



Single Channel LoRa IoT Kit v2 User Manual

Document Version: 1.0.7

Version	Description	Date
1.0.0	Release	2019-Jan-28
1.0.1	Modify limitation words	2019-May-10
1.0.2	Add description of MQTT publish format	2019-Jun-15
1.0.3	Add ThingSpeak downlink example, and Video Link	2019-Jun-19
1.0.4	Correct broken links.	2019-Jun-25
1.0.5	Add trouble shooting for Arduino 1.8.10, add CN470 support, Change Cayenne to Mydevices	2019-Oct-25
1.0.6	Add Password	2020-Nov-24
1.0.7	Instructions for changing TTnv2 to TTnv3	2021-Aug-04

Index:

1	Overview	4
1.1	What is Dragino Single Channel LoRa IoT Kit v2?	4
1.2	What can you learn from the kit?.....	4
1.3	What parts Dragino LoRa IoT v2 includes?.....	5
2	Preparing.....	6
2.1	Software for End Node	6
2.1.1	Install Arduino IDE and CH340 driver.....	6
2.1.2	Install LoRa Library for Arduino.....	7
2.2	Prepare for LG01-N Gateway.....	8
2.2.1	Configure LG01-N for internet connection.	8
2.2.2	Download putty tool to access LG01-N via SSH	11
3	Example 1: Test a LoRaWAN network	12
3.1	Typology and Data Flow	13
3.2	Create a gateway in TTNV3 Server	14
3.3	Configure LG01-N Gateway	17
3.3.1	Configure to connect to LoRaWAN server	17
3.3.2	Configure LG01-N's LoRa Radio frequency	18
3.4	Create LoRa Shield End Node	19
3.4.1	Hardware Connection	19
3.4.2	Set up OTAA device in TTNV3 and upload sketch to UNO	19
3.4.3	Configure to connect to Mydevices Application Server.....	24
3.4.4	Use downlink message to control relay	28
3.4.5	Test with Interrupt	31
3.5	Create LoRa/GPS Shield End Node	32
3.5.1	Hardware connection.....	32
3.5.2	Set up ABP device in TTNV3 and upload software to UNO	32
3.6	Conclusion and limitation.....	36
3.6.1	Overview for the example.....	37
3.6.2	Limitations.....	38
4	Example 2: Test with a MQTT IoT Server	40
4.1	Typology and Data Flow	40
4.2	Set up sensor channels in ThingSpeak	41
4.3	Simulate MQTT uplink via PC's MQTT tool.....	43
4.4	Try MQTT Publish with LG01-N Linux command	44
4.5	Configure LG01-N Gateway	46
4.5.1	Publish Logic.....	46
4.5.2	Configure LG01-N's Radio frequency	47
4.6	Create LoRa Shield End Node	49
4.6.1	Hardware Connection	49

4.6.2	Test with uplink	50
4.6.3	Test with interrupt by flame detect	51
4.6.4	Test with downlink	53
4.7	Conclusion and limitation	55
4.7.1	Overview for the example.....	55
5	Order Info.....	56
6	FAQ & Trouble Shooting.....	57
6.1	I can't upload sketch to LoRa Shield in MAC OS, shows "dev/cu.usbmodem1421 is not available "	57
6.2	My IoT Kit has the model LG01-P instead of LG01-N, Can I still use this manual.	57
6.3	Duplicate library issue while upload in Arduino IDE 1.8.10.	58
6.4	How can I set to use CN470 band?.....	59
7	Technical Support.....	60
8	Reference	61

1 Overview

1.1 What is Dragino Single Channel LoRa IoT Kit v2?

Dragino Single Channel LoRa IoT Kit v2 is designed to facilitate beginners and developers to quickly learn LoRa and IoT technology. It helps users to turn the idea into a practical application and make the Internet of Things a reality. It is easy to program, create and connect your things everywhere. A number of telecom operators are currently rolling out networks, but because LoRa operates in the open spectrum you can also set up your own network.

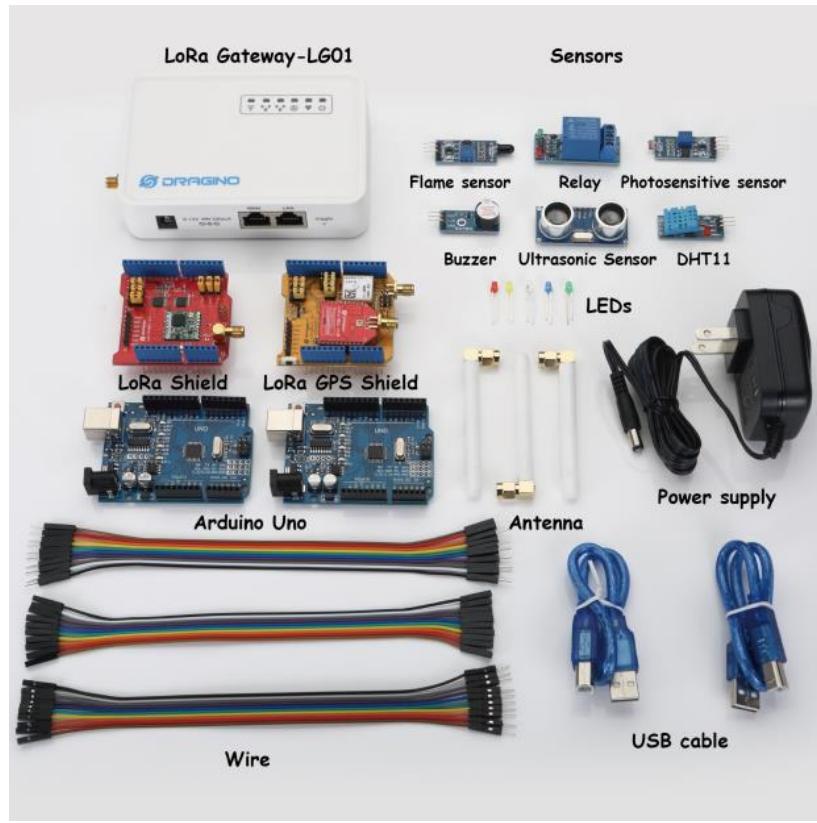
The LoRa IoT kit v2 shows how to build a LoRa network, and how to use the network to send data from a LoRa sensor node to the cloud server. Depends on the actually use environment, the LoRa gateway will connect your other LoRa nodes up to 500 ~ 5,000 meters.

1.2 What can you learn from the kit?

The goals through this LoRa IoT kit v2:

- ✓ Understand the structure of an Internet of Things network, and how does an IoT network works
- ✓ Learn coding method for Arduino micro controller
- ✓ Learn some common sensors.
- ✓ Learn some basic commands for Linux and
- ✓ Learn about LoRa and how to set up a LoRa network.
- ✓ Learn different way to connect LoRa network to IoT Server and compare their advantages / disadvantages.

1.3 What parts Dragino LoRa IoT v2 includes?



Single Channel LoRa IoT Kit Packing List.

- ✓ 1 x [LG01-N](#) single channel LoRa Gateway
- ✓ 1 x LoRa end node ([LoRa Shield](#) + Arduino UNO)
- ✓ 1 x LoRa end node ([LoRa/GPS Shield](#) + Arduino UNO)
- ✓ 1 x flame Sensor
- ✓ 1 x relay
- ✓ 1 x photosensitive sensor
- ✓ 1 x buzzer
- ✓ 1 x ultrasonic sensor
- ✓ 1 x DHT11 temperature and humidity sensor
- ✓ 20 x dupont cable (male to male)
- ✓ 20 x dupont cable (female to female)
- ✓ 20 x dupont cable (male to female)

2 Preparing

In the kit, there are two LoRa End Node, they are LoRa Shield + UNO and LoRa/GPS Shield + UNO. Both of them use Arduino UNO as MCU to control the LoRa transceiver.

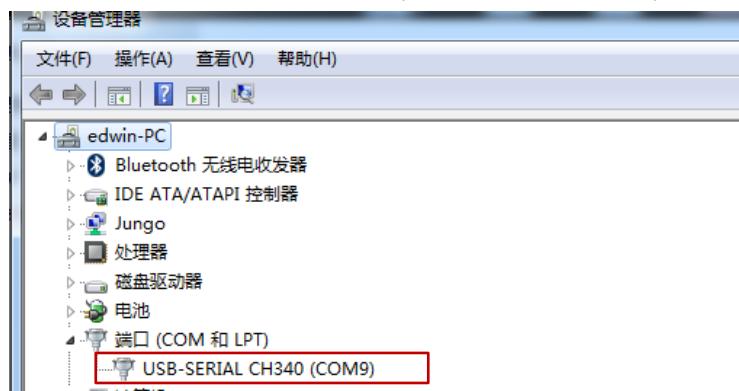
We need to program the Arduino UNO during our testing to support the required functions for end nodes. To finish this, we need to install some software and library first.

2.1 Software for End Node

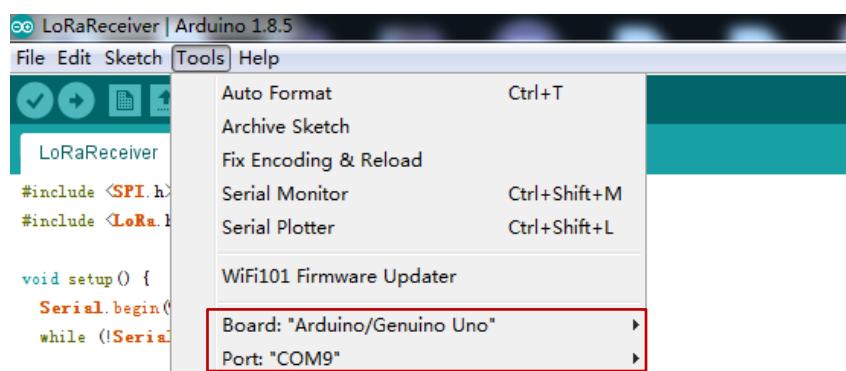
2.1.1 Install Arduino IDE and CH340 driver

First download and install [Arduino IDE](#). This is the tool to program the Arduino UNO.

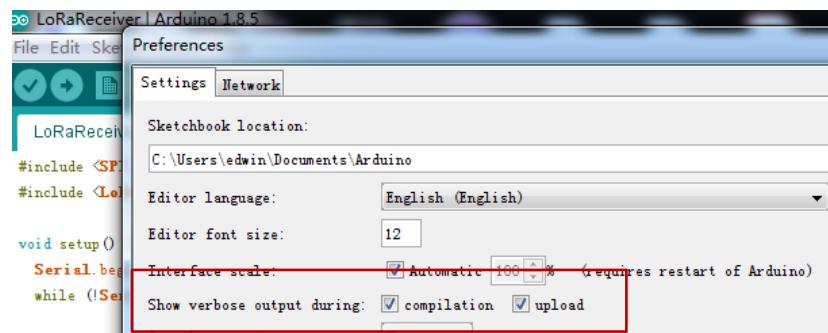
The Arduino UNO in the kit is clone version and is equipped with CH340 USB to UART chip. We need to install CH340 driver in the PC to let the Arduino IDE program it via USB. If we successful install the driver, a com port will show in the system device manager:



After install the driver, start Arduino and we will be able to use the board Arduino UNO and corresponding COM port to program UNO now.



We can enable compilation and upload in Arduino → File → Preference. This will help us in debug.



2.1.2 Install LoRa Library for Arduino

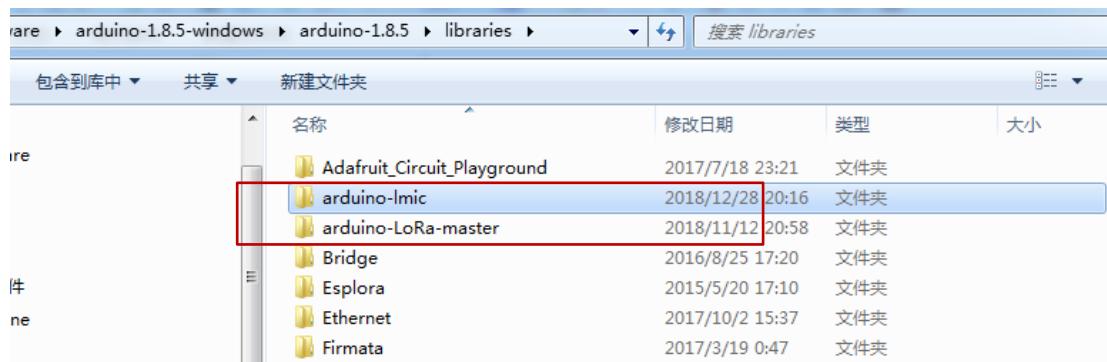
In our examples, we will use two different LoRa libraries for End Node to build different type of LoRa network. They are:

- [Arduino-LMIC](#): LoRaWAN library to configure the End node as a standard LoRaWAN end node.
- [LoRa-raw](#): This is a simple library for LoRa transmit & receive, all data transfer without ID control, encryption. If user wants to develop a LoRa network with private LoRa protocol, he can modify base on this Library.

We also need to install some libraries to connect to different sensors:

- [DHTlib](#): This is the library to use DHT11 temperature & humidity sensor.
- [TinyGPS](#): Library for LoRa GPS Shield to get the GPS data.

Download all above libraries and put them in the [Arduino → Libraries](#) directory



2.2 Prepare for LG01-N Gateway

In LoRa IoT Kit v2, we use LG01-N as LoRa Gateway. Unlike LG01-P in v1 kit, the LG01-N has its own LoRa utility and not need to program it via Arduino. Since we need to connect to Internet IoT Server, we need to configure the LG01-N to have internet access.

2.2.1 Configure LG01-N for internet connection.

Below steps show how to set up LG01-N to use WiFi for internet access.

Step1:

Connect PC to LG01-N's LAN port via RJ45 cable and set up PC Ethernet port to DHCP.

PC will then get IP from LG01-N. The ip range is 10.130.1.xx

Use browser to access the LG01-N via IP 10.130.1.1. (Recommend use Chrome here)

User can also connect to the wifi AP network from LG01-N, the **wifi password** is **dragino+dragino**.

Step2:

Open a browser in the laptop and type

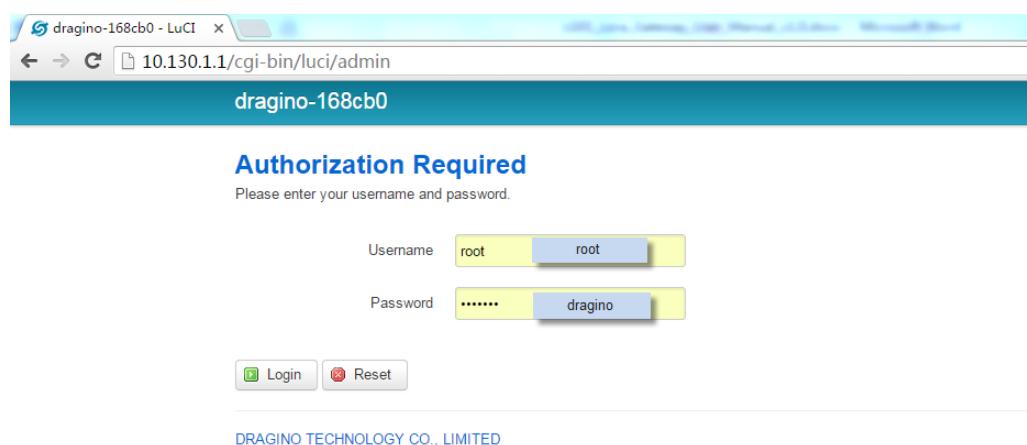
<http://10.130.1.1/cgi-bin/luci/admin>

User will see the login interface of LG01-N.

The account for Web Login is:

User Name: root

Password: dragino



Step3:

In network -> Wireless, select radio0 interface and scan.

radio0: Master "dragino-1b8288"

Wireless Overview


Generic MAC80211 802.11bgn
Channel: 11 (2.462 GHz) | Bitrate: ? Mbit/s

Restart

Scan

Add


SSID: dragino-1b8288 | Mode: Master
BSSID: A8:40:41:1B:82:88 | Encryption: None

Disable

Edit
Remove

Step4:

Select the wireless AP and join the wifi network:

Join Network: Wireless Scan

Signal	SSID	Channel	Mode	BSSID	Encryption	
100%	dragino-office	8	Master	50:64:2B:1A:B8:4D	mixed WPA/WPA2 - PSK	Join Network
84%	ChinaNet-gLnb	2	Master	A4:29:40:66:F4:E7	mixed WPA/WPA2 - PSK	Join Network

Joining Network: "dragino-office"

Replace wireless configuration
 Check this option to delete the existing networks from this radio.

WPA passphrase

*

 Specify the secret encryption key here.

Name of the new network

 The allowed characters are: A-Z, a-z, 0-9 and -

Create / Assign firewall-zone

 Choose the firewall zone you want to assign to this interface. Select *unspecified* to remove the interface from the associated zone or fill out the *create* field to define a new zone and attach the interface to it.

Back to scan results
Submit

Step5:

In network->wireless page, disable WiFi AP network. Notice: After doing that, you will lose connection if your computer connects to the LG01-N via its WiFi network.

radio0: Master "dragino-1b8288"

Wireless Overview

 radio0	Generic MAC80211 802.11bgn Channel: 11 (2.462 GHz) Bitrate: ? Mbit/s	<button>Restart</button> <button>Scan</button> <button>Add</button>
 0%	SSID: dragino-1b8288 Mode: Master BSSID: A8:40:41:1B:82:88 Encryption: None	<button>Disable</button> <button>Edit</button> <button>Remove</button>
 0%	SSID: dragino-office Mode: Client BSSID: 50:64:2B:1A:B8:4D Encryption: -	<button>Disable</button> <button>Edit</button> <button>Remove</button>

Associated Stations

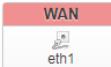
Network	MAC-Address	Host	Signal / Noise	RX Rate / TX Rate
No information available				

(Note: make sure click the Save & Apply after configure)

After successful associate, the WiFi network interface can be seen in the same page and see LG01-N get the ip from the uplink router.

[WAN](#) [WWAN](#) [LAN](#)

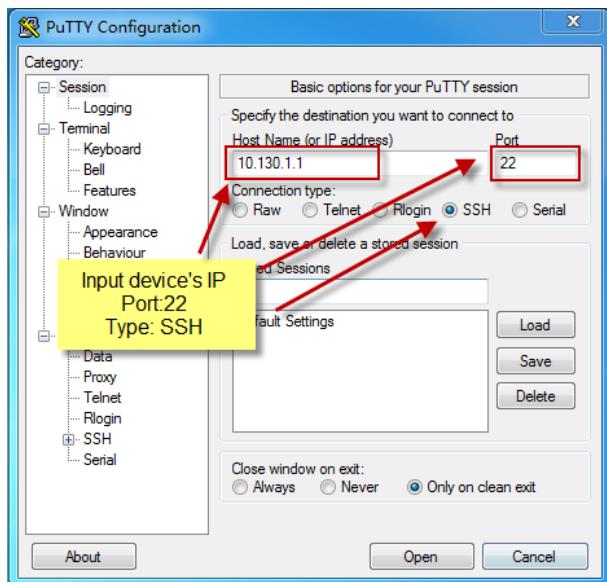
Interfaces

 LAN br-lan	Protocol: Static address Uptime: 2h 0m 4s MAC: A8:40:41:1B:82:8B RX: 1.40 MB (13346 Pkts.) TX: 2.79 MB (10321 Pkts.) IPv4: 10.130.1.1/24	<button>Restart</button> <button>Stop</button> <button>Edit</button> <button>Delete</button>
 WAN eth1	Protocol: DHCP client MAC: A8:40:41:1B:82:8A RX: 4.30 MB (51840 Pkts.) TX: 55.77 KB (429 Pkts.)	<button>Restart</button> <button>Stop</button> <button>Edit</button> <button>Delete</button>
 WWAN Client "dragino-office"	Protocol: DHCP client Uptime: 0h 6m 6s MAC: A8:40:41:1B:82:88 RX: 549.38 KB (5659 Pkts.) TX: 14.90 KB (94 Pkts.) IPv4: 10.130.2.169/24	<button>Restart</button> <button>Stop</button> <button>Edit</button> <button>Delete</button>
Add new interface...		
Save & Apply Save Reset		

2.2.2 Download putty tool to access LG01-N via SSH

It will be helpful to see the LG01-N inside Linux system to understand the data flow and debug.

User can access to the Linux console via SSH protocol. Make sure your PC and the LG01-N is in the same network, then use a SSH tool (such as [putty](#)) to access it. Below are screenshots:



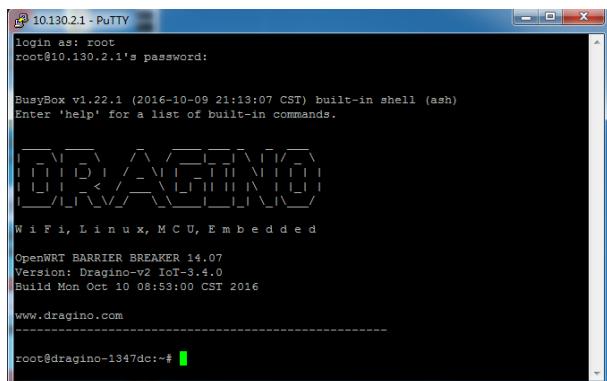
IP address: IP address of LG01-N

Port: 22

User Name: root

Password: dragino (default)

After log in, you will be in the Linux console and can input commands here.



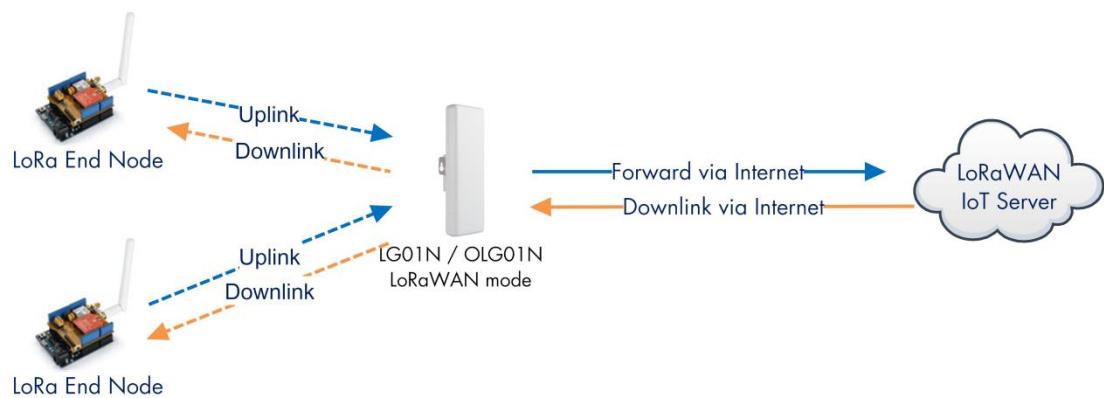
So we have prepare what we need and let's go for the examples!

3 Example 1: Test a LoRaWAN network

This example describes how to use LG01-N, LoRa Shield & LoRa GPS Shield to set up a LoRaWAN network and connect it to [TTNv3 LoRaWAN Server](#). It also shows how to use external application server to monitor / manage the LoRa Nodes.

LoRaWAN mode:

Use LG01N / OLG01N as a LoRaWAN gateway* to forward packet to LoRaWAN IoT Server

**Operate Principle:**

- LG01N/OLG01N running packet forward and will forward the uplink LoRa packet from end node to LoRaWAN server.
- It will also forward downlink LoRa packet from LoRaWAN server to end node.
- The end node can use OTAA or ABP mode in the LoRaWAN protocol.

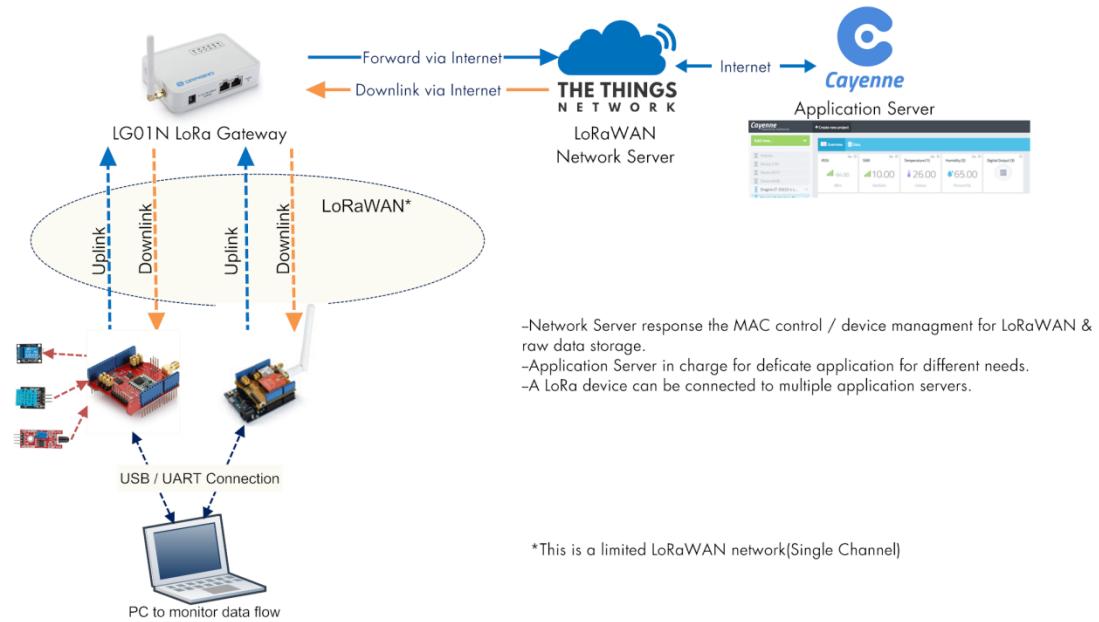
Limitation:

- The LG01 only support one LoRaWAN frequency for uplink. So the end node should be set to fix frequency.
- If end node use multiply frequencies to transfer, The LG01 will only be able to receive the same frequency set in LG01N.

3.1 Typology and Data Flow

The network topology and dataflow for the example is as below:

Topology for Thethingsnetwork Connection:

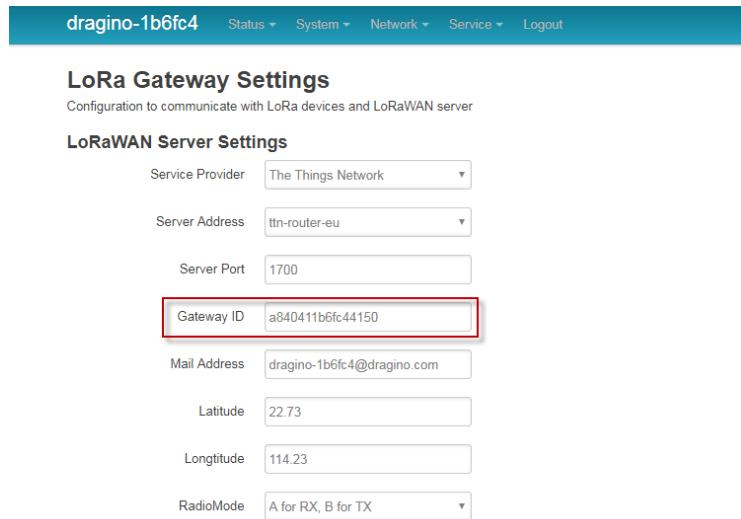


In next section we will start to configure for this example.

3.2 Create a gateway in TTNv3 Server

Step 1: Get a Unique gateway ID.

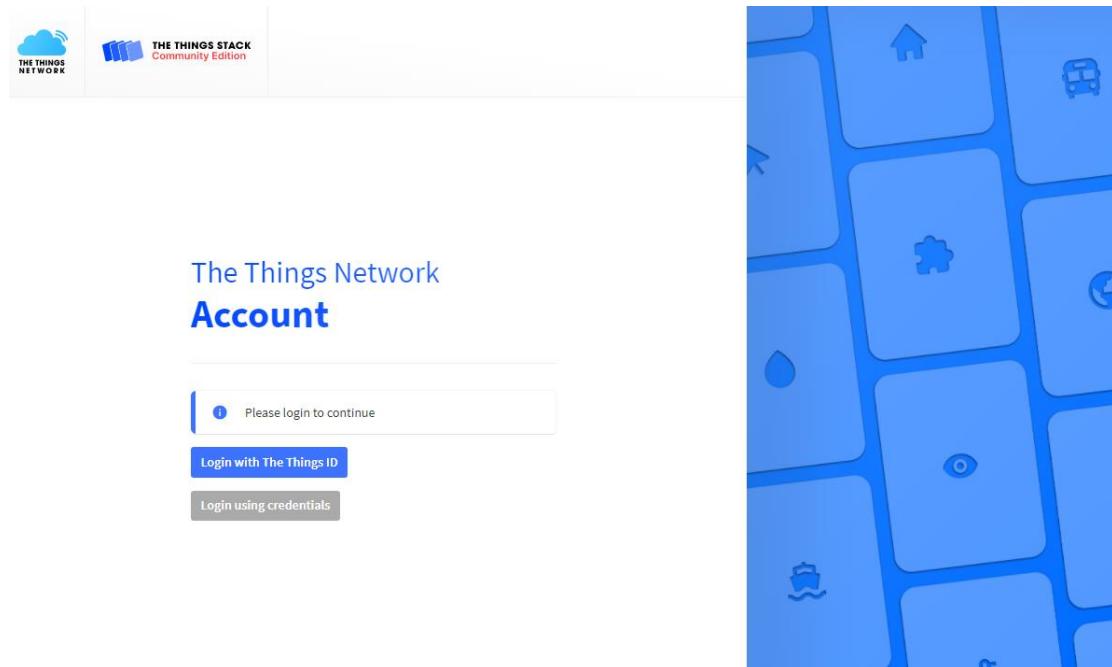
Every LG01-N has a unique gateway id. The id can be found at LoRaWAN page:



The screenshot shows the 'LoRa Gateway Settings' configuration page. It includes fields for Service Provider (The Things Network), Server Address (ttn-router-eu), Server Port (1700), and Gateway ID (a840411b6fc44150, which is highlighted with a red box). Other fields include Mail Address (dragino-1b6fc4@dragino.com), Latitude (22.73), Longitude (114.23), and RadioMode (A for RX, B for TX).

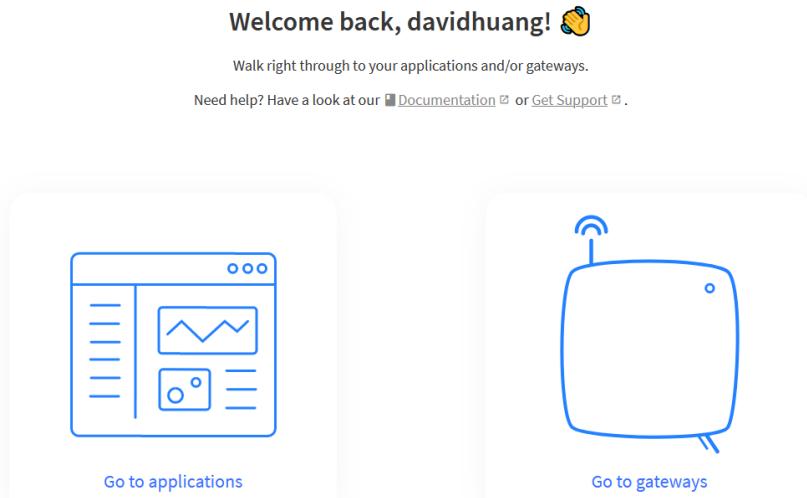
The gateway id is: **a840411b6fc44150**

Step 2: Sign up a user account in TTNv3 server



The screenshot shows the 'The Things Network Account' sign-up page. It features two main sections: a left sidebar with 'THE THINGS NETWORK' and 'THE THINGS STACK Community Edition' icons, and a right panel with a 'Please login to continue' message, 'Login with The Things ID' button, and 'Login using credentials' link. To the right of the login form is a map of a city with blue tiles representing coverage areas and icons for various IoT devices like a bus, a person, and a water drop.

Step 3: Create a Gateway in TTNv3



Add gateway

General settings

Owner*
davidhuang

Gateway ID*
a84041168f24ff00

Gateway EUI
A8 40 41 16 8F 24 FF 00

Put the Gateway ID here

Gateway name
My new gateway

Gateway description
LG01-N

Optional gateway description; can also be used to save notes about the gateway

Gateway Server address
eu1.cloud.thethings.network

Gateway status ⓘ

Public
The status of this gateway may be visible to other users

Gateway location ⓘ

Public
The location of this gateway may be visible to other users and on public gateway maps

Attributes ⓘ

[+ Add attributes](#)

Attributes can be used to set arbitrary information about the entity, to be used by scripts, or simply for your own organization

LoRaWAN options

Frequency plan ⓘ

Europe 863-870 MHz (SF9 for RX2 - recommended) 

Schedule downlink late ⓘ

Enabled
Enable server-side buffer of downlink messages

Enforce duty cycle ⓘ

Enabled
Recommended for all gateways in order to respect spectrum regulations

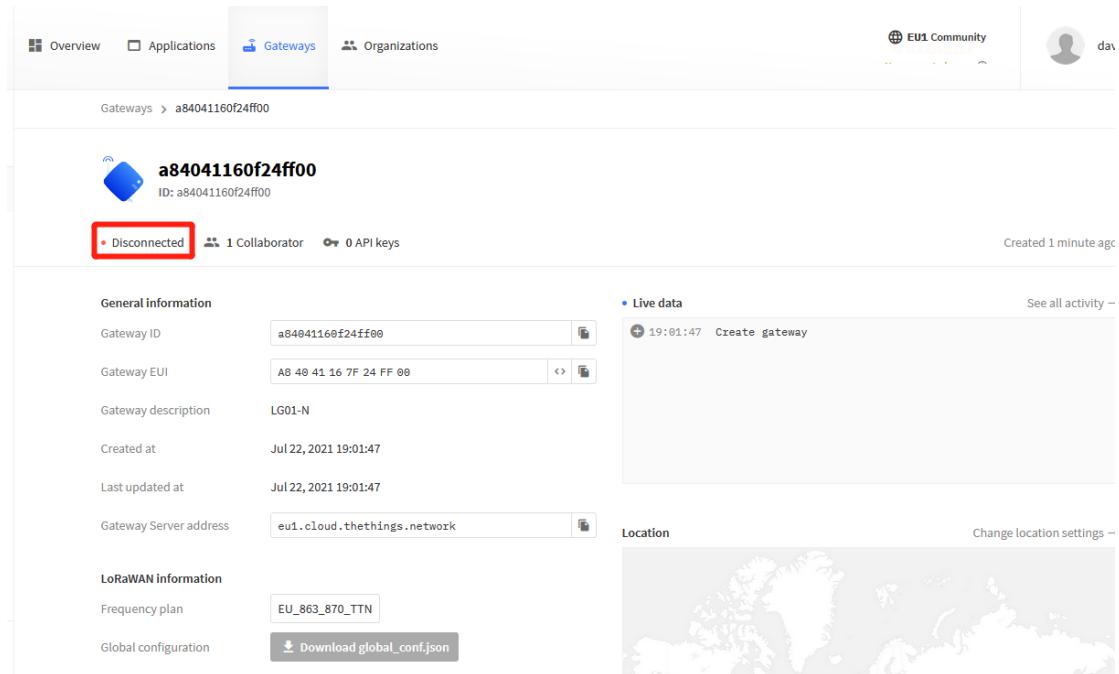
Schedule any time delay ⓘ *

530 milliseconds 

Configure gateway delay (minimum: 130ms, default: 530ms)

Choose the right frequency plan

After create the gateway, we can see the gateway info, as below, the **Status** shows “not connected” because the LG01-N doesn’t configure to send update status yet.



The screenshot shows the Dragino cloud interface for managing gateways. The top navigation bar includes 'Overview', 'Applications', 'Gateways' (selected), and 'Organizations'. On the right, there's a user profile for 'dav' and a link to 'EU1 Community'. The main content area displays a gateway with the ID 'a84041160f24ff00'. The status is shown as 'Disconnected' (highlighted with a red box). Other details include '1 Collaborator' and '0 API keys'. The gateway was 'Created 1 minute ago'. The 'General information' section lists the Gateway ID ('a84041160f24ff00'), Gateway EUI ('A8 40 41 16 7F 24 FF 00'), Gateway description ('LG01-N'), Creation date ('Jul 22, 2021 19:01:47'), and Last update date ('Jul 22, 2021 19:01:47'). The 'LoRaWAN information' section shows the Frequency plan ('EU_863_870_TTN') and a 'Download global_conf.json' button. The 'Live data' section shows a single event: 'Create gateway' at '19:01:47'. The 'Location' section shows a map of Europe with a placeholder for the gateway's location.

3.3 Configure LG01-N Gateway

3.3.1 Configure to connect to LoRaWAN server

We should configure the LG01-N now to let it connect to TTNV3 network. Make sure your LG01-N has Internet Connection first.

Step1: Configure LG01-N to act as LoRaWAN forwarder mode



Single Channel LoRa Gateway

Configuration to communicate with LoRa devices and LoRaWAN server

LoRaWAN Server Settings

IoT Service	LoRaWan/Raw forwarder
Debug Level	Little message output

Step2: Input server info and gateway id

Choose the correct the server address and gateway ID.



LoRa Gateway Settings

Configuration to communicate with LoRa devices and LoRaWAN server

LoRaWAN Server Settings

Service Provider	The Things Network
Server Address	ttn-router-eu
Server Port	1700
Gateway ID	a840411b
Mail Address	edwin@dragino.com
Latitude	22.73
Longitude	114.23

Check Result

After above settings, the LG01-N will be able to connect to TTNV3, as shown in below:

Gateways > LG01N

LG01N
ID: a840-[REDACTED]0

Last seen 28 seconds ago ↑ 1 ↓ 0 1 Collaborator 0 API keys Created 22 hours ago

General information		Live data	
Gateway ID	a840-[REDACTED]f00	See all activity →	
Gateway EUI	A8 40 41 16 8F 24 FF 00		
Gateway description	None		
Created at	Jul 21, 2021 20:35:56		
Last updated at	Jul 21, 2021 20:35:56		
Gateway Server address	eu1.cloud.thethings.network		
LoRaWAN information		Location	
Frequency plan	EU_863_870_TTN	Change location settings →	
Global configuration	Download global_conf.json		

3.3.2 Configure LG01-N's LoRa Radio frequency

Now we should configure LG01-N's radio parameter to receive the LoRaWAN packets. We are using 868.1Mhz and other parameters as below:

Radio Settings

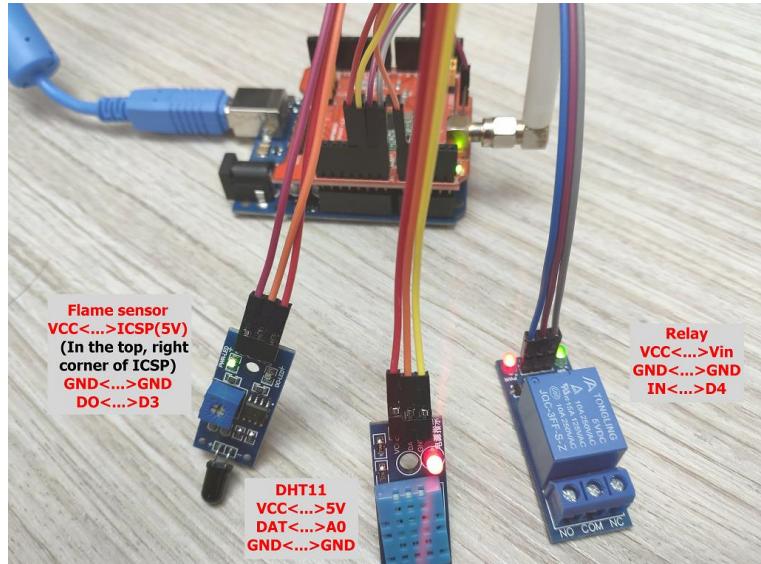
Radio settings for Channel

Frequency (Unit:Hz)	868100000
Spreading Factor	SF7
Coding Rate	4/5
Signal Bandwidth	125 kHz
Preamble Length	8
LoRa Sync Word	52
Encryption Key	Encryption Key

This parameters set is for uplink (receive data for LoRa End Node).According to LoRaWAN spec, the downlink radio parameters frequency is defined by network server (TTNV3). LG01-N will adjust downlink parameters according to info from TTNV3.

3.4 Create LoRa Shield End Node

3.4.1 Hardware Connection



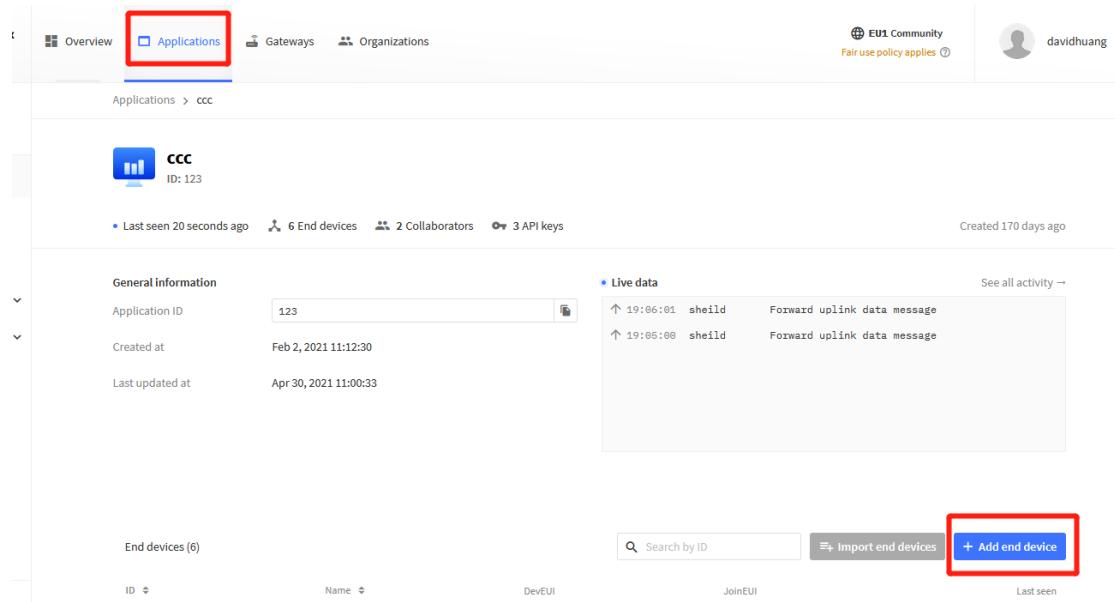
There are three sensors connect to the LoRa Shield + UNO. These sensors are flame sensors, DHT11 (Temperature & Humidity sensor) and Relay. Please use the connection as we show in the photo.

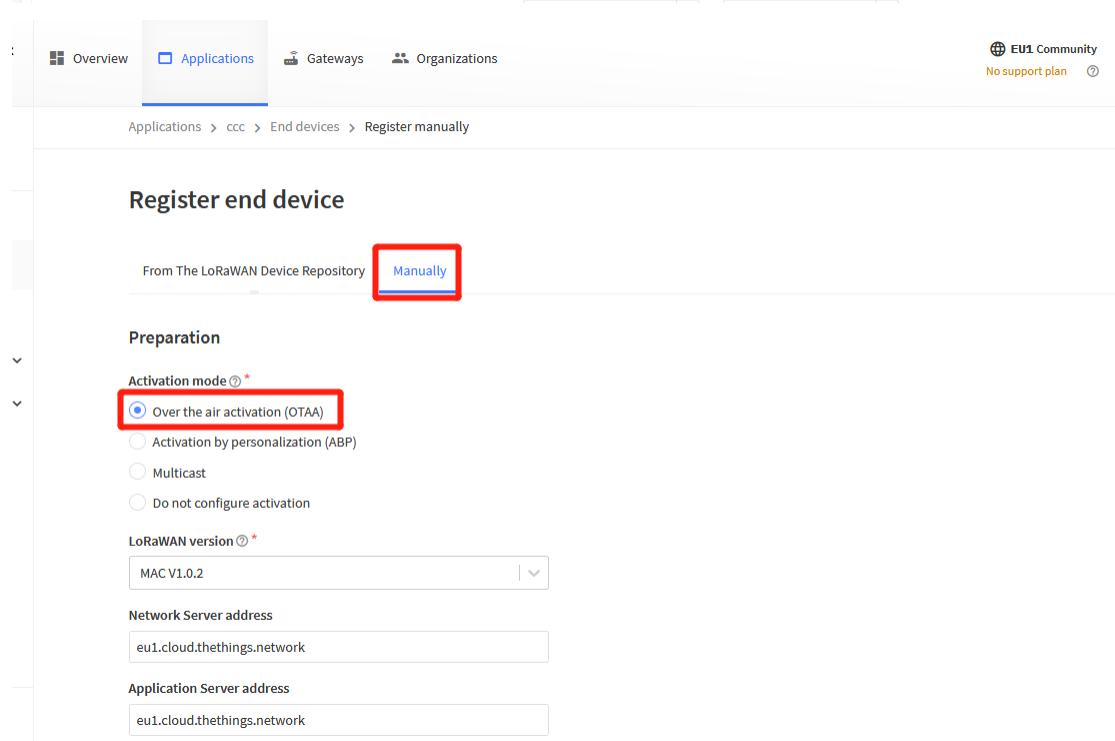
Note: There is a trick in above connection, the relay connects to VIN. In this case, The UNO can only be power via USB port. If user need to power via DC power adapter, please use another 5v pin to power the relay.

3.4.2 Set up OTAA device in TTNV3 and upload sketch to UNO

Here we set up the LoRa Shield + UNO as an OTAA device in TTNV3. We will tell the difference of OTAA and ABP mode later.

[Step 1](#): Create an OTAA device in TTNV3 server -- > Application page.





For this device, set up to use Cayenne payload, so TTNV3 can parse the sensor data properly.

Step 2: Modify the LMIC library

To use LoRaWAN with LG01-N, we need to modify the LMIC library to support single channel mode.

Find the [Arduino LMIC](#) install path in Arduino library. Before compiling the code, user needs to change the Frequency Band to use with LG01-N. The change is in the file **arduino\libraries\arduino-lmic\src\lmic\config.h**. Changes are as below:

```
#define CFG_eu868 1
//#define CFG_us915 1
//#define CFG_au921 1
//#define CFG_as923 1
//#define CFG_in866 1

#define LG02_LG01 1

//US915: DR_SF10=0, DR_SF9=1, DR_SF8=2, DR_SF7=3, DR_SF8C=4
//          DR_SF12CR=8, DR_SF11CR=9, DR_SF10CR=10, DR_SF9CR=11, DR_SF8CR=12, DR_SF7CR
#ifndef defined(CFG_us915) && defined(LG02_LG01)
  #define LG02_UPFREQ End Device Uplink Frequency
  #define LG02_DNWRFREQ End Device Uplink Frequency
  #define LG02_RXSF End Device Uplink (transmit) SF
  #define LG02_TXSF End Device Downlink (receive) SF
  #define LG02_RXSF 3      // DR_SF7
  #define LG02_TXSF 8      // DR_SF12CR
  #elif defined(CFG_eu868) && defined(LG02_LG01)
    #define LG02_UPFREQ 902320000
    #define LG02_DNWRFREQ 923300000
    #define LG02_RXSF 5      // DR_SF7
    #define LG02_TXSF 0      // DR_SF12
#endif
```

Choose the Frequency Band, same as in LoRaWAN server

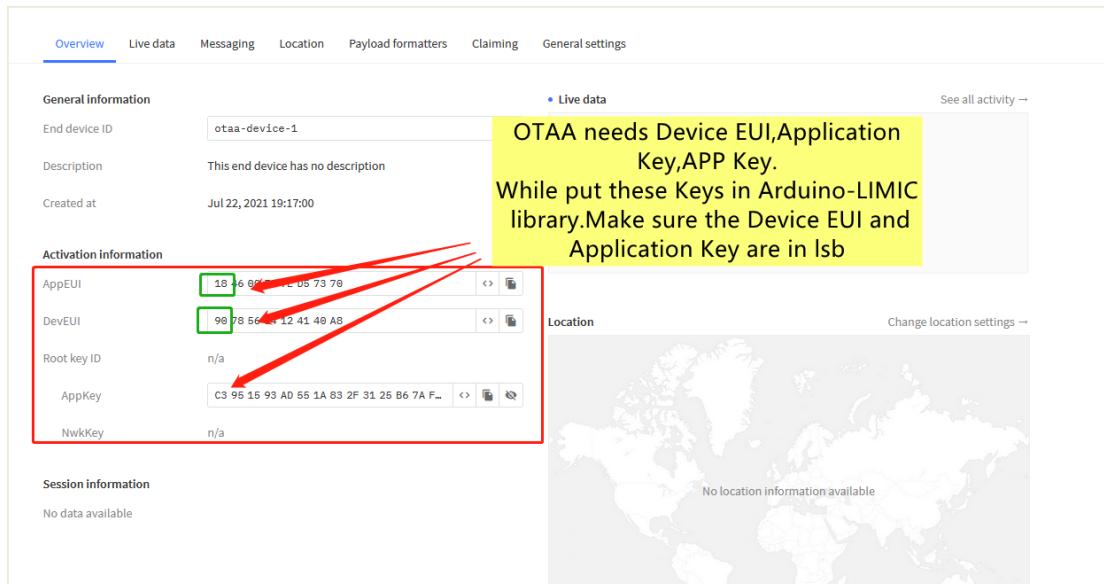
uncomment this for LG01 / LG02

The TXSF is now set to default value:
US915/AS923 : 923300000 , SF12BW500
EU868: 869525000, SF12BW125

Step 3: Input keys in Arduino Sketch and upload to device.

The sketch for this example is [lora_shield_cayenne_and_TTNv3-otaaClient.ino](#). Download and open it, we need to modify the keys to match the keys in TTNv3. Get Device EUI/Application EUI

& APP Key from TTNV3 and put them in the sketch, make sure the Device EUI and Application Key are lsb and the APP key is msb.



General information

End device ID: otaa-device-1

Description: This end device has no description

Created at: Jul 22, 2021 19:17:00

Activation information

AppEUI	18 46 07 2C 09 73 70
DevEUI	98 78 56 04 12 41 40 A8
Root key ID	n/a
AppKey	C3 95 15 93 AD 55 1A 83 2F 31 25 B6 7A F...
NwkKey	n/a

Session information

No data available

Location

See all activity →

OTAA needs Device EUI, Application Key, APP Key.
While put these Keys in Arduino-LIMIC library. Make sure the Device EUI and Application Key are in lsb

Change location settings →

No location information available

ttn-otaa

```
#include <SPI.h>

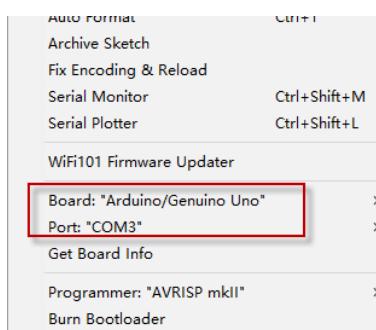
// This EUI must be in little-endian format, so least-significant-byte
// first. When copying an EUI from ttncctl output, this means to reverse
// the bytes. For TIN issued EUIs the last bytes should be 0xD5, 0xB3,
// 0x70.
static const ui_t PROGMEM APPEUI[8]={ 0x18, 0x46, 0x00, 0xF0, 0x7E, 0xD5, 0xB3, 0x70 };
void os_getArtEui (ui_t* buf) { memcpy_P(buf, APPEUI, 8);}

// This should also be in little endian format, see above.
static const ui_t PROGMEM DEVEUI[8]={ 0x90, 0x78, 0x56, 0x34, 0x12, 0x41, 0x40, 0xA8 };
void os_getDevEui (ui_t* buf) { memcpy_P(buf, DEVEUI, 8);}

// This key should be in big endian format (or, since it is not really a
// number but a block of memory, endianness does not really apply). In
// practice, a key taken from ttncctl can be copied as-is.
// The key shown here is the semtech default key.
static const ui_t PROGMEM APPKEY[16] = { 0xC3, 0x95, 0x15, 0x93, 0xAD, 0x55, 0x1A, 0x83, 0x2F, 0x31, 0x25, 0xB6, 0x7A, 0xF5, 0x74, 0x1D };
void os_getDevKey (ui_t* buf) { memcpy_P(buf, APPKEY, 16);}
```

Input Keys in Arduino Sketch

Upload the code to UNO:



Step 4: Analyze output result

From output of LoRa Node Serial Monitor, we can see it send Joining after start(TX), then get join ACK (RX), then upload the data to TTNV3 (TX).

The screenshot shows two windows side-by-side. On the left is the TTN-UI interface with a header "ttn-esp | Arduino 1.8.5". It has tabs for Overview, Applications, Gateways, and Organizations. The main area displays a log message: "Connect to TTN and Send data to mydevice cayenne(Use DHT11 Sensor): RXMODE_RSSI". Below this, there is a red box highlighting the text "The temperature and humidity: [28.00°C, 63.00%]". On the right is a window titled "COM12 (Arduino/Genuine Uno)". It shows a continuous stream of log messages from the Arduino, including "The temperature and humidity: [28.00°C, 63.00%]" which corresponds to the highlighted text in the TTN-UI log.

From gateway logread, we can see the data send from end node (txpk), dats get from server(rxpk).

This screenshot shows the "dragino-1b6fb0" device page. At the top, there are tabs for Status, System, Network, Service, and Logout. Below that is a "Logread" section with tabs for FreqINFO, Report, RxTxJson, and ErrorMSG. A yellow box highlights the "Report" tab, which contains the log output. The log is filled with numerous entries related to LoRa traffic, with many lines starting with "TXPK:" or "RXPK:". Several lines are highlighted with red boxes, showing examples of TXPK (transmit) and RXPK (receive) messages, including their timestamps, frequencies, and payload details.

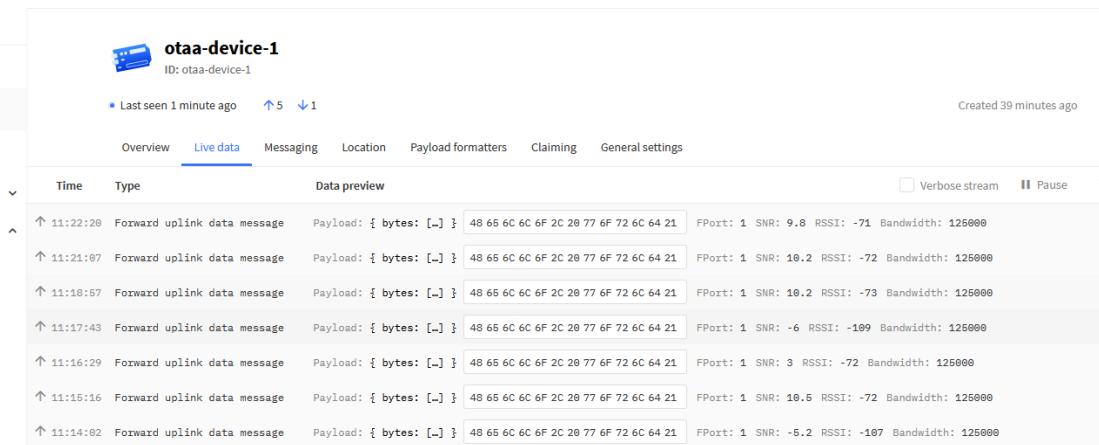
In TTNV3-Gateway page, we can also see the traffic.

This screenshot shows the "TTN Traffic Page" for the device. At the top, it says "TTN Traffic Page shows the device status" with a yellow box. Below this, there is a table with columns for Time, Type, and Data preview. The table shows several log entries. Red arrows point to specific entries: one arrow points to a "Receive uplink message" entry at 18:55:41, and another arrow points to a "Transmit downlink message suc" entry at 18:55:40. A third arrow points to a "Send downlink message" entry at 18:55:40. Another red arrow points to a "Receive uplink message" entry at 18:55:38. A yellow box highlights the text "TTN Get Join request" near the bottom of the table. To the right of the table, there is a "Verbose stream" checkbox and other controls.

Note: The LG02_DNWREQ value in `Arduino_LMIC/src/lmic/config.h` should match downlink frequency from TTNV3. TTNV3 shows 868.1 here, So LG02_DNWREQ should be 868100000

TTNV3 shows 868.1 here, So LG02_DNWREQ should be 868100000

After success Joined, we can see the data in the device page:

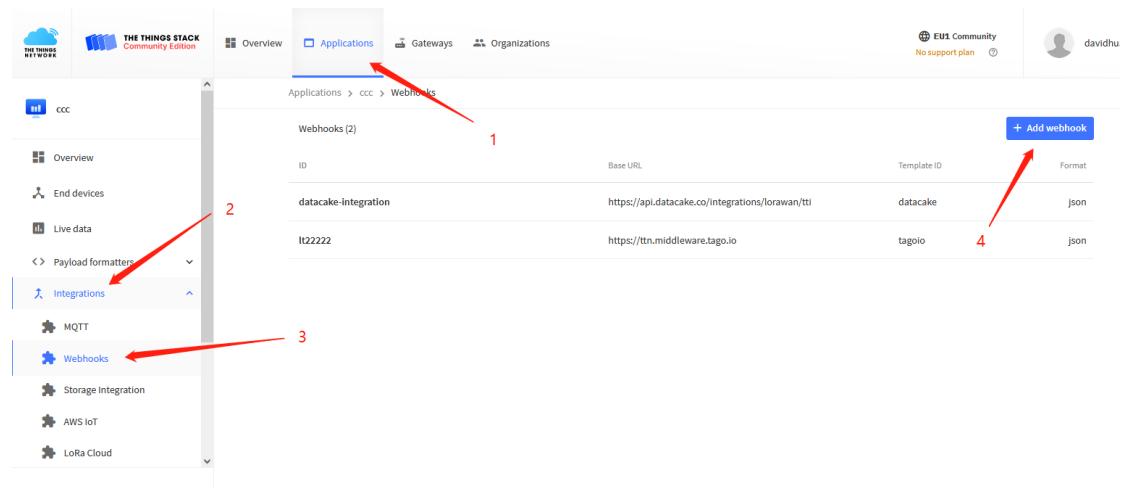


The screenshot shows the DRAGINO Device Management interface. At the top, it displays the device name 'otaa-device-1' and ID 'otaa-device-1'. Below this, there are tabs for 'Overview', 'Live data' (which is selected), 'Messaging', 'Location', 'Payload formatters', 'Claiming', and 'General settings'. The 'Live data' tab shows a table of received messages. The columns are 'Time', 'Type', and 'Data preview'. The data preview shows the raw hex payload and some decoded information like FPort, SNR, RSSI, and bandwidth. There are 7 messages listed, all of which are 'Forward uplink data message' type.

3.4.3 Configure to connect to Mydevices Application Server

In TTNV3, we can see the raw data, now we try to connect it to the application server.

Step 1: Add Mydevice in Application page



The screenshot shows the THE THINGS STACK Community Edition interface. On the left, there is a sidebar with icons for Overview, End devices, Live data, Payload formatters (with 'Integrations' expanded), MQTT, Webhooks (with 'Storage Integration', 'AWS IoT', and 'LoRa Cloud' listed), and a 'Webhooks' section. The main area shows the 'Applications' tab selected under 'Webhooks'. It lists two webhooks: 'datacake-integration' and 'lt2222'. Each entry includes fields for ID, Base URL, Template ID, and Format. A red arrow labeled '1' points to the 'Applications' tab. A red arrow labeled '2' points to the 'Integrations' menu item. A red arrow labeled '3' points to the 'Webhooks' menu item. A red arrow labeled '4' points to the '+ Add webhook' button.

ID	Base URL	Template ID	Format
datacake-integration	https://api.datacake.co/integrations/lorawan/tti	datacake	json
lt2222	https://ttn.middleware.tagio.io	tagio	json

The screenshot shows the DRAGINO CCC interface under the 'Webhooks' section. On the left sidebar, 'Webhooks' is selected. In the main area, a 'Choose webhook template' dialog is open, showing several options:

- Akenza Core**: Integrate with Akenza Core.
- Cayenne**: Drag-and-Drop IoT Project Builder. This option is highlighted with a red arrow.
- Cloud Studio**: Integrate with Cloud Studio IoT platform.
- deZem**
- Homey**: Integrate The Things Stack with your Homey.
- iot in a Box**

The screenshot shows the DRAGINO CCC interface under the 'Webhooks' section. On the left sidebar, 'Webhooks' is selected. In the main area, a 'Add custom webhook' dialog is open, showing the 'Cayenne' template settings:

Template information

Cayenne
Cayenne Drag-and-Drop IoT Project Builder
[About Cayenne](#) | [Documentation](#)

Template settings

Webhook ID * (optional)

Client ID Optional Cayenne Client ID

Create cayenne webhook

Red arrows point to the 'Webhook ID' field and the 'Client ID' field, with the word 'optional' written next to the 'Client ID' field.

Step2: Log in [Mydevices account](#) and add devices.

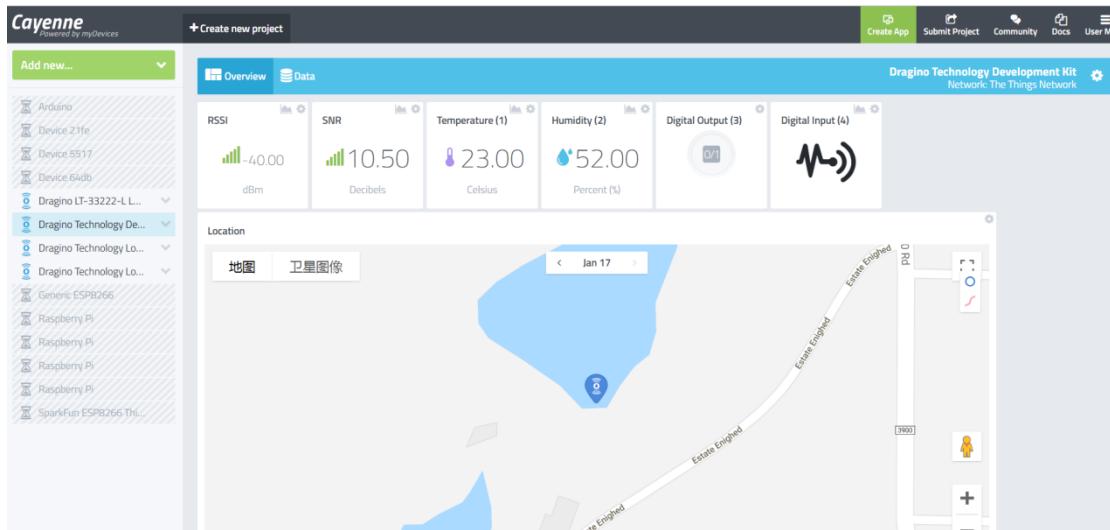
The screenshot shows the Cayenne MyDevices interface. On the left, a sidebar menu has 'Device/Widget' selected, indicated by a red arrow labeled '1'. The main area displays a grid of IoT service logos under the heading 'LoRa'. One specific entry, 'X-TELIA', is highlighted with a red arrow labeled '2'.

The screenshot shows the Cayenne MyDevices interface. The sidebar menu has 'Devices & Widgets' selected, indicated by a red arrow labeled '1'. In the main area, a search bar contains 'Development Kit'. A specific entry, 'Dragino Technology Development Kit', is highlighted with a red arrow labeled '2'.

Add DevEUI of the End node

The screenshot shows the 'Enter Settings' page for the 'Dragino Technology Development Kit'. The 'Name' field is set to 'Dragino Technology Development Kit'. The 'DevEUI' field is highlighted with a red arrow labeled '1' and contains the placeholder 'your DevEUI'. The 'Activation Mode' dropdown is set to 'Already Registered'. Below this, the 'Tracking' section shows a location entry 'Independence, KS 67301美国' with a red arrow labeled '2' pointing to it. At the bottom is a green 'Add device' button with a red arrow labeled '3' pointing to it.

After above steps, we can now see the sensor data in Mydevices:



3.4.4 Use downlink message to control relay

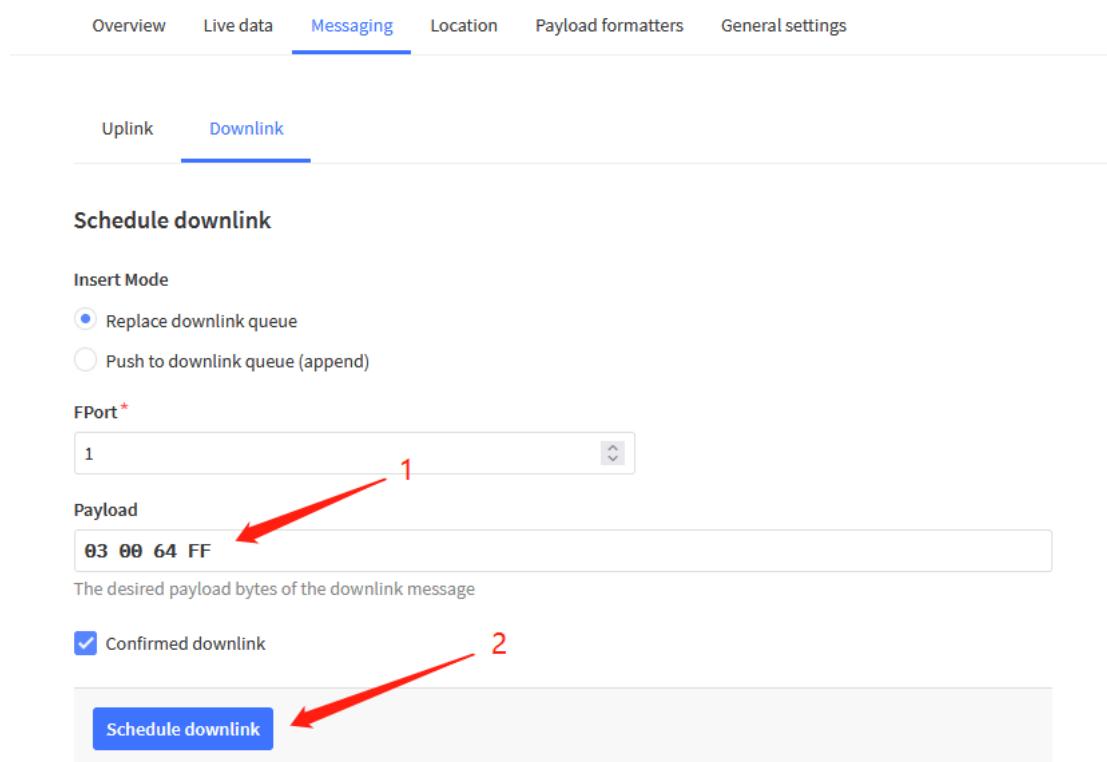
We can use either TTNv3 or Cayenne to control the relay.

Control relay via TTNv3:

The string for ON is: 03 00 64 FF

The string for OFF is: 03 00 00 FF

In put above value in the TTNv3 Downlink payload, we can see the relay can switch between different states, since we are in Class A, the downlink will only happen after each uplink.



Overview Live data **Messaging** Location Payload formatters General settings

Uplink **Downlink**

Schedule downlink

Insert Mode

Replace downlink queue

Push to downlink queue (append)

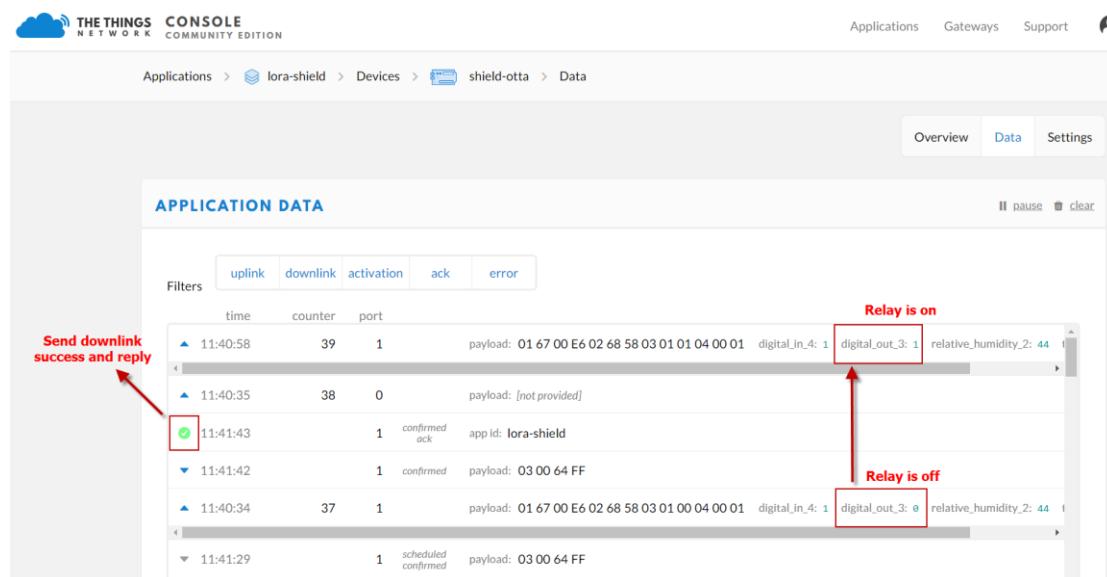
FPort*
1

Payload
03 00 64 FF

The desired payload bytes of the downlink message

Confirmed downlink

Schedule downlink



THE THINGS NETWORK CONSOLE COMMUNITY EDITION

Applications > lora-shield > Devices > shield-otta > Data

Overview Data Settings

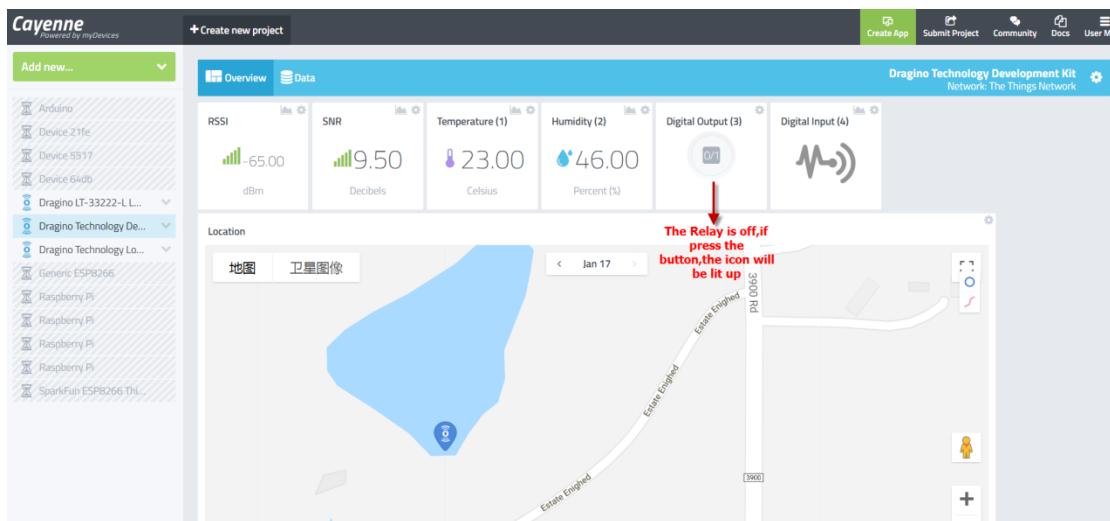
APPLICATION DATA

Filters: uplink, downlink, activation, ack, error

time	counter	port	payload	digital_in_4: 1	digital_out_3: 1	relative_humidity_2: 44
11:40:58	39	1	payload: 01 67 00 E6 02 68 58 03 01 01 04 00 01	1	1	44
11:40:35	38	0	payload: [not provided]			
11:41:43	1	confirmed ack	app id: lora-shield			
11:41:42	1	confirmed	payload: 03 00 64 FF			
11:40:34	37	1	payload: 01 67 00 E6 02 68 58 03 01 00 04 00 01	1	0	44
11:41:29	1	scheduled confirmed	payload: 03 00 64 FF			

Control relay via Cayenne

In Cayenne, just click the digital output button, it will auto send out the command strings: ON: 03 00 64 FF , OFF is: 03 00 00 FF



Cayenne will pass the string to TTNV3 and TTNV3 will show as above. In the serial monitor of End Node, we can see below output if downlink string arrives:

```

lora_shield_cayenne_and_ttn_abpClient | Arduino 1.8.5
文件 编辑 项目 工具 帮助
lora_shield_cayenne_and_ttn_abpClient
49 static const u1_t PROGMEM APPSKEY
50
51 // LoRaWAN end-device address (De
52 static const u4_t DEVADDR = 0x260
53
54 // These callbacks are only used
55 // left empty here (we cannot lea
56 // DISABLE_JOIN is set in config
57 void os_getArtEui (u1_t* buf) {
58 void os_getDevEui (u1_t* buf) {
59 void os_getDevKey (u1_t* buf) {
60
avrduke done. Thank you.

[25.00°C, 66.00%]
3955087: engineUpdate, opmode=0x908
3957856: TXMODE, freq=868100000, len=23, SF=7, BW=125, CR=4/
Packet queued
4012411: RXMODE_SINGLE, freq=868100000, SF=7, BW=125, CR=4/5
4022496: Received downlink, window=RX1, port=1, ack=0
4022541: EV_TXCOMPLETE (includes waiting for RX windows)
Received :
3 0 64 FF
Set pin to HIGH.

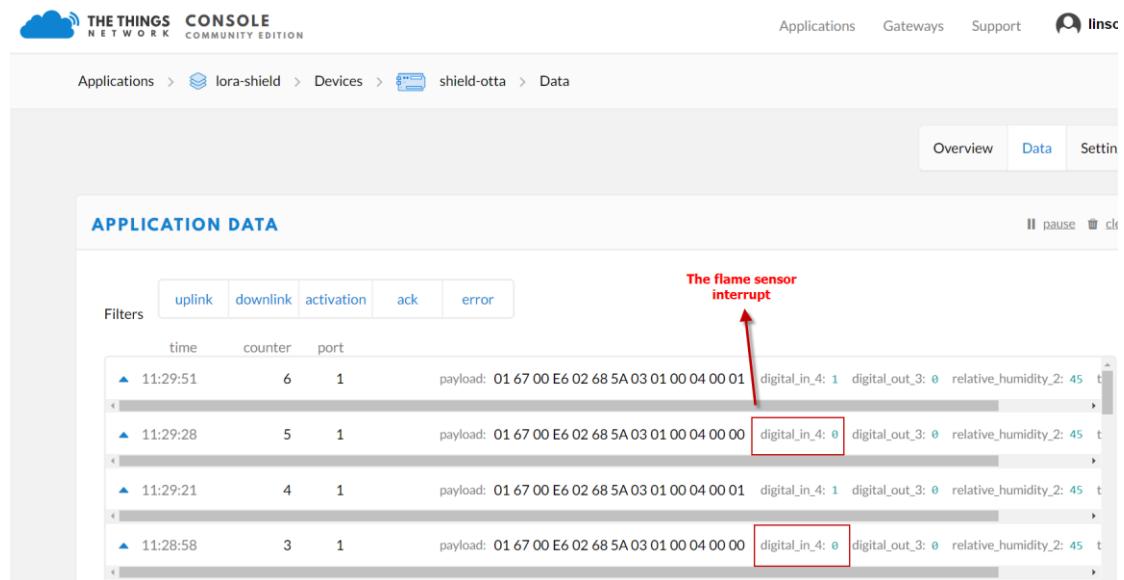
4028292: engineUpdate, opmode=0x810
4030903: TXMODE, freq=868100000, len=12, SF=7, BW=125, CR=4/
4085487: RXMODE_SINGLE, freq=868100000, SF=7, BW=125, CR=4/5
4124275: RXMODE_SINGLE, freq=869525000, SF=9, BW=125, CR=4/5
4188702: EV_TXCOMPLETE (includes waiting for RX windows)
4188747: engineUpdate, opmode=0x900

```

3.4.5 Test with Interrupt

The temperature & humidity in this example are updated periodically (once several minutes/hours), in some case, we need to update the data once an action is happen. So we need to use interrupt.

The DO pin of Flame sensor is high in normal case. While it detects a flame, this pin will become low and act as an external interrupt for Arduino. The Arduino UNO will then immediately upload the temperature and humidity to TTNV3

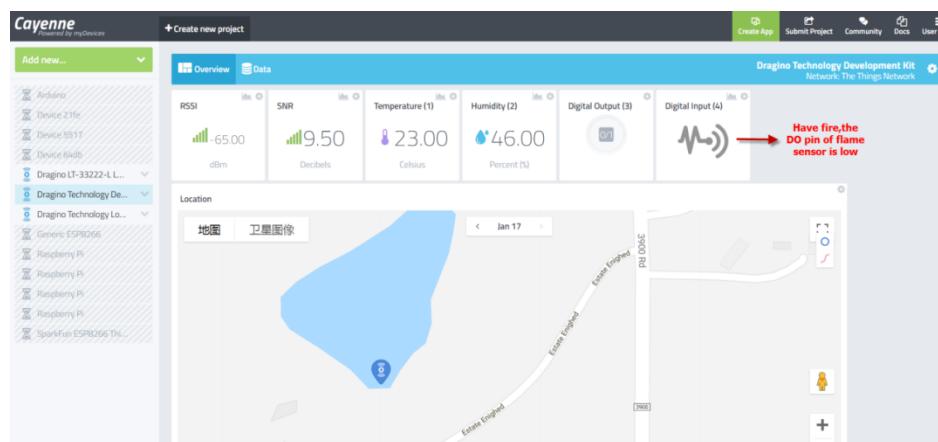


The screenshot shows the 'APPLICATION DATA' section of the The Things Network Console. It lists four uplink events:

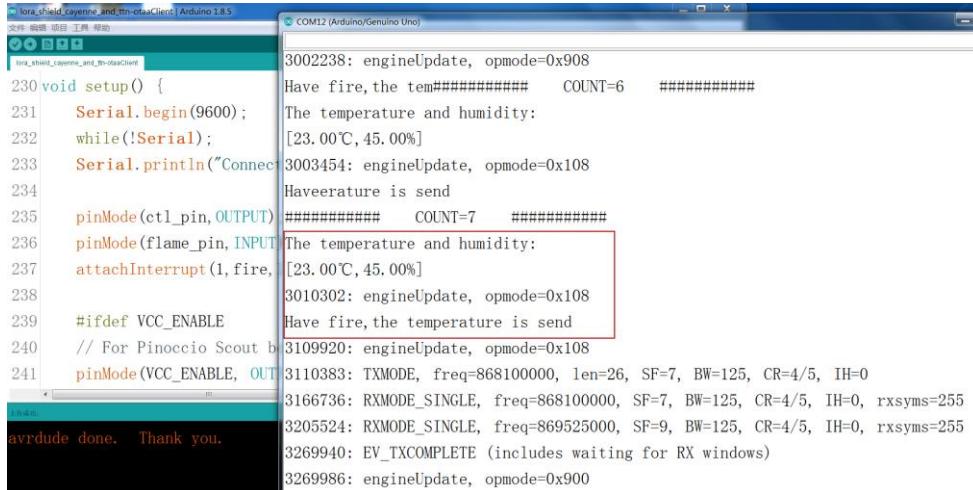
- 11:29:51, counter 6, port 1, payload: 01 67 00 E6 02 68 5A 03 01 00 04 00 01, digital_in_4: 1, digital_out_3: 0, relative_humidity_2: 45
- 11:29:28, counter 5, port 1, payload: 01 67 00 E6 02 68 5A 03 01 00 04 00 00, digital_in_4: 0, digital_out_3: 0, relative_humidity_2: 45
- 11:29:21, counter 4, port 1, payload: 01 67 00 E6 02 68 5A 03 01 00 04 00 01, digital_in_4: 1, digital_out_3: 0, relative_humidity_2: 45
- 11:28:58, counter 3, port 1, payload: 01 67 00 E6 02 68 5A 03 01 00 04 00 00, digital_in_4: 0, digital_out_3: 0, relative_humidity_2: 45

A red arrow points to the second event (11:29:28) with the annotation "The flame sensor interrupt".

Then we can see on the cayenne:



The screenshot shows the Cayenne IoT Platform interface. On the left, there's a sidebar with a list of devices, including 'Dragino Technology Dev...'. The main area displays device status metrics: RSSI (65.00 dBm), SNR (9.50 Decibels), Temperature (23.00 Celsius), Humidity (46.00 Percent), Digital Output (3), and Digital Input (4). A red arrow points to the Digital Input (4) section, which shows a waveform icon and the text "Have fire.the DO pin of flame sensor is low".



The screenshot shows the Arduino IDE with an open sketch named "lora_shield_cayenne_and_fotaClient". The code is as follows:

```

230 void setup() {
231     Serial.begin(9600);
232     while(!Serial);
233     Serial.println("Connect");
234
235     pinMode(ctl_pin, OUTPUT);
236     pinMode(flame_pin, INPUT);
237     attachInterrupt(1, fire, RISING);
238
239 #ifdef VCC_ENABLE
240 // For Pinoccio Scout board
241     pinMode(VCC_ENABLE, OUTPUT);

```

The serial monitor output shows the following messages:

```

3002238: engineUpdate, opmode=0x908
Have fire, the tem##### COUNT=6 #####
The temperature and humidity:
[23. 00°C, 45. 00%]
3003454: engineUpdate, opmode=0x108
Havevperature is send
#####
The temperature and humidity:
[23. 00°C, 45. 00%]
3010302: engineUpdate, opmode=0x108
Have fire, the temperature is send
3109920: engineUpdate, opmode=0x108
3110383: TXMODE, freq=868100000, len=26, SF=7, BW=125, CR=4/5, IH=0
3166736: RXMODE_SINGLE, freq=868100000, SF=7, BW=125, CR=4/5, IH=0, rxsysms=255
3205524: RXMODE_SINGLE, freq=869525000, SF=9, BW=125, CR=4/5, IH=0, rxsysms=255
3269940: EV_TXCOMPLETE (includes waiting for RX windows)
3269986: engineUpdate, opmode=0x900

```

At the bottom of the serial monitor, it says "avrduke done. Thank you."

3.5 Create LoRa/GPS Shield End Node

3.5.1 Hardware connection

The method to use LoRa/GPS Shield is similar with LoRa Shield. Below is the hardware connection of LoRa GPS Shield.



3.5.2 Set up ABP device in TTNV3 and upload software to UNO

In LoRa Shield, we set up OTAA for connection. In this example, we will try ABP mode.

Step 1: Create an ABP device in TTNV3 server --> Application page. And change it to ABP mode.

Overview Applications Gateways Organizations

Applications > ccc

ccc
ID: 123

Last seen 20 seconds ago 6 End devices 2 Collaborators 3 API keys Created 170 days ago

General information		Live data	
Application ID	123	See all activity →	
Created at	Feb 2, 2021 11:12:30	↑ 19:06:01	sheild Forward uplink data message
Last updated at	Apr 30, 2021 11:00:33	↑ 19:05:00	sheild Forward uplink data message

End devices (6)

ID	Name	DevEUI	JoinEUI	Last seen

+ Import end devices + Add end device

Overview Applications Gateways Organizations

Applications > ccc > End devices > Register manually

Register end device

From The LoRaWAN Device Repository Manually

Preparation

Activation mode *

- Over the air activation (OTAA)
- Activation by personalization (ABP)
- Multicast
- Do not configure activation

LoRaWAN version *

MAC V1.0.2

Network Server address
eu1.cloud.thethings.network

Application Server address
eu1.cloud.thethings.network

Step 2: Input keys into Arduino Sketch.

The sketch for the LoRa /GPS Shield is [LoRa GPS Sketch code](#)

Overview Live data Messaging Location Payload formatters General settings

TTNV3 LoRaWAN End Device page

General information		Live data	
End device ID	sheild	↑ 19:45:52	Forward uplink data message Payload: { bytes: [...] } 48 65 6C
Description	This end device has no description	↑ 19:44:50	Forward uplink data message Payload: { bytes: [...] } 48 65 6C
Created at	Jul 22, 2021 09:31:21	↑ 19:42:47	Forward uplink data message Payload: { bytes: [...] } 48 65 6C
Activation information		↑ 19:41:46	Forward uplink data message Payload: { bytes: [...] } 48 65 6C
AppEUI	n/a	↑ 19:40:44	Forward uplink data message Payload: { bytes: [...] } 48 65 6C
DevEUI	45 45 45 52 52 52 52	↑ 19:39:43	Forward uplink data message Payload: { bytes: [...] } 48 65 6C
Session information		See all activity →	
Device address	03 FF 00 11	↑ 19:45:52	Forward uplink data message Payload: { bytes: [...] } 48 65 6C
NwkSKey	2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 C...	↑ 19:44:50	Forward uplink data message Payload: { bytes: [...] } 48 65 6C
SNwkSIntKey	2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 C...	↑ 19:42:47	Forward uplink data message Payload: { bytes: [...] } 48 65 6C
NwkSEncKey	2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 C...	↑ 19:41:46	Forward uplink data message Payload: { bytes: [...] } 48 65 6C
AppSKey	2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 C...	↑ 19:40:44	Forward uplink data message Payload: { bytes: [...] } 48 65 6C

Location Change location settings →

Make sure the Network Session Key and App Session Key are in MSB order

No location information available

ttn-abp

Arduino Sketch TTNV3-abp

Input the keys from TTNv3

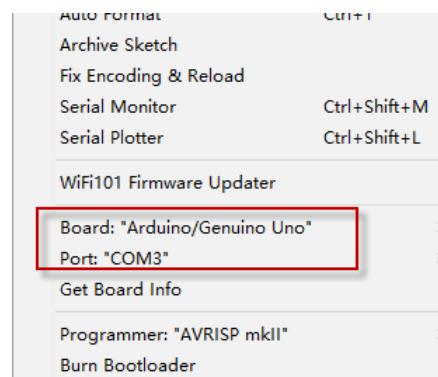
```
#include <mic.h>
#include <hal/hal.h>
#include <SPI.h>

// LoRaWAN NwkSKey, network session key
// This is the default Semtech key, which is used by the early prototype TTN
// network.
static const PROGMEM u1_t NWKSKEY[16] = { 0x9A, 0xEA, 0xD0, 0x93, 0x06, 0xE3, 0x2B, 0x73, 0xDD, 0x54, 0x7B, 0x8B, 0xFF, 0xDC, 0x20, 0xF9 };

// LoRaWAN AppSKey, application session key
// This is the default Semtech key, which is used by the early prototype TTN
// network.
static const u1_t PROGMEM APPSKEY[16] = { 0xB6, 0x07, 0x5B, 0xB5, 0xE4, 0xCE, 0x40, 0xA2, 0xA3, 0xEE, 0x7B, 0xDF, 0xDC, 0x23, 0x0E, 0x2B };

// LoRaWAN end-device address (DevAddr)
static const u4_t DEVADDR = 0x26011C22 ; //-- Change this address for every node!
```

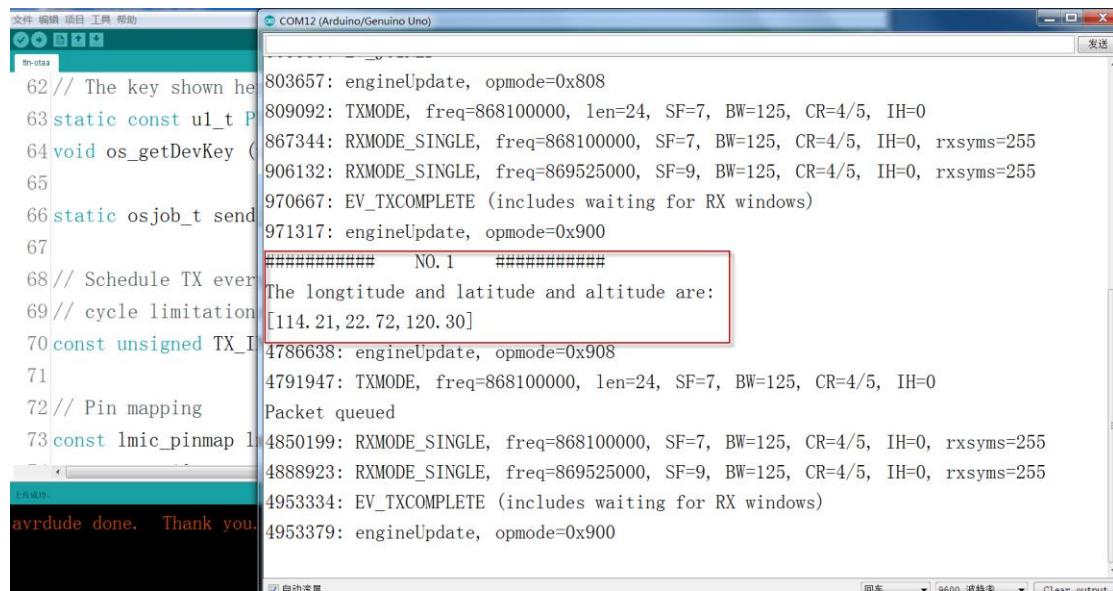
Choose Arduino UNO to upload the sketch to LoRa GPS Shield and UNO



All other steps are similar with how we use with LoRa Shield.

Below are the outputs for reference:

Output from LoRa GPS Shield:



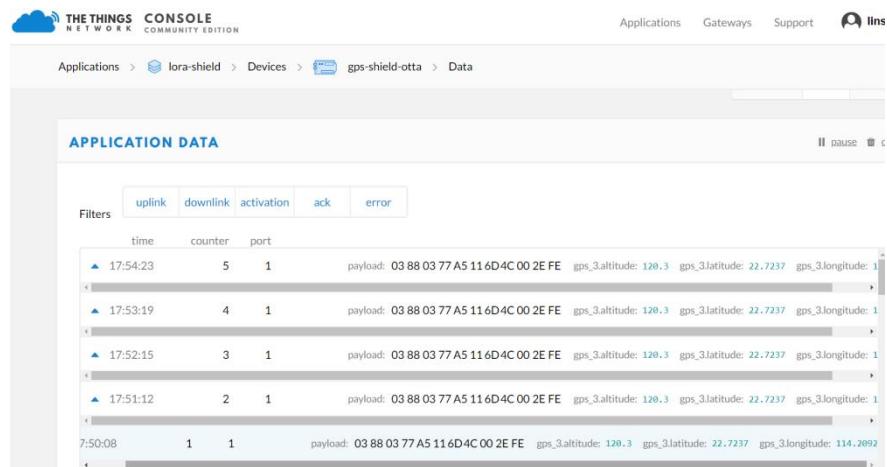
```

文件-编辑-项目-工具-帮助
th-otaa
803657: engineUpdate, opmode=0x808
809092: TXMODE, freq=868100000, len=24, SF=7, BW=125, CR=4/5, IH=0
867344: RXMODE_SINGLE, freq=868100000, SF=7, BW=125, CR=4/5, IH=0, rxsysms=255
906132: RXMODE_SINGLE, freq=869525000, SF=9, BW=125, CR=4/5, IH=0, rxsysms=255
970667: EV_TXCOMPLETE (includes waiting for RX windows)
971317: engineUpdate, opmode=0x900
#####
NO. 1 #####
The longitude and latitude and altitude are:
[114.21, 22.72, 120.30]
4786638: engineUpdate, opmode=0x908
4791947: TXMODE, freq=868100000, len=24, SF=7, BW=125, CR=4/5, IH=0
Packet queued
4850199: RXMODE_SINGLE, freq=868100000, SF=7, BW=125, CR=4/5, IH=0, rxsysms=255
4888923: RXMODE_SINGLE, freq=869525000, SF=9, BW=125, CR=4/5, IH=0, rxsysms=255
4953334: EV_TXCOMPLETE (includes waiting for RX windows)
4953379: engineUpdate, opmode=0x900

avrduke done. Thank you.

```

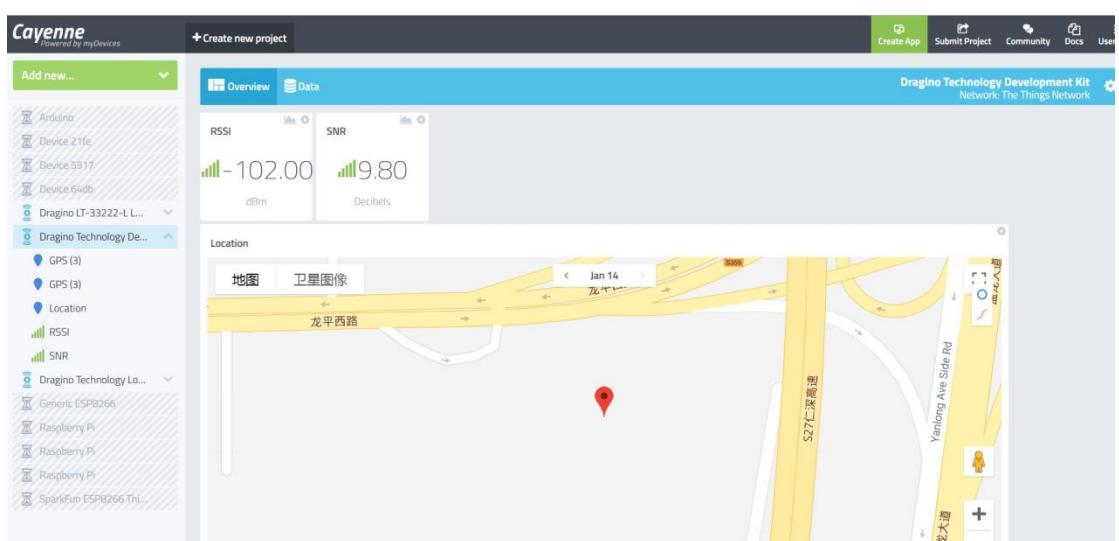
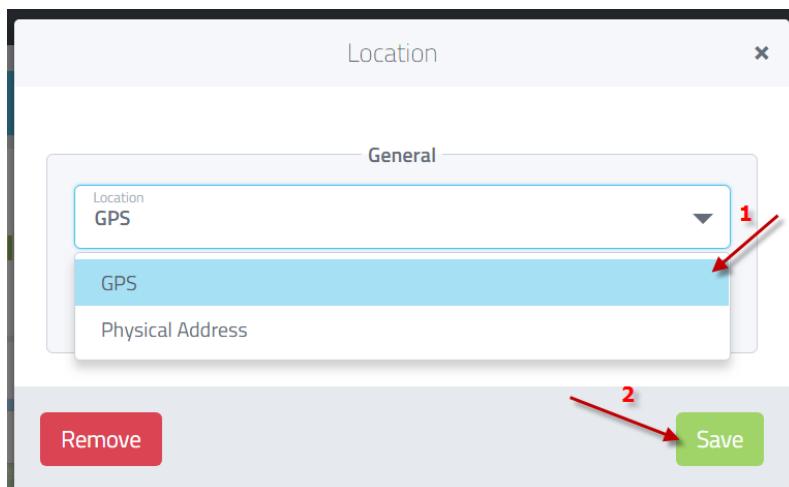
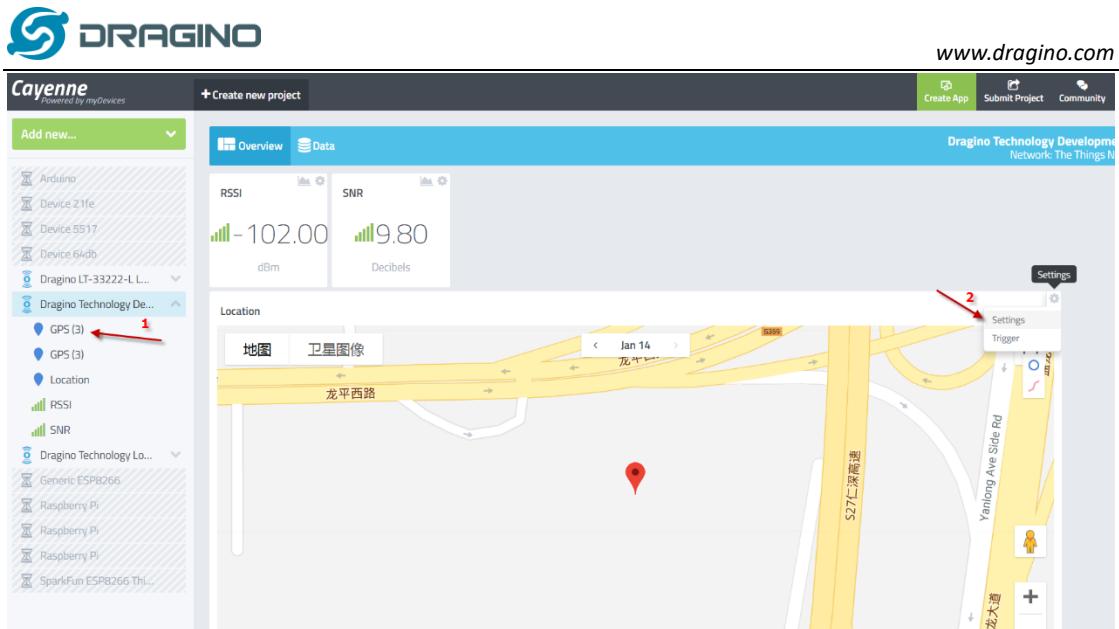
Upload GPS data to TTNV3:



The screenshot shows the 'APPLICATION DATA' section of the The Things Network Console. It displays a list of uplink messages received by the gateway. Each message includes the time, port number, and the raw payload. The payload contains GPS coordinates: latitude, longitude, and altitude.

time	counter	port	payload
17:54:23	5	1	03 88 03 77 A5 11 6D 4C 00 2E FE gps_3.latitude: 120.3 gps_3.longitude: 22.7237 gps_3.altitude: 1
17:53:19	4	1	03 88 03 77 A5 11 6D 4C 00 2E FE gps_3.latitude: 120.3 gps_3.longitude: 22.7237 gps_3.altitude: 1
17:52:15	3	1	03 88 03 77 A5 11 6D 4C 00 2E FE gps_3.latitude: 120.3 gps_3.longitude: 22.7237 gps_3.altitude: 1
17:51:12	2	1	03 88 03 77 A5 11 6D 4C 00 2E FE gps_3.latitude: 120.3 gps_3.longitude: 22.7237 gps_3.altitude: 1
7:50:08	1	1	03 88 03 77 A5 11 6D 4C 00 2E FE gps_3.latitude: 120.3 gps_3.longitude: 22.7237 gps_3.altitude: 114.2092

Output in Cayenne:



3.6 Conclusion and limitation

3.6.1 Overview for the example

This example shows how to set up a simple LoRaWAN network with public server. The LoRaWAN specification is for easy deploy the IoT network base on LoRa wireless. It contains the encryption, MAC control, device management etc. More info about LoRaWAN, please see [this link](#).

There are some frequently ask points for the example:

1/ Difference between OTAA & ABP mode:

We have tested OTAA and ABP mode for LoRaWAN. They are two different modes. In OTAA mode, we can see the device will send a join request, the IoT server will send back a Join confirm with dynamic device address, network session key and app session key. Then the device will use these key to communicate with the LoRaWAN server. This make sure the device will only communicate with one server.

In ABP mode, it will use the FIX device address, network session key and app session key. It doesn't have join process. So in theory, any server with match keys is possible to decrypt the data from this end device.

We can see OTAA has better security than ABP mode.

2/ AES 128 encryption:

The data between end device and server are AES128 encryption. So the gateway can't parse the packets, it just forward them.

3/ LoRaWAN Network Server:

A LoRaWAN network server is necessary in a LoRaWAN network for device control/management/data management. If user wants to build the NS , there are some open sources LoRaWAN NS such as [LoRaServer](#) can be used. And some gateways already include LoRaWAN NS (this is also a plan for LG01-N).

4/ Downlink message

In this example, we use LoRaWAN Class A. The end node will open two short downlink windows after each uplink. More info about LoRaWAN class A, please refer [LoRaWAN specification](#).

3.6.2 Limitations

The LG01-N is a single channel gateway (Same for LG02). And there are limitations:

1/ It works only on one frequency at a time. It can support multiply end nodes, but all end nodes must transmit data at the same frequency so the LG01-N can receive it. For example: if the End node transmits at 868.1Mhz, The LG01-N's RX setting must be 868.1Mhz so to receive this packet.

2/ It works only for one DR at a time. DR specifies the Spreading Factor and Bandwidth. In LG01-N, even the rx frequency match , if DR doesn't match, it still can't get the sensor data.

3/ LoRaWAN compatible issue

In LoRaWAN protocol, the LoRaWAN end nodes send data in a hopping frequency. Since LG01-N only supports one single frequency, it will only be able to receive the packets sent from the same radio parameters (frequency & DR) in LG01-N.

For example, in EU868, a standard LoRaWAN device may send the data in eight frequencies with different Frequency & SF, such as:

```
LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B), BAND_CENTI); // g-band
LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK, DR_FSK), BAND_MILLI); // g2-band
```

So the LG01-N will only able to receive the 868100000, SF7 packet and will not receive others. Means only one packet will arrive the TTNV3 server in every 8 packet sent from the LoRaWAN end node.

If user wants to receive all packets from LoRaWAN end node, user needs to set up the LoRaWAN node to send packets in a single frequency.

4/ Downlink & OTAA issue

According to the LoRaWAN class A spec, the end node will open two receive windows to get the message from LoRaWAN server for OTAA or downlink function. These two receive windows are quite short (milliseconds), if LoRa packet from the gateway can't reach End Node in the receive window time, the end node won't get the rx message and Downlink / OTAA won't work.

In our example, the Arduino LMIC library is modified to enlarge the RX window to let OTAA & downlink works.

4 Example 2: Test with a MQTT IoT Server

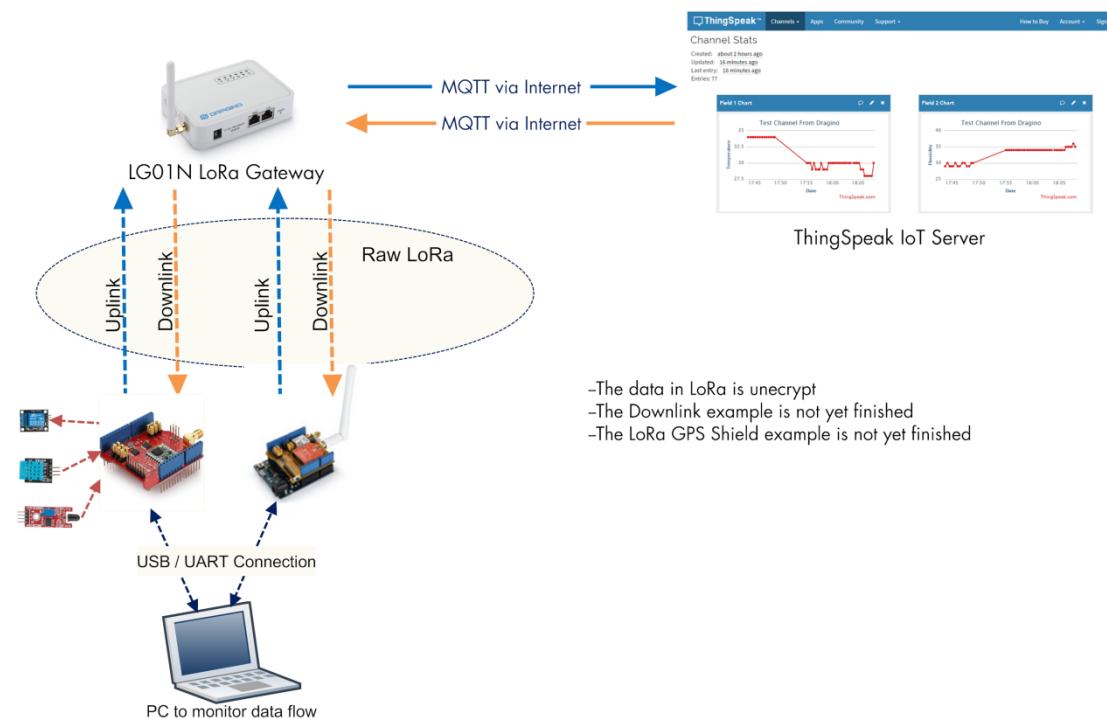
This example describes how to use LG01-N, LoRa Shield & LoRa GPS Shield to set up a LoRa network and connect it to [ThingSpeak IoT Server](#).

A Video Instruction of this example can be found at this url: <https://youtu.be/asoNyFYZam0>

4.1 Topology and Data Flow

The network topology and dataflow for the example is as below:

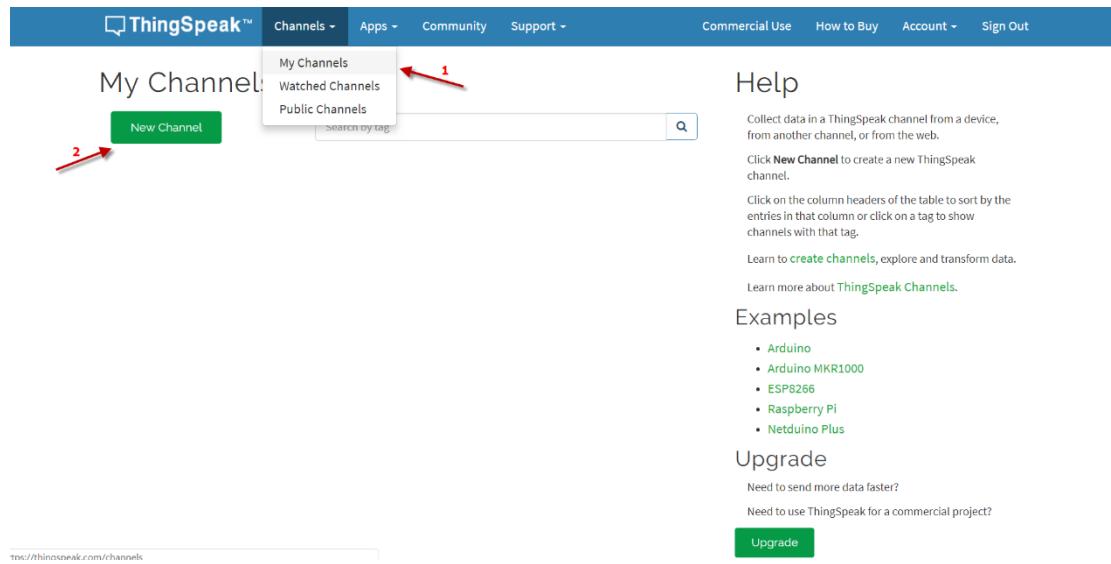
Topology for ThingSpeak Connection:



In next section we will start to configure for this example.

4.2 Set up sensor channels in ThingSpeak

Step 1: Log in ThingSpeak and set up channels

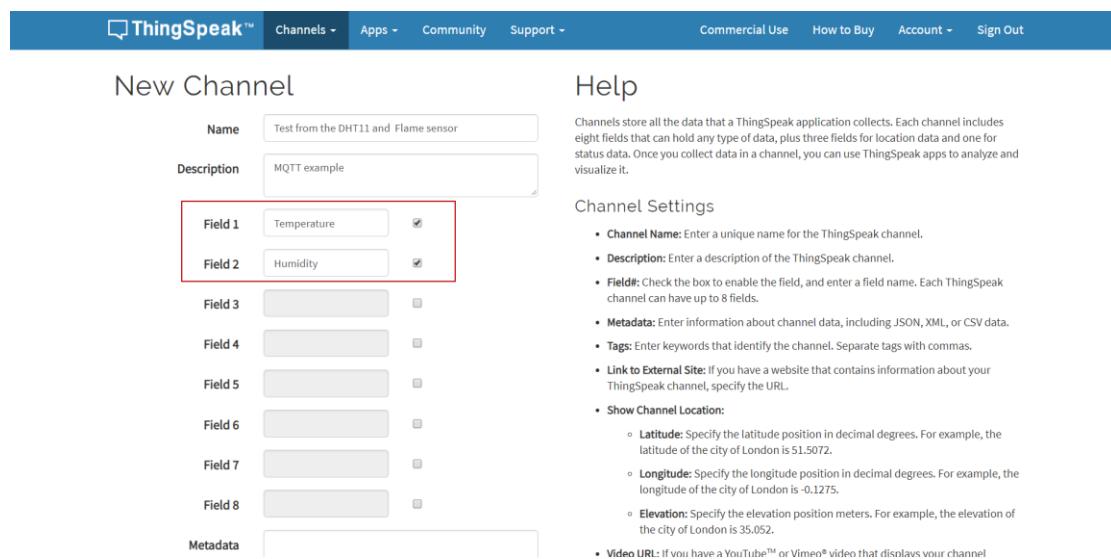


The screenshot shows the ThingSpeak interface. At the top, there's a navigation bar with links for 'Channels', 'Apps', 'Community', 'Support', 'Commercial Use', 'How to Buy', 'Account', and 'Sign Out'. Below the navigation bar, the main content area has a title 'My Channel' and a green 'New Channel' button. To the right of the channel list, there's a 'Help' section with instructions on how to collect data from a device or channel. It also lists examples like Arduino, ESP8266, Raspberry Pi, and Netduino Plus. There's a 'Upgrade' button at the bottom of the help section.

Set up two channels:

Field 1: Temperature

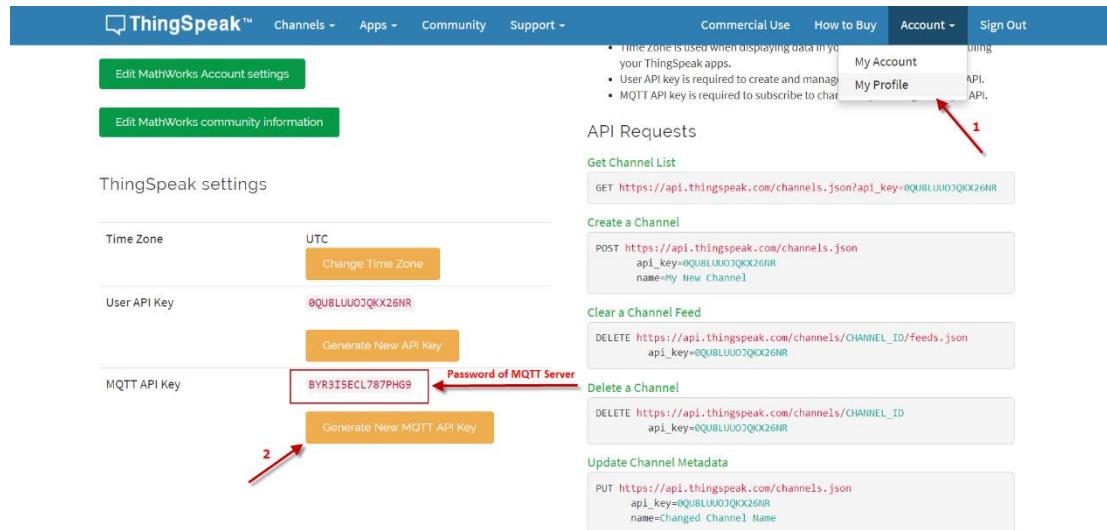
Field 2: Humidity



The screenshot shows the 'New Channel' setup page. It has fields for 'Name' (Test from the DHT11 and Flame sensor) and 'Description' (MQTT example). Below these, there are eight fields labeled 'Field 1' through 'Field 8', with 'Field 1' and 'Field 2' being checked and highlighted with a red border. There's also a 'Metadata' field. To the right, there's a 'Help' section with detailed instructions on channel settings, including descriptions for each field type and additional options like latitude, longitude, and elevation.

Step 2: Get MQTT keys for these channels.

Go to Account → My profile and get the [MQTT API Key](#)



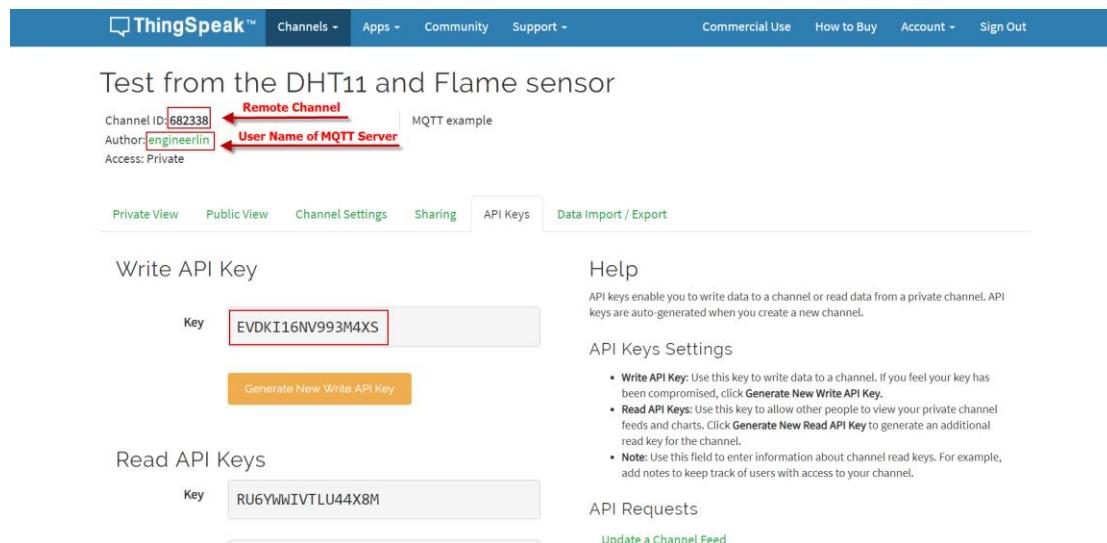
The screenshot shows the 'ThingSpeak™' account settings page. In the top right corner, there is a dropdown menu with options: 'My Account', 'My Profile' (which is highlighted), and 'API'. A red arrow labeled '1' points to the 'My Profile' option. Below the dropdown, the 'MQTT API Key' section is visible, containing the key 'BYR3I5ECL787PHG9'. A red arrow labeled '2' points to this key. The page also includes sections for 'ThingSpeak settings' (Time Zone set to UTC), 'API Requests' (with examples for GET, POST, DELETE, PUT methods), and 'Create a Channel' (with examples for creating, clearing, deleting, and updating channels).

Go to channel page: get the sensor channel:

Channel ID: This is the remote Channel ID in ThingSpeak

Author: User Name for MQTT connection

Write API Key: API key for each channel



The screenshot shows the 'Test from the DHT11 and Flame sensor' channel page. At the top, it displays the Channel ID '682338' and the Author 'engineerin'. A red arrow labeled 'Remote Channel' points to the Channel ID, and another red arrow labeled 'User Name of MQTT Server' points to the Author. Below this, there are tabs for 'Private View', 'Public View', 'Channel Settings', 'Sharing', 'API Keys' (which is selected), and 'Data Import / Export'. The 'API Keys' section contains a 'Write API Key' input field with the value 'EVDKI16NV993M4XS' and a 'Generate New Write API Key' button. To the right, there is a 'Help' section with information about API keys and a 'API Keys Settings' section with notes about Write API Key, Read API Keys, and Notes. At the bottom, there is an 'API Requests' section with a 'Update a Channel Feed' button.

4.3 Simulate MQTT uplink via PC's MQTT tool

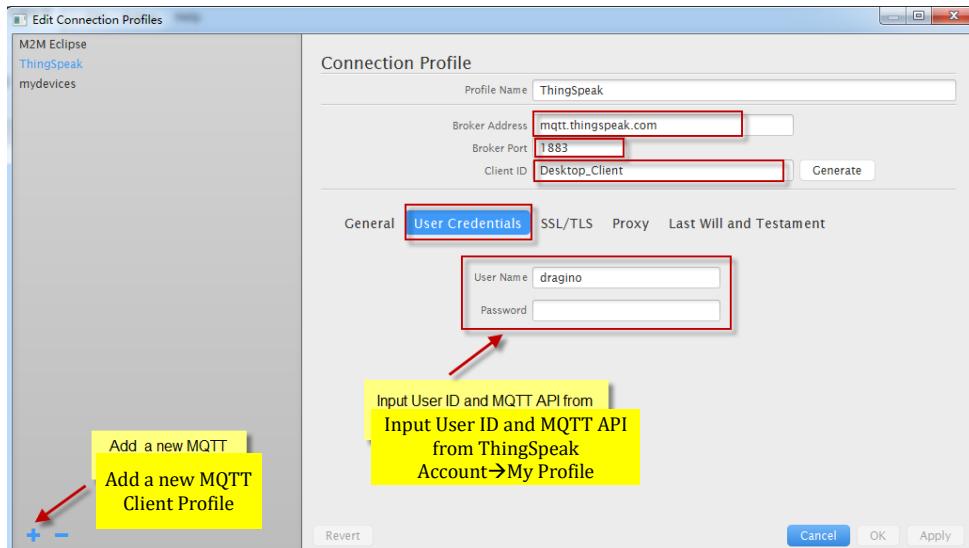
This step is not necessary, it just to help user to understand the MQTT protocol and simulate the MQTT connection to ThingSpeak. And check if the account info is valid and correct.

In the PC, download and install [MQTT.fx](#). Open MQTT.fx and configure add a new MQTT client, as below:

Broker Address: mqtt.thingspeak.com

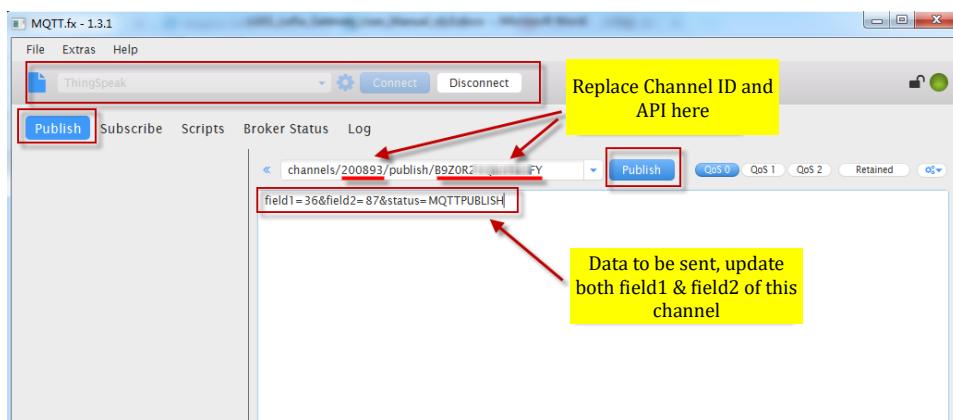
Broker Port: 1883

Client ID: User Defined

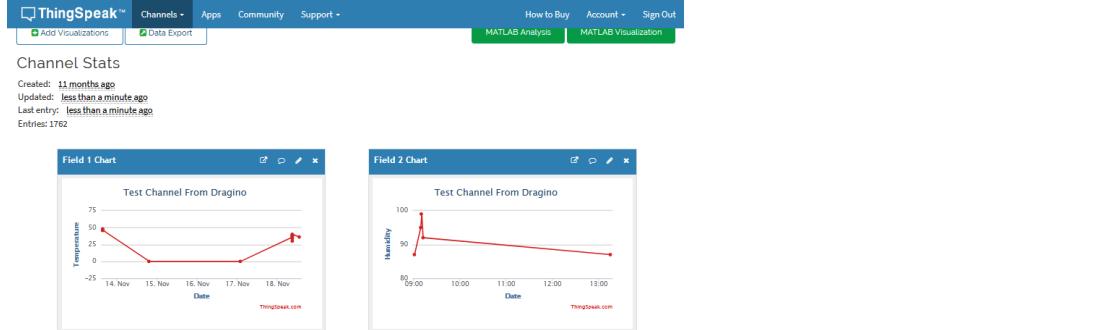


After add the profile, connect it and publish to the corresponding Channel with correct API key.

MQTT API see [this document](#):



If update successful, we can see the update in the channel:



4.4 Try MQTT Publish with LG01-N Linux command

This step is also not necessary; it is to show the basic command used for MQTT connection and will help for further debug when connection fails.

First, we need to make sure the LG01-N has internet access. We can log in the SSH and ping an Internet address and see if there is reply. As below:

```
172.31.255.254 - SecureCRT
文件(F) 编辑(E) 查看(V) 选项(O) 传输(T) 脚本(S) 工具(L) 帮助(H)
阿里云_美国服务器 | 172.31.255.254
root@dragino-146d78:~# ping www.163.com
PING www.163.com (58.63.233.35): 56 data bytes
64 bytes from 58.63.233.35: seq=0 ttl=54 time=8.231 ms
64 bytes from 58.63.233.35: seq=1 ttl=54 time=8.709 ms
64 bytes from 58.63.233.35: seq=2 ttl=54 time=8.313 ms
64 bytes from 58.63.233.35: seq=3 ttl=54 time=7.953 ms
64 bytes from 58.63.233.35: seq=4 ttl=54 time=8.539 ms
^C
--- www.163.com ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 7.953/8.349/8.709 ms
root@dragino-146d78:~#
```

LG01-N has built-in Linux utility **mosquitto_pub**. We can use this command to publish the data to ThingSpeak.

The command to update a feed is as below:

```
mosquitto_pub -h mqtt.thingspeak.com -p 1883 -u dragino -P QZXTxxxxxO2J -i
dragino_Client -t channels/200893/publish/B9Z0R25QNVEBKIFY -m
"field1=34&field2=89&status=MQTTPUBLISH"
```

(Make sure the “” is included, otherwise only one data will be uploaded)

Below is the output window:

```
172.31.255.254 - SecureCRT
文件(F) 编辑(E) 查看(V) 选项(O) 传输(T) 脚本(S) 工具(L) 帮助(H)
172.31.255.254
root@dragino-146d78:~# mosquitto_pub -h mqtt.thingspeak.com -p 1883 -u dragino -P Q
ZXTxxxxxO2J -i dragino_Client -t channels/200893/publish/B9Z0R25QNVEBKIFY -m "
field1=34&field2=89&status=MQTTPUBLISH"
root@dragino-146d78:~#
```

After running this command, we can see the data are updated to ThingSpeak, which has same result as what we did at mqtt.fx.

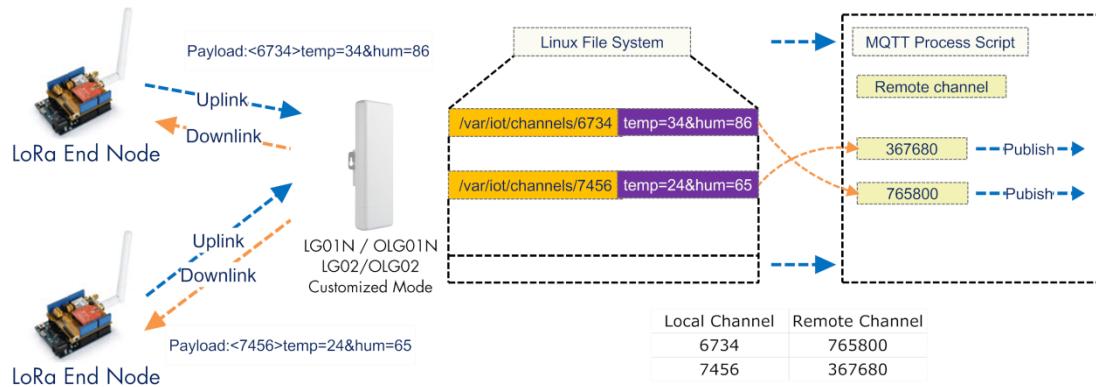
So we success to use LG01-N to uplink data to ThingSpeak, the **mosquitto_pub** command is executed in the Linux side, finally, we will have to call **mosquitto_pub** command while the LoRa sensor data arrive. We will explain how to do that in next step.

4.5 Configure LG01-N Gateway

4.5.1 Publish Logic

In LG01-N (firmware version > LG02_LG08--build-v5.1.1545908833-20181227-1908), there is a built-in script to process the MQTT data. The logic of this flow is as below:

How MQTT script works:



Operate Principle:

- > LoRa End Node sends the data to gateway in specify format: <node_ID>value
- > Gateway get the data and will put the data in corresponding files under /var/iot/channels.
- > MQTT Process Script will publish data to remote channel according to the pre-configure mapping

Step1: Configure LG01-N to act as MQTT mode

dragino-1b7060 Status ▾ System ▾ Network ▾ Service ▾ Logout

LoRa Gateway Settings

Configuration to communicate with LoRa devices and LoRaWAN server

LoRaWAN Server Settings

IoT Service	LoRaRAW forward to MQTT ser ▾
Debug Level	Little message output ▾
Service Provider	The Things Network ▾

Step2: Configure MQTT server info

MQTT Server Settings

Configuration to communicate with MQTT server

Configure MQTT Server

Select Server	ThingSpeak MQTT
User Name [-u]	dragino1
Password [-P]	32W6GMEXYTEQ7049
Client ID [-i]	dragino_Client

In step 2, we have below settings:

- ✓ UserName[-u option]: Input Author (user name for MQTT Connection)
- ✓ Password[-P option]: Input MQTT API key

- ✓ Client_ID[-i]: dragino_Client (can put any string)
- ✓ Because we choose Thingspeak so we have below pre-set options but not show in web
 - Broker Address[-h]: mqtt.thingspeak.com
 - Broker Port[-p]: 1883
 - Topic Format[-t]: channels/CHANNEL/publish/WRITE_API.
 - Data String Format[-m]: DATA&status=MQTTPUBLISH

And we configure this channel:

- ✓ Local Channel ID: 10009
- ✓ Remote Channel ID: 396640
- ✓ Write_api_key: Write API key for this channel.

In the mqtt script, the upper **CHANNEL** will be replaced by the parameter (remote channel in IoT server). and the **WRITE_API** will be replaced by the settings in write api key. The **DATA** will be replaced by the value stored in the /var/iot/channels/LOCAL_CHANNEL file.

MQTT script will keep checking the files in /var/iot/channels/. If it finds a match Local channel, then the MQTT script will send out the data of this local channel to a remote channel according to the setting above.

User can also enable MQTT debug level and run logread in Linux console to see how the mqtt command is composed. Below is an example:

```

Tue Nov 27 15:07:43 2018 kern.notice syslog: [IOT-MQTT]: Found Local channels:
Tue Nov 27 15:07:49 2018 kern.notice syslog: [IOT-MQTT]: Check for sensor update
Tue Nov 27 15:07:49 2018 kern.notice syslog: [IOT-MQTT]: Found Local channels:
Tue Nov 27 15:07:55 2018 kern.notice syslog: [IOT-MQTT]: Check for sensor update
Tue Nov 27 15:07:55 2018 kern.notice syslog: [IOT-MQTT]: Found Local channels:
Tue Nov 27 15:07:59 2018 kern.notice syslog: [IOT-MQTT]: Internet Connection Check: FAIL
Tue Nov 27 15:08:01 2018 kern.notice syslog: [IOT-MQTT]: Check for sensor update
Tue Nov 27 15:08:01 2018 kern.notice syslog: [IOT-MQTT]: Found Local channels:
Tue Nov 27 15:08:02 2018 kern.notice syslog: [IOT-MQTT]: DNS Resolve Check: FAIL
Tue Nov 27 15:08:02 2018 kern.notice syslog: [IOT-MQTT]: Internet Connection Check: FAIL
Tue Nov 27 15:08:03 2018 kern.notice syslog: [IOT-MQTT]: Topic Format: v1/USERNAME/things/CLIENTID/data/CHANNEL
Tue Nov 27 15:08:03 2018 kern.notice syslog: [IOT-MQTT]: Data Format: DATA
Tue Nov 27 15:08:09 2018 kern.notice syslog: [IOT-MQTT]: Check for sensor update
Tue Nov 27 15:08:09 2018 kern.notice syslog: [IOT-MQTT]: Found Local channels: 100
Tue Nov 27 15:08:09 2018 kern.notice syslog: [IOT-MQTT]: Find Local channel for 100
Tue Nov 27 15:08:09 2018 kern.notice syslog: [IOT-MQTT]: [ - ] v1/e74b78d9-3858-11e7-afce-8d5fd2a310a7/things/2b1fab30-3859-11e7-afce-8d5fd2a310a7/data/0
Tue Nov 27 15:08:09 2018 kern.notice syslog: [IOT-MQTT]: [ -m] temp,c=30.2
root@dragino-193a18:#

```

4.5.2 Configure LG01-N's Radio frequency

Now we should configure LG01-N's radio parameter to receive the LoRaWAN packets. We are using 868.0Mhz (868000000 Hz) as below:

dragino-1893c4 Status System Network Service Logout

Latitude: 22.73
Longitude: 114.23
Radio Power (Unit dBm): range 5 ~ 20 dBm

Radio Settings

Radio settings for Channel

Frequency (Unit Hz): 868000000 1

Spreading Factor: SF7 2

Coding Rate: 4/5

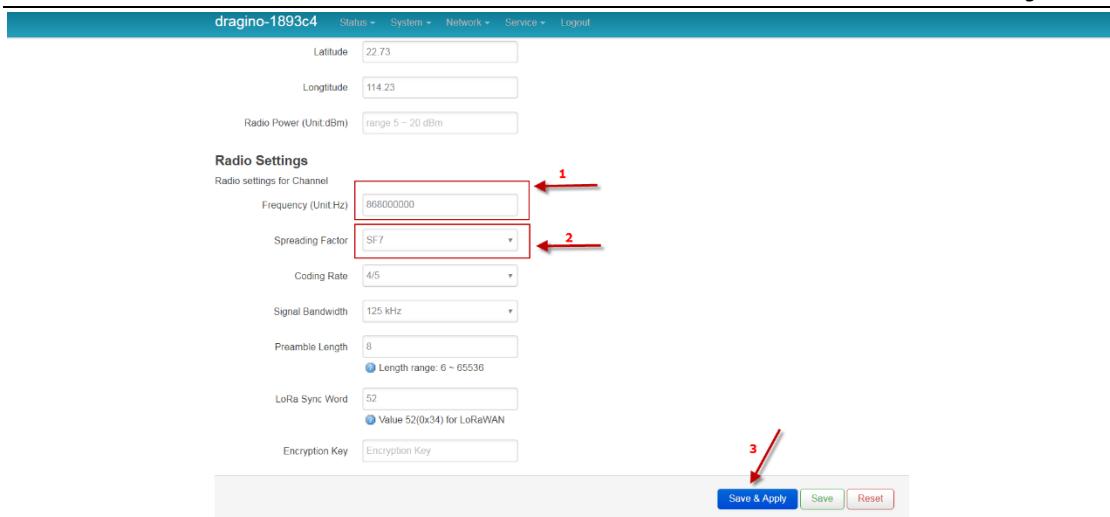
Signal Bandwidth: 125 kHz

Preamble Length: 8
Length range: 6 ~ 65536

LoRa Sync Word: 52
Value 52(0x34) for LoRaWAN

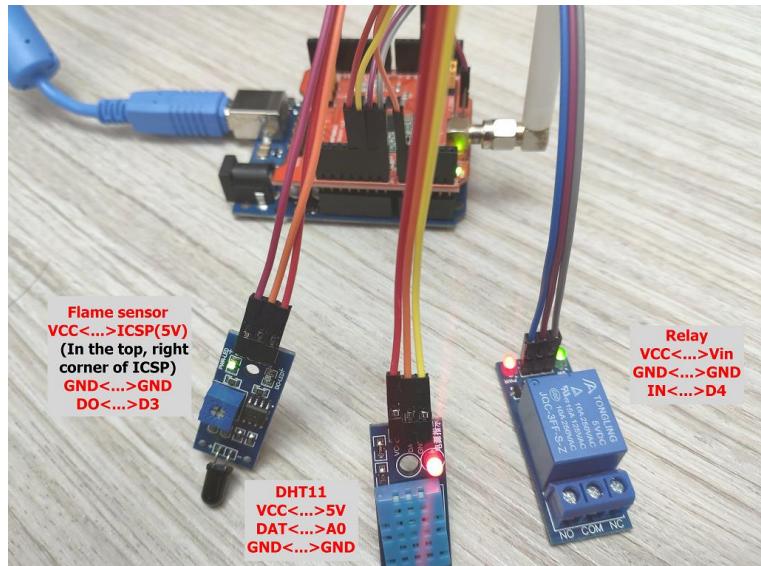
Encryption Key:

3 Save & Apply | Save | Reset



4.6 Create LoRa Shield End Node

4.6.1 Hardware Connection



There are three sensors connect to the LoRa Shield + UNO. These sensors are flame sensors, DHT11 (Temperature & Humidity sensor) and Relay. Please use the connection as we show in the photo.

Note: There is a trick above, the relay is connected to VIN. In this case, The UNO can only be power via USB port. If need to power via DC power adapter, please use another 5v pin to power relay.

Upload [this sketch](#) to the UNO, this sketch will send temperature and humidity data to gateway at every 60 seconds. If there is a flame detect, it will then immediately send the value to gateway and then upload to the IoT Server.

4.6.2 Test with uplink

After we upload the sketch to UNO, we can see below output from Arduino

The screenshot shows the Arduino IDE interface with two main sections: the code editor and the串行监视器 (Serial Monitor) window.

Code Editor:

```
MQTT_DHT11_and_Flame_sensor_Client_update_to_ThingSpeak
文件 软件 项目 工具 帮助
MQTT_DHT11_and_Flame_sensor_Client_update_to_ThingSpeak_
1 #include <dht.h>
2 #include <SPI.h>
3 #include <LoRa.h>
4
5 // Singleton instance of t
6
7 dht DHT;
8 #define DHT11_PIN A0
9 const int flame_pin=3; /
10 float temperature, humidity
11 char tem_1[8]={"\0"} hum_1
```

Serial Monitor:

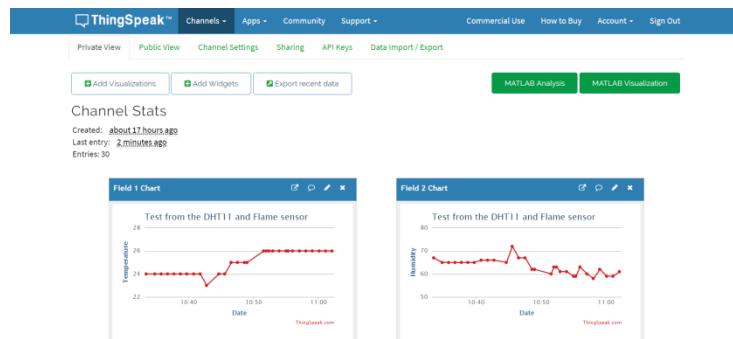
```
COM12 (Arduino/Genuino Uno)
The temperature and humidity:
[24.00°C, 65.00%]
The packet is send successful
#####
COUNT=4 #####
The temperature and humidity:
[24.00°C, 65.00%]
The packet is send successful
#####
COUNT=5 #####
The temperature and humidity:
[24.00°C, 65.00%]
The packet is send successful
#####
COUNT=6 #####
The temperature and humidity:
[24.00°C, 65.00%]
The packet is send successful
```

在Serial Monitor中，第5行和第10行的输出被红色方框包围，表示它们是通过LoRa模块发送的数据包。

And we can see the logread of gateway as below, means the packet arrive gateway:

10.130.2.125 - SecureCRT
文件(F) 编辑(E) 查看(V) 选项(O) 传输(T) 脚本(S) 工具(L) 帮助(H)
10.130.2.125
23d36312e30
Sun Jun 20 02:52:38 2019 user.notice root: [iot.MQTT]: Find Match Entry for 5678
Sun Jun 20 02:52:38 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVDK116n993MKS
Sun Jun 20 02:52:38 2019 user.notice root: [iot.MQTT]: [-] FieldId=26,0xFieldID=61,0xStatus=<QTTPUBLISH
Sun Jun 20 02:52:38 2019 daemon.info lsgol_pk_t_fwd[26279]: RXTRA_receive(HEX):3c3531637381e669656c641313d32362e30266669656c643
Sun Jun 20 02:53:37 2019 user.notice root: [iot.MQTT]: Find Match Entry for 5678
Sun Jun 20 02:53:37 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVKC16n993MKS
Sun Jun 20 02:53:38 2019 user.notice root: [iot.MQTT]: [-] FieldId=26,0xFieldID=61,0xStatus=<QTTPUBLISH
Sun Jun 20 02:54:57 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVKC16n993MKS
Sun Jun 20 02:54:58 2019 user.notice root: [iot.MQTT]: Find Match Entry for 5678
Sun Jun 20 02:54:58 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVKC16n993MKS
Sun Jun 20 02:54:59 2019 user.notice root: [iot.MQTT]: [-] FieldId=26,0xFieldID=59,0xStatus=<QTTPUBLISH
Sun Jun 20 02:55:22 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVKC16n993MKS
Sun Jun 20 02:55:22 2019 user.notice root: [iot.MQTT]: [-] FieldId=26,0xFieldID=61,0xStatus=<QTTPUBLISH
Sun Jun 20 02:55:38 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVKC16n993MKS
Sun Jun 20 02:55:38 2019 user.notice root: [iot.MQTT]: [-] FieldId=26,0xFieldID=61,0xStatus=<QTTPUBLISH
Sun Jun 20 02:56:48 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVKC16n993MKS
Sun Jun 20 02:56:48 2019 user.notice root: [iot.MQTT]: [-] FieldId=26,0xFieldID=61,0xStatus=<QTTPUBLISH
Sun Jun 20 02:57:04 2019 user.notice root: [iot.MQTT]: Find Match Entry for 5678
Sun Jun 20 02:57:04 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVKC16n993MKS
Sun Jun 20 02:57:04 2019 user.notice root: [iot.MQTT]: [-] FieldId=26,0xFieldID=61,0xStatus=<QTTPUBLISH
Sun Jun 20 02:57:58 2019 user.notice root: [iot.MQTT]: RXTRA_receive(HEX):3c3531637381e669656c641313d32362e30266669656c643
23d35329e230
Sun Jun 20 02:55:58 2019 user.notice root: [iot.MQTT]: Find Match Entry for 5678
Sun Jun 20 02:55:58 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVKC16n993MKS
Sun Jun 20 02:55:58 2019 user.notice root: [iot.MQTT]: [-] FieldId=26,0xFieldID=61,0xStatus=<QTTPUBLISH
Sun Jun 20 02:56:48 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVKC16n993MKS
Sun Jun 20 02:56:48 2019 user.notice root: [iot.MQTT]: [-] FieldId=26,0xFieldID=61,0xStatus=<QTTPUBLISH
Sun Jun 20 02:57:58 2019 user.notice root: [iot.MQTT]: RXTRA_receive(HEX):3c3531637381e669656c641313d32362e30266669656c643
23d40302e230
Sun Jun 20 02:57:58 2019 user.notice root: [iot.MQTT]: Find Match Entry for 5678
Sun Jun 20 02:57:58 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVKC16n993MKS
Sun Jun 20 02:57:58 2019 user.notice root: [iot.MQTT]: [-] FieldId=26,0xFieldID=59,0xStatus=<QTTPUBLISH
Sun Jun 20 02:58:04 2019 user.notice root: [iot.MQTT]: RXTRA_receive(HEX):3c3531637381e669656c641313d32362e30266669656c643
23d35329e230
Sun Jun 20 02:58:04 2019 user.notice root: [iot.MQTT]: Find Match Entry for 5678
Sun Jun 20 02:58:04 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVKC16n993MKS
Sun Jun 20 02:58:04 2019 user.notice root: [iot.MQTT]: [-] FieldId=26,0xFieldID=61,0xStatus=<QTTPUBLISH
Sun Jun 20 02:59:04 2019 user.notice root: [iot.MQTT]: RXTRA_receive(HEX):3c3531637381e669656c641313d32362e30266669656c643
Sun Jun 20 02:59:04 2019 user.notice root: [iot.MQTT]: Find Match Entry for 5678
Sun Jun 20 02:59:04 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVKC16n993MKS
Sun Jun 20 02:59:04 2019 user.notice root: [iot.MQTT]: [-] FieldId=26,0xFieldID=61,0xStatus=<QTTPUBLISH
Sun Jun 20 02:59:58 2019 user.notice root: [iot.MQTT]: RXTRA_receive(HEX):3c3531637381e669656c641313d32362e30266669656c643
Sun Jun 20 02:59:58 2019 user.notice root: [iot.MQTT]: Find Match Entry for 5678
Sun Jun 20 02:59:58 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVKC16n993MKS
Sun Jun 20 02:59:58 2019 user.notice root: [iot.MQTT]: [-] FieldId=26,0xFieldID=61,0xStatus=<QTTPUBLISH
Sun Jun 20 03:00:04 2019 user.notice root: [iot.MQTT]: RXTRA_receive(HEX):3c3531637381e669656c641313d32362e30266669656c643
Sun Jun 20 03:00:04 2019 user.notice root: [iot.MQTT]: Find Match Entry for 5678
Sun Jun 20 03:00:04 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVKC16n993MKS
Sun Jun 20 03:00:04 2019 user.notice root: [iot.MQTT]: [-] FieldId=26,0xFieldID=61,0xStatus=<QTTPUBLISH
Sun Jun 20 03:00:58 2019 user.notice root: [iot.MQTT]: RXTRA_receive(HEX):3c3531637381e669656c641313d32362e30266669656c643
Sun Jun 20 03:01:04 2019 user.notice root: [iot.MQTT]: Find Match Entry for 5678
Sun Jun 20 03:01:04 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVKC16n993MKS
Sun Jun 20 03:01:04 2019 user.notice root: [iot.MQTT]: [-] FieldId=26,0xFieldID=61,0xStatus=<QTTPUBLISH
Sun Jun 20 03:01:58 2019 user.notice root: [iot.MQTT]: RXTRA_receive(HEX):3c3531637381e669656c641313d32362e30266669656c643
Sun Jun 20 03:01:58 2019 user.notice root: [iot.MQTT]: Find Match Entry for 5678
Sun Jun 20 03:01:58 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVKC16n993MKS
Sun Jun 20 03:01:58 2019 user.notice root: [iot.MQTT]: [-] FieldId=26,0xFieldID=61,0xStatus=<QTTPUBLISH
Sun Jun 20 03:02:04 2019 user.notice root: [iot.MQTT]: RXTRA_receive(HEX):3c3531637381e669656c641313d32362e30266669656c643
Sun Jun 20 03:02:04 2019 user.notice root: [iot.MQTT]: Find Match Entry for 5678
Sun Jun 20 03:02:04 2019 user.notice root: [iot.MQTT]: [-] channels/_/682338/publish/EVKC16n993MKS
Sun Jun 20 03:02:04 2019 user.notice root: [iot.MQTT]: [-] FieldId=26,0xFieldID=61,0xStatus=<QTTPUBLISH

Finally, we can see on the ThingSpeak:



4.6.3 Test with interrupt by flame detect

The DO pin of Flame sensor is high in normal state. When a flame is detected, the DO pin of Flame sensor will become low, then, the UNO generates an external interrupt, and immediately uploads the temperature and humidity to the server.

The DO pin of Flame sensor is low when a flame is detected, and we can see on the Serial Monitor:

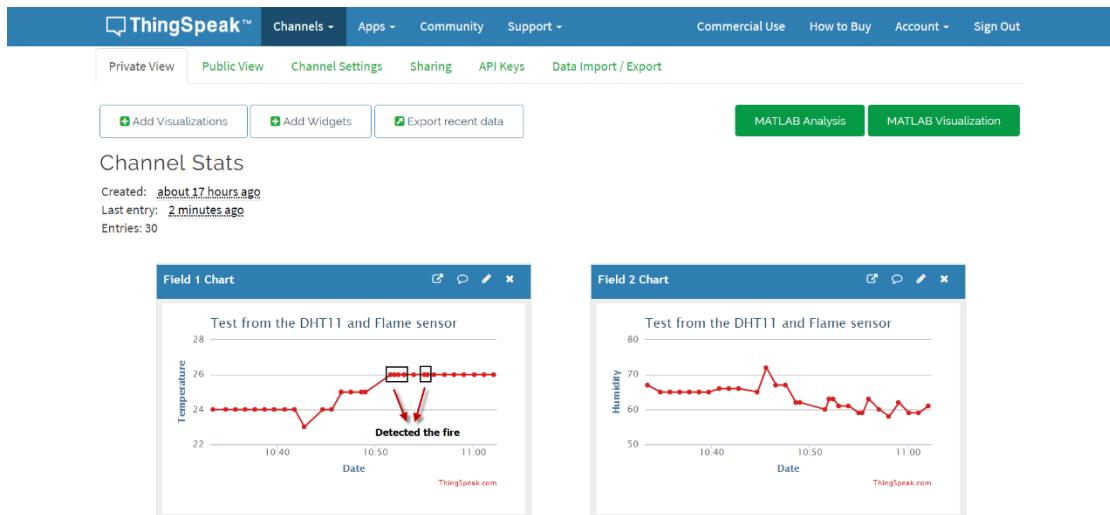
The screenshot shows the Arduino IDE with two windows open. The left window displays the code for the sketch, which includes functions for handling fire detection, reading DHT11 sensor data, and sending data via LoRaWAN to ThingSpeak. The right window is the Serial Monitor, showing the transmitted data frames. Each frame consists of a header ('##### COUNT=2 #####'), followed by the message 'The temperature and humidity:', the data '[26.00°C, 61.00%]', and a footer ('##### COUNT=3 #####'). This pattern repeats for COUNT=3, COUNT=4, and COUNT=5, with the last frame also containing the message 'Have fire, the temperature is send'.

```
MQTT_DHT11_and_Flame_sensor_Client_update_to_ThingSpeak.ino
```

```
## COUNT=2 #####
The temperature and humidity:
[26.00°C, 61.00%]
The packet is send successful
#####
COUNT=3 #####
The temperature and humidity:
[26.00°C, 61.00%]
The packet is send successful
#####
COUNT=4 #####
The temperature and humidity:
[26.00°C, 59.00%]
The packet is send successful
#####
Have fire, the temperature is send
The temperature and humidity:
[26.00°C, 59.00%]
```

Similarly, we can see the logread of gateway via SSH access:

Finally, we can see on the ThingSpeak:

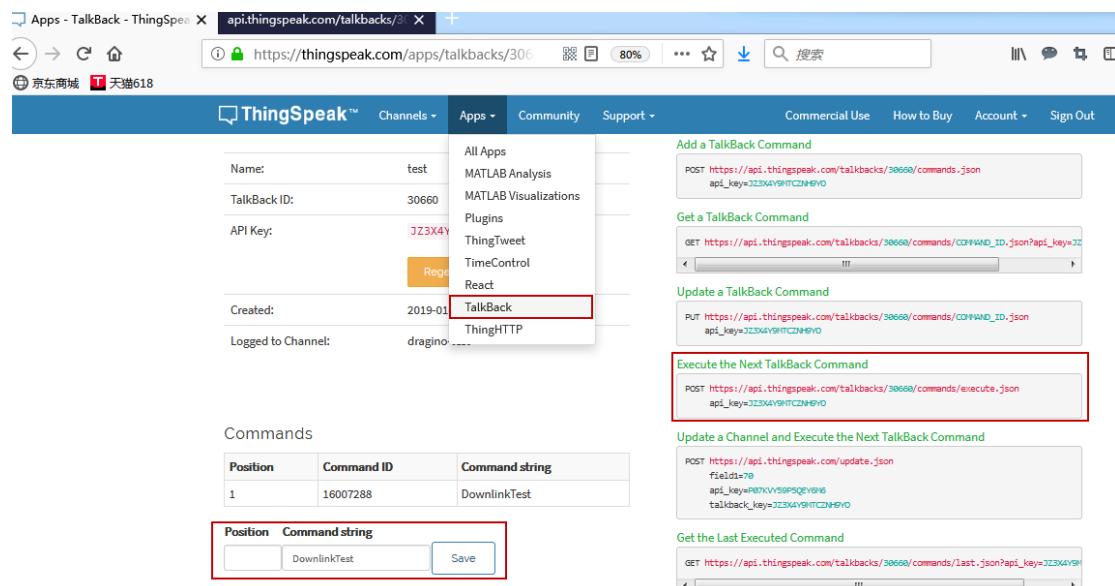


4.6.4 Test with downlink

The http downlink feature is now support since firmware LG02_LG08--build--v5.2.1560931576--20190619-1607.

ThingSpeak downlink command can be found in TalkBack App.

The **Command String input box** is the command you want to send to LoRa device.



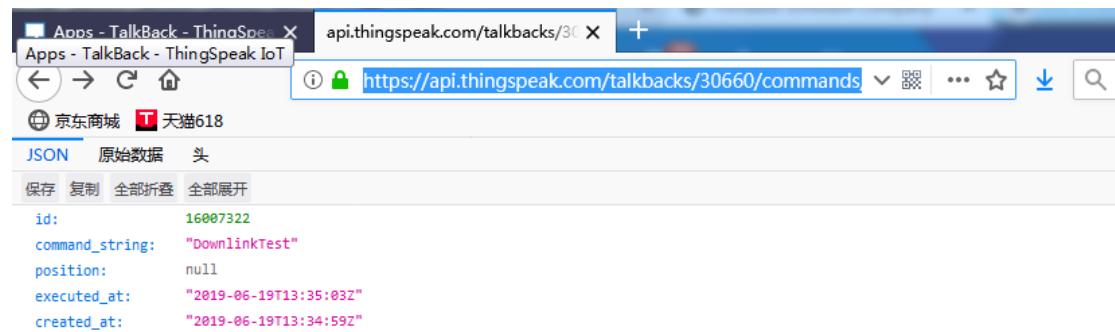
The screenshot shows the ThingSpeak TalkBack app interface. In the top navigation bar, there is a dropdown menu labeled 'TalkBack'. A red box highlights the 'TalkBack' option in this menu. Below the navigation bar, there is a table titled 'Commands' with one row containing 'Position' 1, 'Command ID' 16007288, and 'Command string' DownlinkTest. At the bottom of the page, there is a form with a 'Position' field containing '1', a 'Command string' field containing 'DownlinkTest', and a 'Save' button. The 'Save' button is also highlighted with a red box.

Execute The next Talkback Command is the API to get one command from the commands queue.

We can test in the web with this API. Format is:

https://api.thingspeak.com/talkbacks/XXXXXX/commands/execute.json?api_key=XXXXXXXXXX

Result as below:



The screenshot shows a browser window displaying a JSON response. The response is as follows:

```

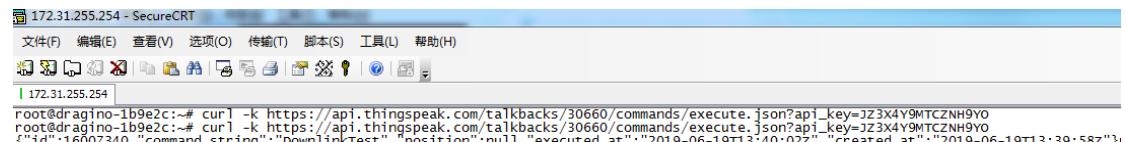
{
  "id": 16007322,
  "command_string": "DownlinkTest",
  "position": null,
  "executed_at": "2019-06-19T13:35:03Z",
  "created_at": "2019-06-19T13:34:59Z"
}

```

We can also test this API in LG01-N Linux console:

By using:

`curl -k https://api.thingspeak.com/talkbacks/XXXXXX/commands/execute.json?api_key=XXXXXXXXXX`

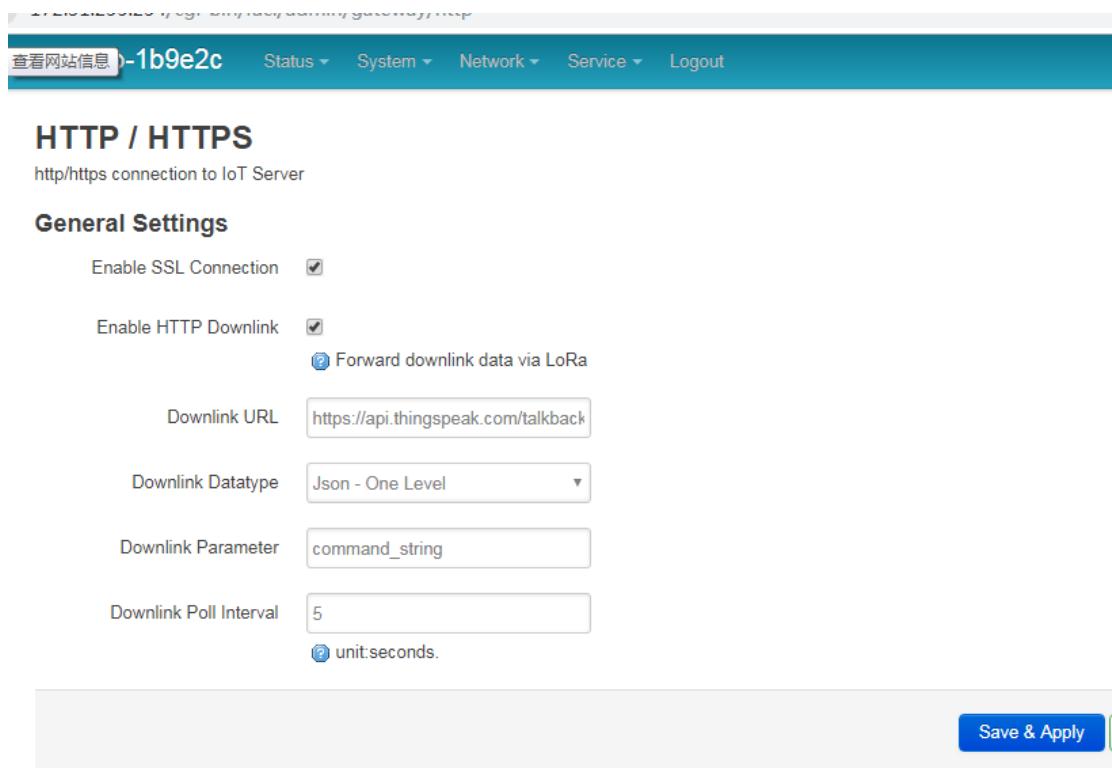


```

root@dragino-1b9e2c:~# curl -k https://api.thingspeak.com/talkbacks/30660/commands/execute.json?api_key=j23x4y9mtcznh9yo
root@dragino-1b9e2c:~# curl -k https://api.thingspeak.com/talkbacks/30660/commands/execute.json?api_key=j23x4y9mtcznh9yo
{"id":16007340,"command_string":"DownlinkTest","position":null,"executed_at":"2019-06-19T13:40:02Z","created_at":"2019-06-19T13:39:58Z"}

```

To get this result automatically in LG01-P and send out via LoRa, we can configure as below:



HTTP / HTTPS

http/https connection to IoT Server

General Settings

Enable SSL Connection

Enable HTTP Downlink Forward downlink data via LoRa

Downlink URL

Downlink Datatype

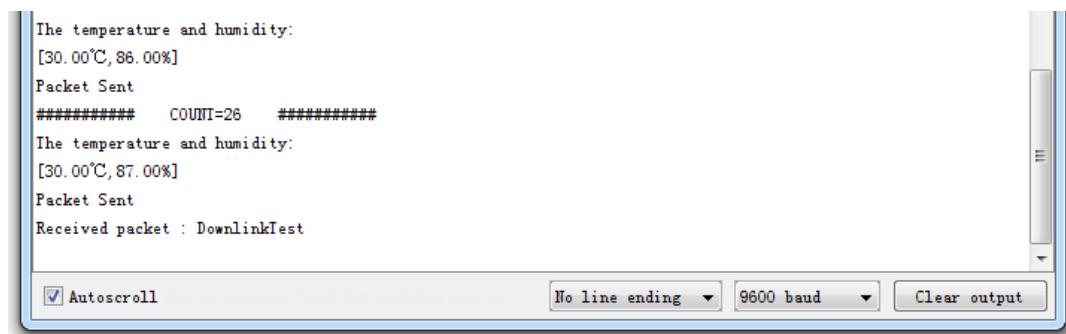
Downlink Parameter

Downlink Poll Interval unit:seconds.

Save & Apply

- Because URL is https, So need to Enable SSL Connection
- Downlink URL use the URL we use in Web and Curl
- Downlink datatype for ThingSpeak is Json.
- Downlink Parameter is command_string. We will fetch the value of `command_string` from the downlink data string.
- LG01-N will poll the URL every 5 seconds. When there is valid command_string found, it will send out via LoRa (Radio parameter is defined in LoRaWAN gateway Radio settings or Radio2 settings for LG02)

Result in the LoRa Shield:



```
The temperature and humidity:  
[30.00°C, 86.00%]  
Packet Sent  
##### COUNT=26 #####  
The temperature and humidity:  
[30.00°C, 87.00%]  
Packet Sent  
Received packet : DownlinkTest
```

Autoscroll No line ending 9600 baud Clear output

4.7 Conclusion and limitation

4.7.1 Overview for the example

This example shows how to set up a simple LoRa network with ThingSpeak IoT server. In this example, we use the raw LoRa protocol (private protocol) for transmission. It is simpler compare via LoRaWAN protocol

There are some frequently ask points for the example:

1/ Difference between LoRaWAN & Private LoRa protocol:

- The private LoRa protocol here doesn't have MAC control/management, (of course developer can develop this). In LoRaWAN protocol, this feature is supported already.
- The transmission is unencrypted in this example, user can see the data in gateway. In LoRaWAN, the transmission is designed in AES encryption.
- Private LoRa protocol means the gateway only works with specify LoRa End node which runs the same protocol, the gateway can't work with a standard LoRaWAN devices.
- Private LoRa protocol doesn't need the LoRaWAN IoT Server. Gateway can send data to user defined IoT server, in terms the gateway and the server can communicate with each other.
- User can more features in the private protocol such as MAC control, encryption, that is how LoRaWAN protocol comes, the advantage of LoRaWAN protocol is that it is designed for carrier level use, and developer can use it directly with many features and compatible with the LoRaWAN node from different manufacturers.

5 Order Info

LoRa_IoT_Kit-v2-XXX-YYY

XXX: Frequency Band

433: For Bands: EU433, CN470

868: For Bands: EU868, IN865

915: For Bands: US915, AU915, AS923, KR920

YYY: 4G Cellular Option

EC25-E: EMEA, Korea, Thailand, India.

EC25-A: North America/ Rogers/AT&T/T-Mobile.

EC25-AU: Latin America, New Zeland, Taiwan

EC25-J: Japan, DOCOMO/SoftBank/ KDDI

More info about valid bands, please see EC25-E product page

(<https://www.quectel.com/product/ec25.htm>)

6 FAQ & Trouble Shooting

6.1 I can't upload sketch to LoRa Shield in MAC OS, shows "dev/cu.usbmodem1421 is not available"

Error Info as below:

```
Arduino: 1.8.3 (Mac OS X), Board: "Arduino/Genuino Uno"
Archiving built core (caching) in:
/var/folders/jq/8fnvlfj90tgbnbcy whole0gn/T/arduino_cache_833512/core/core_arduino
_avr_arduino_fc9a32205aafa27e4eda988d5ed9b7ac.a
Sketch uses 20142 bytes (62%) of program storage space. Maximum is 32256 bytes.
Global variables use 1189 bytes (58%) of dynamic memory, leaving 859 bytes for local variables.
Maximum is 2048 bytes.
Board at /dev/cu.usbmodem1421 is not available
```

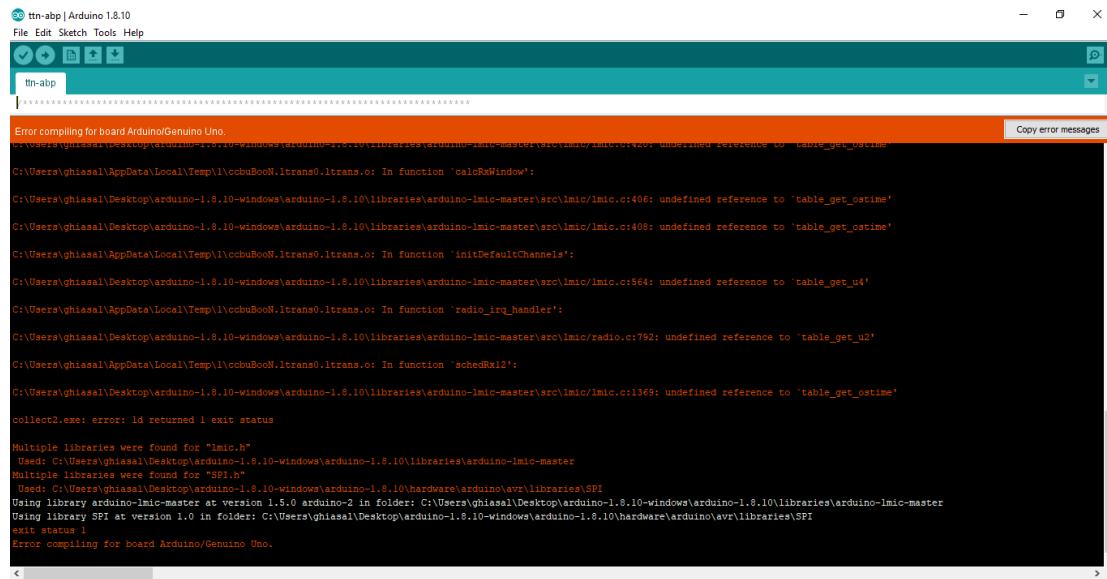
The Arduino UNOs in the Kit are clone version and use CH340 USB to serial chip. User has to install the CH340 driver in PC to make it work. Above issue means the MAC OS doesn't have CH340 driver.

6.2 My IoT Kit has the model LG01-P instead of LG01-N, Can I still use this manual.

The gateway part of this manual is for LG01-N, if user has the LG01-P version, please check the [LG01-P gateway manual](#).

6.3 Duplicate library issue while upload in Arduino IDE 1.8.10.

While compile the LMIC library in Arduino IDE 1.8.10. This error will happen:



```
ttn-abp | Arduino 1.8.10
File Edit Sketch Tools Help
ttn-abp
Error compiling for board Arduino/Genuino Uno.

C:\Users\ghiasel\AppData\Local\Temp\ccbUBooN.ltrans0.ltrans.o: In function `calcRxWindow':
C:\Users\ghiasel\Desktop\arduino-1.8.10-windows\arduino-lmic-master\src\lmic\lmic.c:406: undefined reference to `table_get_ostime'
C:\Users\ghiasel\Desktop\arduino-1.8.10-windows\arduino-lmic-master\src\lmic\lmic.c:408: undefined reference to `table_get_ostime'
C:\Users\ghiasel\AppData\Local\Temp\ccbUBooN.ltrans0.ltrans.o: In function `initDefaultChannels':
C:\Users\ghiasel\Desktop\arduino-1.8.10-windows\arduino-lmic-master\src\lmic\lmic.c:564: undefined reference to `table_get_u4'
C:\Users\ghiasel\AppData\Local\Temp\ccbUBooN.ltrans0.ltrans.o: In function `radio_irq_handler':
C:\Users\ghiasel\Desktop\arduino-1.8.10-windows\arduino-lmic-master\src\lmic\radio.c:792: undefined reference to `table_get_u2'
C:\Users\ghiasel\AppData\Local\Temp\ccbUBooN.ltrans0.ltrans.o: In function `schedRx12':
C:\Users\ghiasel\Desktop\arduino-1.8.10-windows\arduino-lmic-master\src\lmic\lmic.c:1369: undefined reference to `table_get_ostime'
collect2.exe: error: ld returned 1 exit status

Multiple libraries were found for "lmic.h"
  Used: C:\Users\ghiasel\Desktop\arduino-1.8.10-windows\arduino-1.8.10\libraries\arduino-lmic-master
Multiple libraries were found for "SPI.h"
  Used: C:\Users\ghiasel\Desktop\arduino-1.8.10-windows\arduino-1.8.10\hardware\arduino\avr\libraries\SPI
Using library arduino-lmic-master at version 1.5.0 arduino-2 in folder: C:\Users\ghiasel\Desktop\arduino-1.8.10-windows\arduino-1.8.10\libraries\arduino-lmic-master
Using library SPI at version 1.0 in folder: C:\Users\ghiasel\Desktop\arduino-1.8.10-windows\arduino-1.8.10\hardware\arduino\avr\libraries\SPI
exit status 1
Error compiling for board Arduino/Genuino Uno.
```

To solve this, user can modify the file:

`~/Dev/Arduino.app/Contents/Java/hardware/arduino/avr/platform.txt`

, Change the compile flag from `-Os` to `-O2`. Like below:

```
compiler.warning_flags.more=-Wall
compiler.warning_flags.all=-Wall -Wextra

// Default "compiler.path" is correct, change only if you want to override the initial value
compiler.path={runtime.tools.avr-gcc.path}/bin/
compiler.c.cmd=avr-gcc
compiler.c.flags=-c -g -O2 {compiler.warning_flags} -std=gnu11 -ffunction-sections -fdata-sections -MMD -fno-fat-lto-objects
compiler.c.elf.flags={compiler.warning_flags} -O2 -g -fno-fat-lto -fuse-linker-plugin -Wl,--gc-sections
compiler.c.elf.cmd=avr-gcc
compiler.S.flags=-c -g -x assembler-with-cpp -fno-fat-lto -MMD
compiler.cpp.cmd=avr-g++
compiler.cpp.flags=-c -g -O2 {compiler.warning_flags} -std=gnu++11 -fpermissive -fno-exceptions -ffunction-sections -fdata-sections -fno-threadsafe
  cs -Wno-error=narrowing -MMD -fno-fat-lto
compiler.ar.cmd=avr-gcc-ar
compiler.ar.flags=rCS
compiler.objcopy.cmd=avr-objcopy
compiler.objcopy.eep.flags=-O ihex -j .eeprom --set-section-flags=.eeprom=alloc,load --no-change-warnings --change-section-lma .eeprom=0
compilerElf2hex.flags=-O ihex -R .eeprom
```

6.4 How can I set to use CN470 band?

```
11 // #define CFG_us921 1
12 // #define CFG_as923 1
13 // #define CFG_in866 1
14
15 #define LG02_LG01 1
16
17 //US915: DR_SF10=0, DR_SF9=1, DR_SF8=2, DR_SF7=3, DR_SF8C=4
18 //          DR_SF12CR=8, DR_SF11CR=9, DR_SF10CR=10, DR_SF9CR=11, DR_SF8CR=12, DR_SF7CR
19 #if defined(CFG_us915) && defined(LG02_LG01)
20 // CFG_us915 || CFG_as923
21 #define LG02_UPFREQ 902320000
22 #define LG02_DNWREQ 923300000
23 #define LG02_RXSF 3      // DR_SF7 For LG01/LG02 Tx
24 #define LG02_TXSF 8      // DR_SF12CR For LG02/LG02 Rx
25 #elif defined(CFG_eu868) && defined(LG02_LG01)
26 // CFG_eu868
27 //EU868: DR_SF12=0, DR_SF11=1, DR_SF10=2, DR_SF9=3, DR_SF8=4, DR_SF7=5, DR_SF7B=1, DR_FSK, DR
28 #define LG02_UPFREQ 505300000
29 #define LG02_DNWREQ 505300000
30 #define LG02_RXSF 0      // DR_SF7 For LG01/LG02 Tx
31 #define LG02_TXSF 0      // DR_SF12 For LG02/LG02 Rx
32#endif
33
```

7 Technical Support

- Support is provided Monday to Friday, from 09:00 to 18:00 GMT+8. Due to different timezones we cannot offer live support. However, your questions will be answered as soon as possible in the before-mentioned schedule.
- Provide as much information as possible regarding your enquiry (product models, accurately describe your problem and steps to replicate it etc) and send a mail to

support@dragino.com

8 Reference

- 1) [LoRaWAN official website. And Technical document for LoRaWAN.](#)
- 2) [LG01-N LoRa Gateway User Manual](#)
- 3) [LoRa Low Energy design guide](#) and [Calculator Tool](#).
- 4) About Distance: [LoRa Modem Design Guide](#)
- 5) [SX1276 download resource.](#)
- 6) User Manual: [LG01-N](#), [LoRa Shield](#), [LoRa/GPS Shield](#)