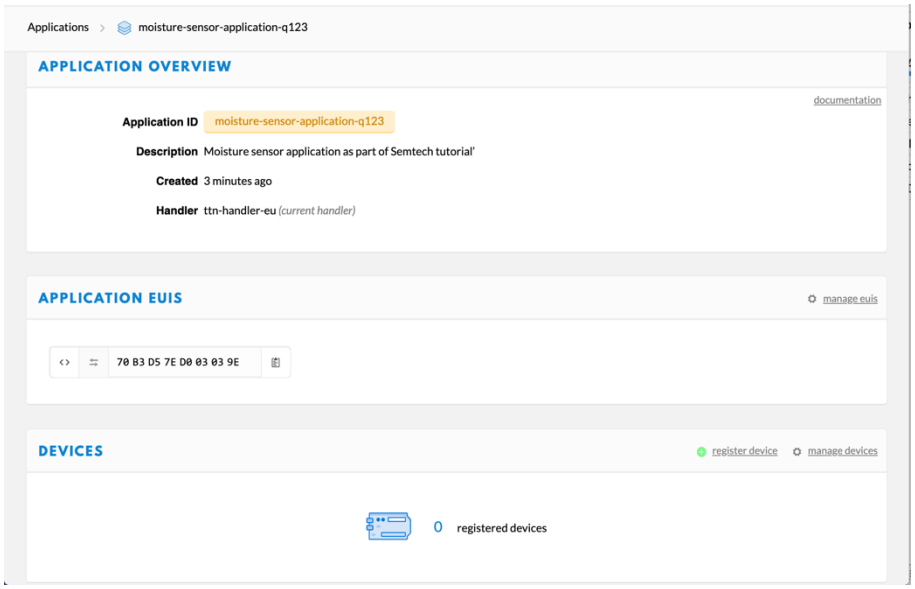


**Figure 30: Application Overview page**

8. Fill out the form as follows:
    a. **Device ID:** give the device a unique identifier for use in this application, using lower case alphanumerical characters and nonconsecutive ' –' and '_', e.g. *stg-moisture-1.*

    **Note:** The device ID cannot be changed later.

b. **Device EUI:** click the **Generate** button to the left of the input field to generate the device EUI automatically. When you purchase a device from a third-party, they will often supply you with a device EUI, and you would need to add the one supplied here.
c. **App Key:** this will be automatically generated. The app key, or 'application key' is used to secure communication between the device and the network.
d. **App EUI:** this field will be prepopulated with the App EUI of the application that we just created.

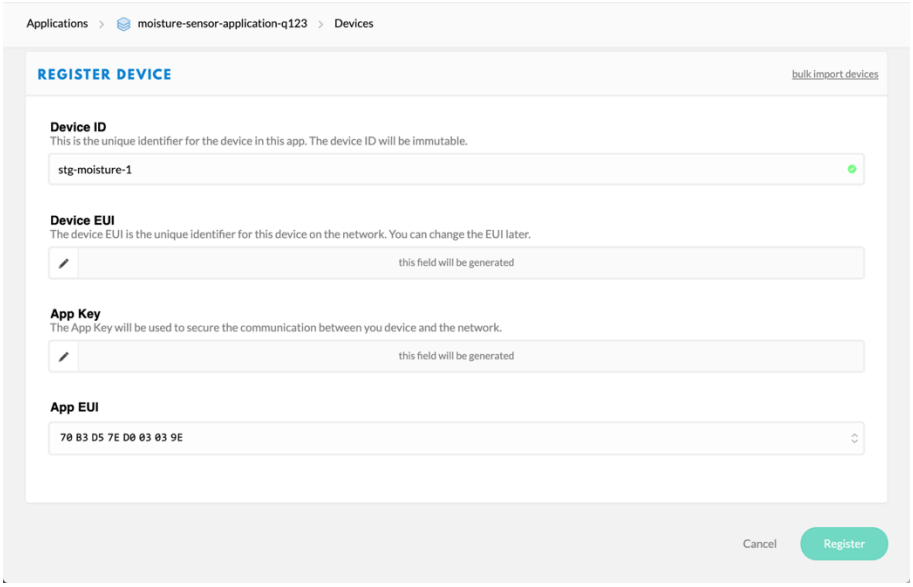


**Figure 31: Adding a device to our application**

9. Click **Register.**
10. If any errors are shown, for example, if the Device ID was not valid, update and click **Register** again.
11. We will be taken to the **Device Overview** page for our registered device.

    The Device EUI, Application EUI and App Key displayed on the **Device Overview** page will be used to configure our node and activate it via Over-The-Air Activation (OTAA). Once complete, when the node sends an uplink message the network server will know that the device is registered with itself and this application.
12. Save the following strings into another file, following the instructions below to display them in the correct format:
    a. Device EUI:
        i. click the **<>** button (1st left) to toggle between hex and C-style
        ii. click the second button from the left to toggle to lsb (least significant byte first, which reverses the bytes)
        iii. click the copy button on the far right
        iv. paste into a file, labelling this 'Device EUI'
    b. Application EUI:

i. click the **<>** button (1st left) to toggle between hex and C-style
ii. click the second button from the left to toggle to lsb format
iii. click the copy button on the far right
iv. paste into a file, labelling this 'Application EUI'

c. App Key:
i. click the **<>** button (1st left) to toggle between hex and C-style
ii. verify the format is msb (most significant byte first)
iii. click the copy button on the far right
iv. paste into a file, labelling this 'App Key'

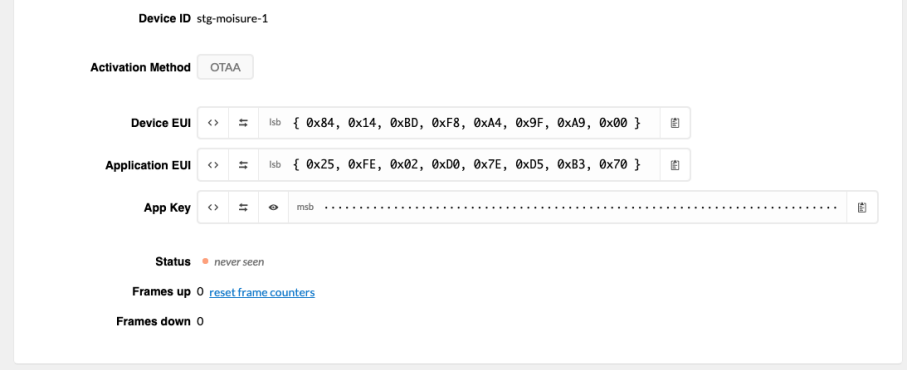


**Figure 32: Device details page on TTN showing the Device EUI and Application EUI in the format required**

The device is now registered in TTN, and we have our keys to use for OTAA.

## Setup the Arduino IDE for LoRaWAN

We need to install the open-source library MCCI LoRaWAN LMIC (LoRaWAN-MAC-in-C) that will allow us to send and receive data using LoRa-based radios, edit a config file (unless you are based in the U.S.), and select our board.

1. Open Arduino IDE.
2. At the Arduino IDE menu select **Sketch > Include Library > Manage Libraries.**
3. Search for **MCCI LoRaWAN LMIC** and install the *MCCI LoRaWAN LMIC library by IBM, Matthijs Kooijman and others*. At the time of this writing, the latest version is 3.2.0.

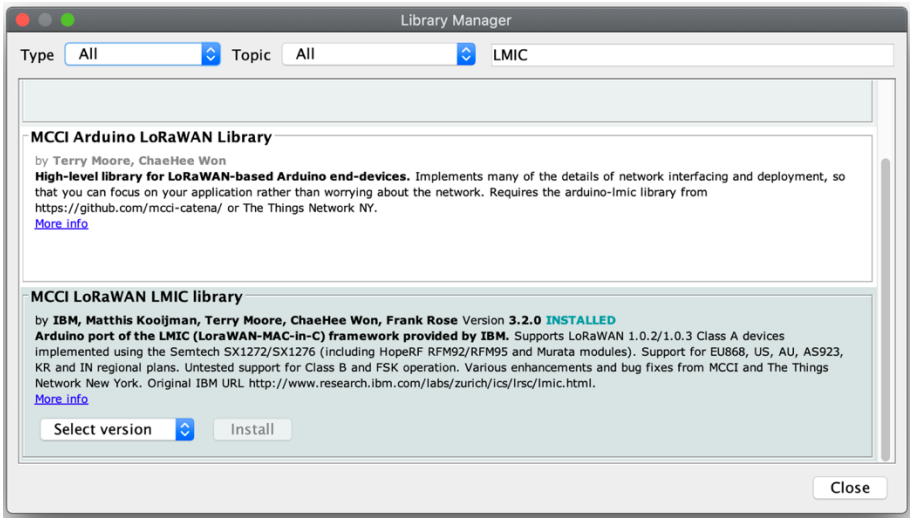


**Figure 33: Arduino IDE Library Manager showing the installed LMIC library**

4. At the menu select **Tools > Board > Arduino Uno** (or whichever board you are using for this tutorial) to ensure that the correct board is still selected.

5. The LMIC library contains a flag to toggle between region frequencies. We next need to make sure the correct frequency is selected for your region. If you are in the U.S., you do not need to perform these remaining steps, since the default region is the U.S. region.
    a. At the Arduino IDE menu select **Arduino > Preferences > Settings,** or **File > Preferences** on Windows and Linux. Locate the Sketchbook location field (see Figure 34), we will refer to this as <path to sketchbook location> On MacOS or Windows the default location is in your documents; **/Users/<username>/Documents/Arduino,** or **D:\Users\<username>\Documents\Arduino** respectively. On Linux the default location is **/home/Sketchbook**.

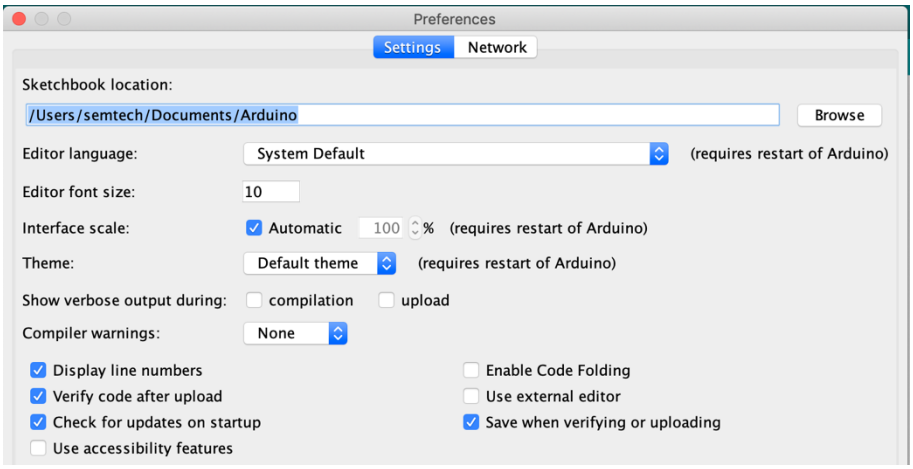


**Figure 34: Arduino IDE settings screen showing Sketchbook location**

    b. Open your code editor, or install Visual Studio Code.

c. Open the file *<path to sketchbook location>/*libraries/MCCI_LoRaWAN_LMIC_library/project_config/lmic_project_config.h. This file contains the settings for the LMIC library.

d. Edit lmic_project_config.h to comment the `#define CFG_us915 1 line`, and uncomment the frequency for your country/region.

The frequencies are:
- CFG_eu868 – EU
- CFG_us915 – U.S.
- CFG_au915 – Australia
- CFG_as923 – Asia
- CFG_in866 – India
- CFG_kr920 – South Korea

For example, to use the EU frequency, comment and uncomment as follows:

```
// project-specific definitions

#define CFG_eu868 1

//#define CFG_us915 1

//#define CFG_au915 1

//#define CFG_as923 1

// #define LMIC_COUNTRY_CODE LMIC_COUNTRY_CODE_JP /* for as923-JP */

//#define CFG_kr920 1

//#define CFG_in866 1

…
```

6. Save the lmic_project_config.h file.

## Configure the LoRa-based Node

Now, we'll update the LoRa-based node we created in Stage 1 to use the LoRaWAN protocol to broadcast the soil moisture readings over the network.

1. At the Arduino IDE menu, select **File > New.**
2. Copy the code below into the sketch. This code, and the code which follows, references the TTN-OTAA sample provided with the LMIC library, which is found here.

```
#include <lmic.h>

#include <hal/hal.h>

#include <SPI.h>

#define SensorPin A0

int sensorValue = -1;

// Pin mapping – set your pin numbers here. These are for the Dragino shield.

const lmic_pinmap lmic_pins = {

    .nss = 10,

    .rxtx = LMIC_UNUSED_PIN,

    .rst = 9,

    .dio = {2, 6, 7},

};
```

3. We import the LMIC and SPI library headers.

   We define `SensorPin`, and `sensorValue`, as in Stage 1.

   We set up the variable `lmic_pinmap` to tell the LMIC library which Arduino pins our shield uses. We have supplied the pin mapping in the code above for the Dragino shield, but if you are using a different shield, you will need to update this struct. Refer to your hardware documentation for the correct pin mappings. The settings are:

   - .nss, for the 'slave select' connection,
   - .rxtx, for controlling the antenna switch, not used by this software so set to `LMIC_UNUSED_PIN`
   - .rst, reset pin, used to reset the transceiver
   - .dio, digital I/O pins to get status information from the shield, for example when a transmission starts or is complete.

   Learn more about pin mapping at the GitHub page for the LMIC library.
4. Add the bolded lines below to the sketch to set the variables for Over-the-Air device activation (OTAA):

```
…
    .dio = {2, 6, 7},
};


// Insert Device EUI here
static const u1_t PROGMEM DEVEUI[8] = <DEVICE_EUI>;
void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}


// Insert Application EUI here
static const u1_t PROGMEM APPEUI[8] = <APPLICATION_EUI>;
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}


// Insert App Key here
static const u1_t PROGMEM APPKEY[16] = <APP_KEY>;
void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}
```

5. Edit the code to replace the fields in tags with the values saved earlier from the TTN console.
    a. **<DEVICE_EUI>**: The saved 'Device EUI'
    b. **<APPLICATION_EUI>**: The saved 'Application EUI'
    c. **<APP_KEY>**: The saved 'Application Key'

6. Check that our code is formatted as below, with the keys from TTN.

```
…

// Insert Device EUI here

static const u1_t PROGMEM DEVEUI[8] = { 0x6B, 0x1E, 0x77, 0xEB, 0xDF, 0x69, 0x20, 0x00 };

void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}



// Insert Application EUI here

static const u1_t PROGMEM APPEUI[8] = { 0x9E, 0x03, 0x03, 0xD0, 0x7E, 0xD5, 0xB3, 0x70 };

void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}



// Insert App Key here

static const u1_t PROGMEM APPKEY[16] = { 0x69, 0x21, 0xFB, 0x85, 0xE4, 0x49, 0x05, 0x97, 0x4D, 0xCF, 0x8D, 0x33, 0x9C, 0xB1, 0x37, 0x9D };

void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}
```

7. To set the frequency of broadcast, add the bolded lines below to the sketch. We have chosen to broadcast once every 150 seconds for this device.

```
…

void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}

// Schedule uplink to send every TX_INTERVAL seconds

const unsigned TX_INTERVAL = 150;
```

The duty cycle, the amount of time a device is allowed to broadcast in the LoRaWAN spectrum, is regulated by government. A duty cycle limit of 1% means that for any given time period the device may only broadcast for 1% of that time, e.g. 864 seconds within a 24-hour period. In addition, TTN has a fair usage policy limiting the uplink time per node to 30 seconds per day.

There are calculators, such as this third-party one on GitHub, that allow us to estimate our

airtime for an uplink message. Once we have completed this tutorial, we will also be able to see the estimated airtime in the TTN console at TTN console applications page **> select our application > 'Devices' > select our device > 'Data' tab > select an uplink > 'Estimated Airtime' section**. The following calculation can then be used to calculate the minimum time between uplinks:

$$delay\ in\ seconds = \frac{number\ of\ seconds\ in\ a\ day\ \times\ airtime\ of\ one\ uplink\ in\ seconds}{maximum\ airtime\ allowed\ per\ node\ in\ seconds}$$

8. Add the following bold lines to the sketch file to define the method we will use to send an uplink:

```
…
const unsigned TX_INTERVAL = 150;
void do_send(osjob_t* j){
    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {
        // Prepare upstream data transmission at the next possible time.
        sensorValue = analogRead(SensorPin);
        Serial.println("Reading is: ");
        Serial.println(sensorValue);
        // int -> byte array
        byte payload[2];
        payload[0] = lowByte(sensorValue);
        payload[1] = highByte(sensorValue);
        // transmit packet at the next available slot. The parameters are
        // - FPort, the port used to send the packet – port 1
        // - the payload to send
        // - the size of the payload
        // - if we want an acknowledgement (ack), costing 1 downlink message, 0
means we do not want an ack
        LMIC_setTxData2(1, payload, sizeof(payload), 0);
        Serial.println(F("Payload queued"));
    }
}
```

The `do_send` function contains the code from Stage 1 to read the sensor. We then convert the reading into the byte array `packet`, using lowByte and highByte to convert the integer value into a small 2-byte array. This tiny payload is then sent using `LMIC_setTxData2`.

9. Add the following bolded lines to the sketch file to handle events from the LMIC library:

```
…
        Serial.println(F("Payload queued"));
    }
}
static osjob_t sendjob;

void onEvent (ev_t ev) {
    switch(ev) {
        case EV_JOINING:
            Serial.println("EV_JOINING");
            break;
        case EV_JOINED:
            Serial.println("EV_JOINED");
            // We will disable link check mode, this is used to periodically
verify network connectivity, which we do not need in this tutorial
            LMIC_setLinkCheckMode(0);
            break;
        case EV_JOIN_FAILED:
            Serial.println("EV_JOIN_FAILED");
            break;
        case EV_REJOIN_FAILED:
            Serial.println("EV_REJOIN_FAILED");
            break;
        case EV_TXCOMPLETE:
            Serial.println("EV_TXCOMPLETE");
            // Schedule next transmission
            os_setTimedCallback(&sendjob, os_getTime() +
sec2osticks(TX_INTERVAL), do_send);
            break;
         default:
            break;
    }
}
```

The `onEvent` function is called from the LMIC library whenever events complete, such as transmission complete (`EX_TXCOMPLETE`). We log join and transmission events to the Arduino Serial Monitor. When the transmission is complete, we schedule the next uplink message via our `do_send` function using the `setTimedCallback` method, the `TX_INTERVAL` constant.

10. Add the following bolded lines to the sketch file:

```
…
            break;
    }
}
void setup() {
    Serial.begin(9600);
    Serial.println(F("Starting"));
    // Initalizes the LIMIC library,
    os_init();
    // Resets the MAC state – this removes sessions, meaning the device must
repeat the join process each time it is started.
    LMIC_reset();
    // Let LMIC compensate for an inaccurate clock
    LMIC_setClockError(MAX_CLOCK_ERROR * 1 / 100);
    // Disable link check validation – this is used to periodically verify
network connectivity. Not needed in this tutorial.
    LMIC_setLinkCheckMode(0);
    // Set data rate to Spreading Factor 7 and transmit power to 14 dBi for
uplinks
    LMIC_setDrTxpow(DR_SF7, 14);
    // Start job
    do_send(&sendjob);
}


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

In the `setup` function, we initialize the LMIC library using `os_init`. We also reset the MAC state to ensure the join process occurs each time the device is started, turn off link-check validation and set the spreading factor and transmit power. We then call `doSend` which will attempt to send the first message and start OTAA automatically.
The `loop` function calls LMIC's `os_runloop_once` method. This hands the loop function over to LMIC and the code then responds to the event **callbacks** returned from LMIC in the `onEvent` method.

11. If you are in the U.S. and using TTN, joining is faster if we select channels 8-15 (subband 1) to broadcast the joining message on.

    If you are not using the U.S. frequency setting in the LMIC library, adding this line will cause an error and, for Helium, you do not need to perform this step. This step is only for U.S.-based TTN users. If you are using a different network server provider, ask them which subband they recommend for join messages.

    Add the following bolded lines to the `setup` function:

```
…

    // Set data rate to Spreading Factor 7 and transmit power to 14 dBi for
uplinks

    LMIC_setDrTxpow(DR_SF7,14);

    // US only, select subband 1 (channels 8-15) for joining TTN

    LMIC_selectSubBand(1);

    // Start job

    do_send(&sendjob);

…
```

12. Plug in your gateway.

13. Sign in to the TTN console and open the **Gateways** page. Check to verify that your gateway is showing as connected. At the time of writing this article there is a known bug in TTN Console which can cause the gateway status to display incorrectly – so do not worry if your gateway is listed as '*not connected.'* Move onto the next steps and you should see the gateway display as connected as soon as your LoRa-based broadcasting node sends a message to the LNS.

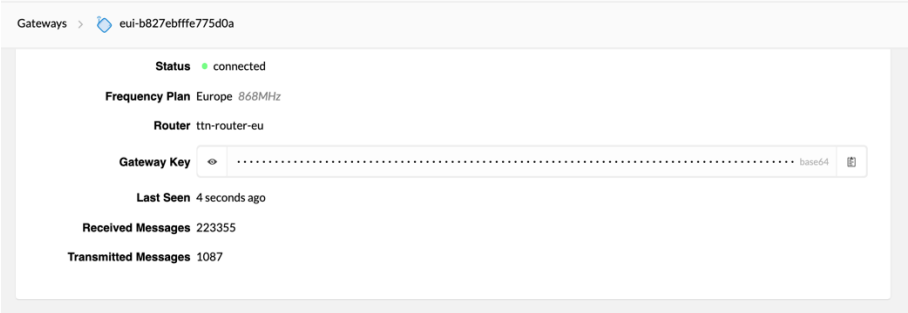


**Figure 35: TTN console showing connected gateway**

14. Connect the broadcasting node Arduino to the computer's USB port using the USB-B cable. We should see a power LED illuminate on the Arduino and on the shield.
15. At the Arduino IDE menu, select **Tools > Port.** We need to select the serial port that the Arduino is connected to. Select the port labeled with the model of your Arduino board. If none of the ports are labelled, disconnect the Arduino board and reopen the menu; the entry which disappears should be your board. Reconnect the board, then select the entry which had disappeared.
16. At the Arduino IDE menu, open **Tools > Serial Monitor** to open the Serial Monitor.
17. Verify and upload the sketch using the buttons in the **Sketch Window** menu bar.
18. We should see the logs showing the EV_JOINING and EV_JOINED messages in the Serial Monitor. The device has now completed the OTAA process, using the Device EUI and Application Key.

    If you only see the EV_JOINING message, double-check that the DEVEUI, APPEUI and APPKEY settings are all correct, as added at the start of this stage.

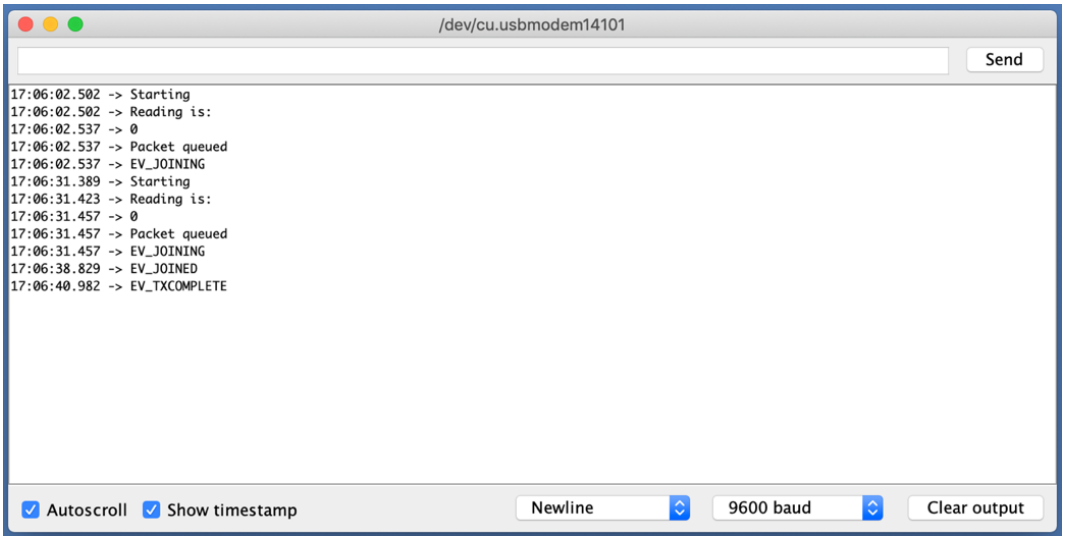


**Figure 36: Serial Monitor showing the join process**

19. Navigate to the TTN console applications page **> select our application > Devices > select our device > Data** tab. We will see the readings from the device.

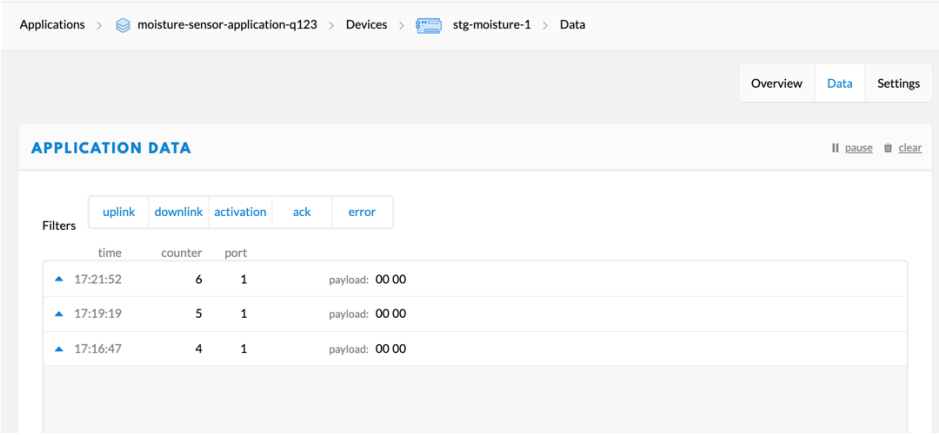


**Figure 37: TTN console showing the application data tab**

20. Next, we'll add a decoder so we can read the payloads more easily. Navigate to the TTN console applications page **> Select our application > Payload Formats** tab.
21. Complete the form as follows:
    a. **Payload Format**: leave set to **custom**
    b. **Decoder** tab: leave selected, paste the bolded lines below into the field below

```
function Decoder(bytes, port) {

  // Decode an uplink message from a buffer

  // (array) of bytes to an object of fields.

  var decoded = {};

  decoded.Moisture = bytes[0] + bytes[1] * 256;

  return decoded;

}
```

    c. **Payload:** test the function by pasting in '61 01' and then select **Test**. We should see a reading in the original numeric format, e.g:

```
{

  "Moisture": 353

}
```

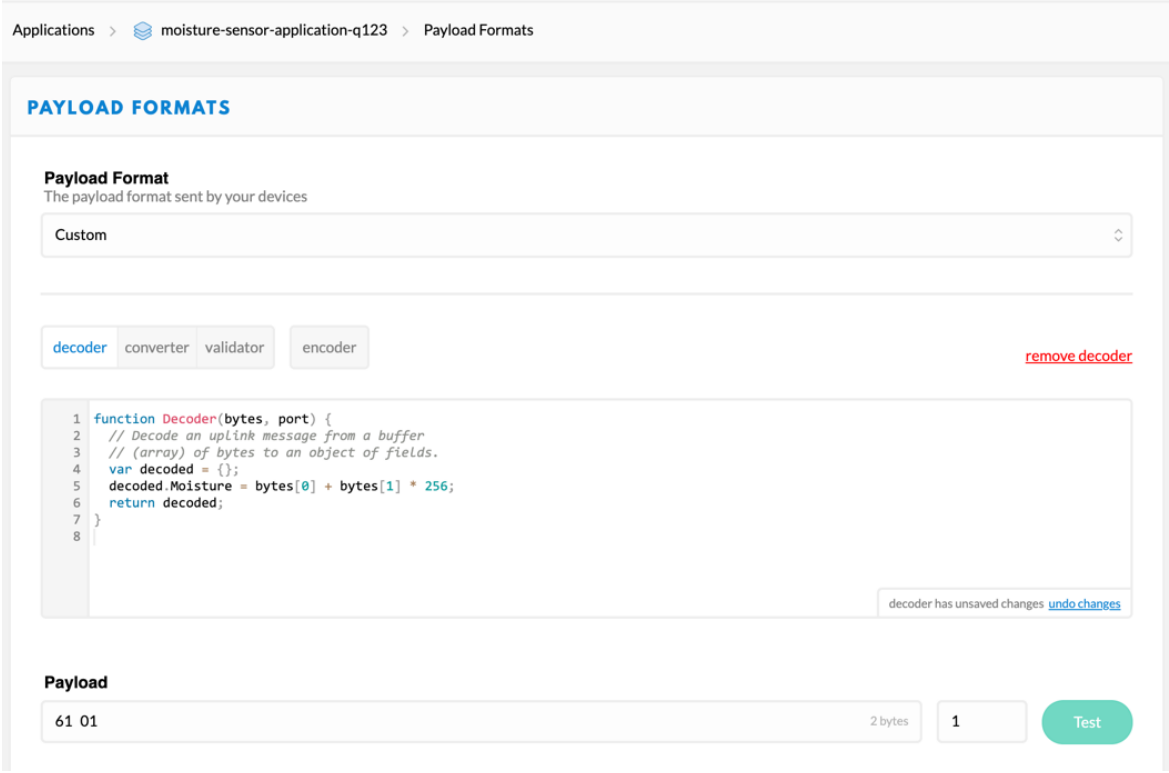


Figure 38: TTN console custom payload format decoder

22. Click the **save payload functions** button at the bottom of the page.
23. Navigate to [the TTN console applications page] **> Select our application > Devices > Select your device > Data** tab. Here we can now see the readings from the device, alongside our translated payload.

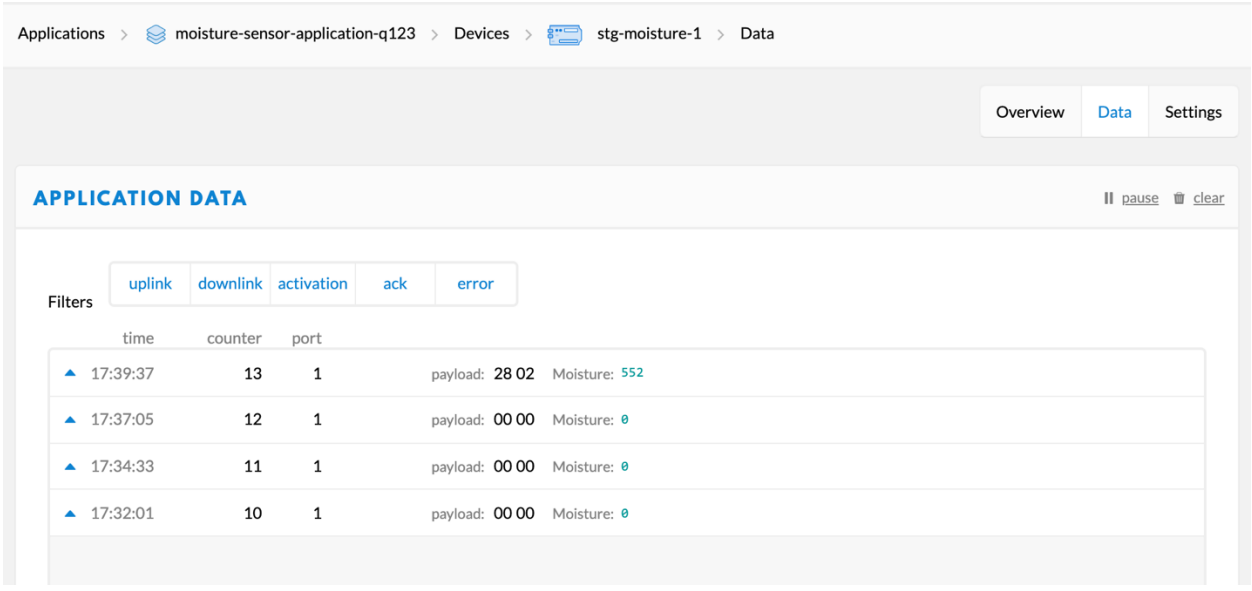


Figure 39: TTN console Application Data log showing moisture readings

## Summary and Next Steps

In this tutorial we have built and configured, end-to-end, a LoRa-based device which broadcasts a reading from an analog soil moisture sensor. The device has been registered with The Things Network and the broadcast message, received by a configured gateway, appears in the TTN console.

You can improve the device you have built by configuring the LoRa-based message to broadcast only if certain threshold criteria are met for the sensor reading. For example, you might want to send a message only if the moisture sensor reading value is above a certain level, or if the value has changed from the previous reading.

You could attach multiple sensors to the same node; for instance, you might want to know the moisture reading as well as the temperature and lighting levels. The node could then be configured to broadcast a payload containing the current readings from all the sensors, or only broadcast if a series of criteria are met, e.g. only broadcast if the moisture level is below a certain threshold and the temperature and light levels are above some other thresholds. Remember to update the application payload formatter so that each reading shows in TTN console data log. Also, remember to recalculate your broadcast delay time if you increase your payload size.

If you are planning to build multiple LoRa-based nodes, the ESP32 microcontroller can be used to build nodes in a very similar way. There are many manufacturers of ESP32 boards, some with LoRa devices built-in. These boards are considerably smaller and cheaper than Arduino boards, but often require some soldering to get started. The boards can be programmed using the same LMIC library and should simply require the pin mapping to be updated. The ESP32 boards also have built-in deep sleep methods, allowing you to easily set sleep time between broadcasts or have the device sleep until an interruption is triggered by one of the sensors. Putting the board to sleep between broadcasts is good practice and highly recommended for extending the battery life of the LoRa-based node.

Now that you have a LoRa-based node broadcasting messages to a third-party network server, you can design and develop your own dashboard or app and use the TTN API to fetch your device data and represent it in any way you wish.

## Important Notice

Information relating to this product and the application or design described herein is believed to be reliable, however such information is provided as a guide only and Semtech assumes no liability for any errors in this document, or for the application or design described herein. Semtech reserves the right to make changes to the product or this document at any time without notice. Buyers should obtain the latest relevant information before placing order and should verify that such information is current and complete. Semtech warrants performance of its products to the specifications applicable at the time of sale, and all sales are made in accordance with Semtech's standard terms and conditions of sale.

SEMTECH PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS, OR IN NUCLEAR APPLICATIONS IN WHICH THE FAILURE COULD BE REASONABLY EXPECTED TO RESULT IN PERSONAL INJURY, LOSS OF LIFE OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. INCLUSION OF SEMTECH PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE UNDERTAKEN SOLELY AT THE CUSTOMER'S OWN RISK. Should a customer purchase or use Semtech products for any such unauthorized application, the consumer shall indemnify and hold Semtech and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages and attorney fees which could arise.



## Contact Information

Semtech Corporation
200 Flynn Road, Camarillo, CA 93012
Phone: (805) 498-2111, Fax: (805) 498-3804
www.semtech.com