

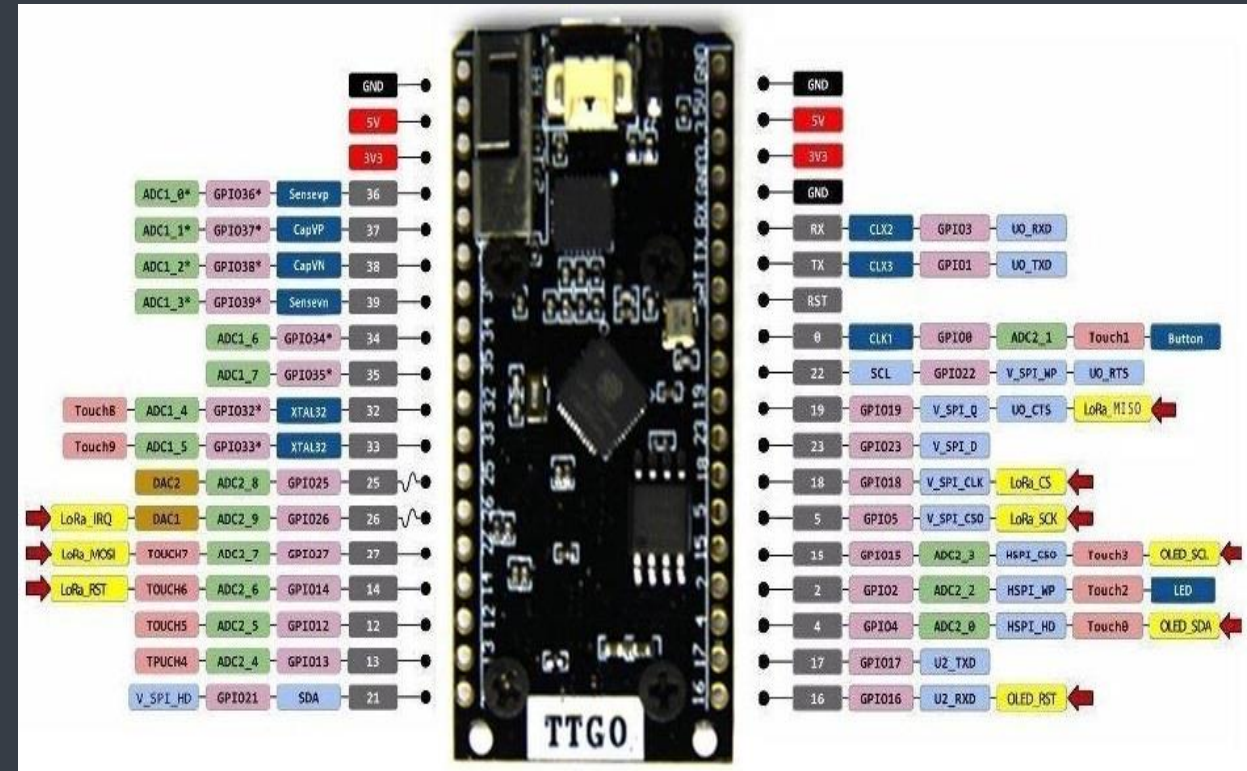
TTGO LORA32 V1 AS NODE FOR LORAWAN SERVER



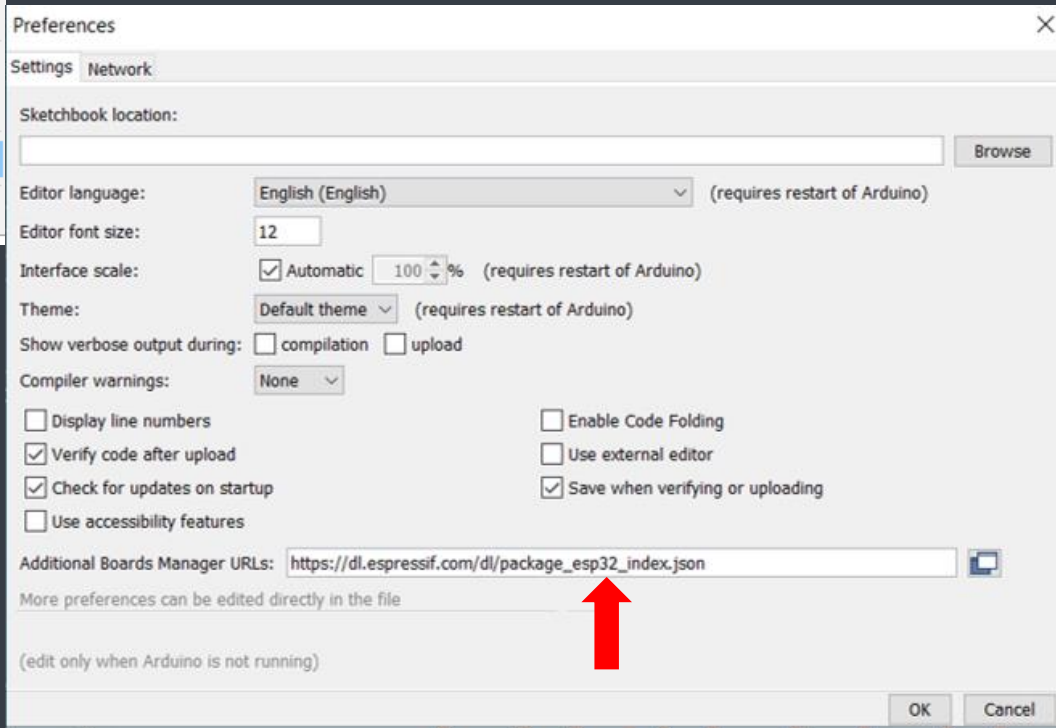
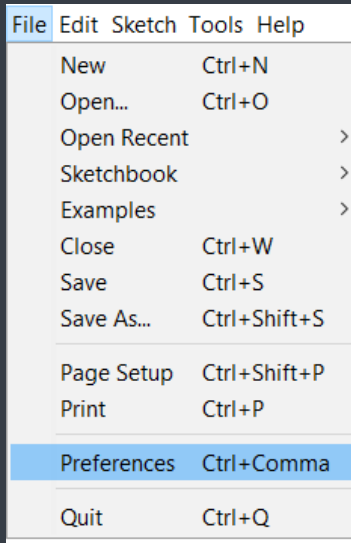
INTRODUCTION

This presentation contains the instructions for connecting a TTGO LoRa32 V1 module to a LoraWan-Server and transmitting data from a sensor (in this example temperature and humidity) into the server. In order to carry out this example the following hardware and software will be needed:

- Arduino IDE environment
- Arduino-LMIC library
- CayenneLPP library
- Dht12 library
- TTGO LoRa32 V1
- ESP32 firmware
- ChirpStack Server
- M5Stack ENV Unit with Temperature Humidity Pressure Sensor
- USB to MicroUSB cable
- Wire (for connecting the sensor to the ESP32)



ESP32 FIRMWARE



- In order to add the ESP32 firmware open the menu “**File**”, select then “**preferences**” and a window will pop up. Add the following URL in the “**Additional Boards Manager URLs**” field and click ok:

https://dl.espressif.com/dl/package_esp32_index.json

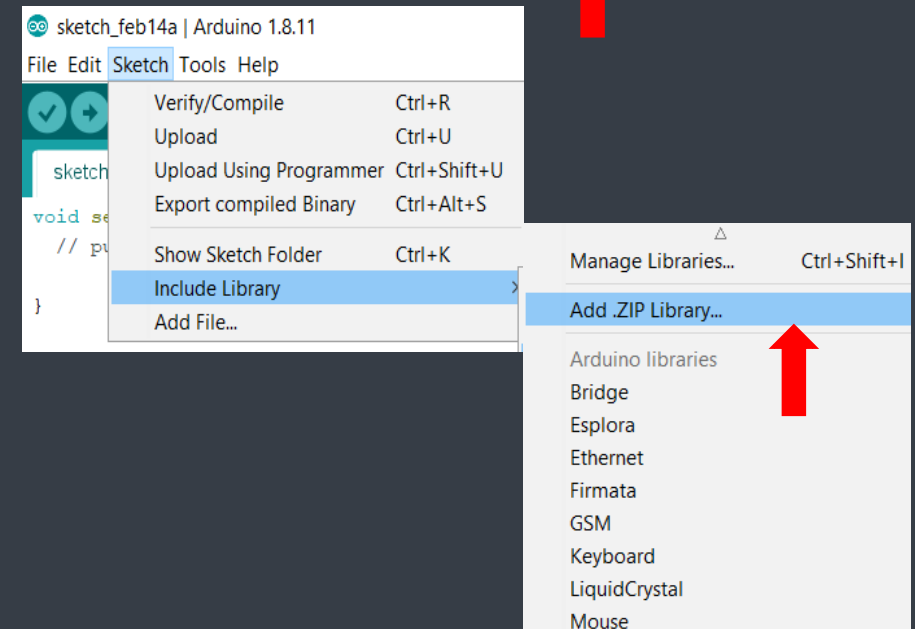
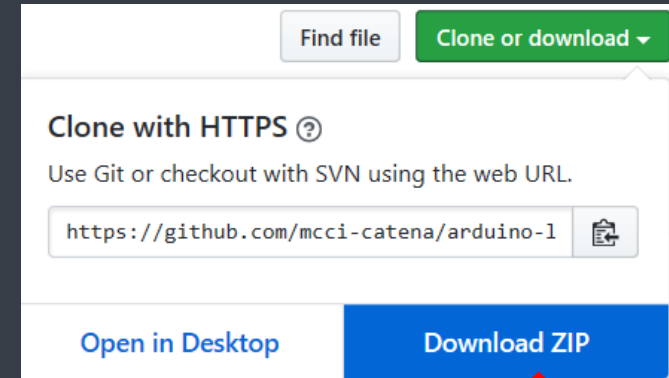
DOWNLOADING THE LIBRARIES

The following links contain the libraries needed for this example:

- **LoRaWAN-MAC-in-C library:** <https://github.com/mcci-catena/arduino-lmic>
- **CayenneLPP library:** <https://github.com/ElectronicCats/CayenneLPP>
- **Dht12 library:** https://github.com/Bobadas/DHT12_library_Arduino

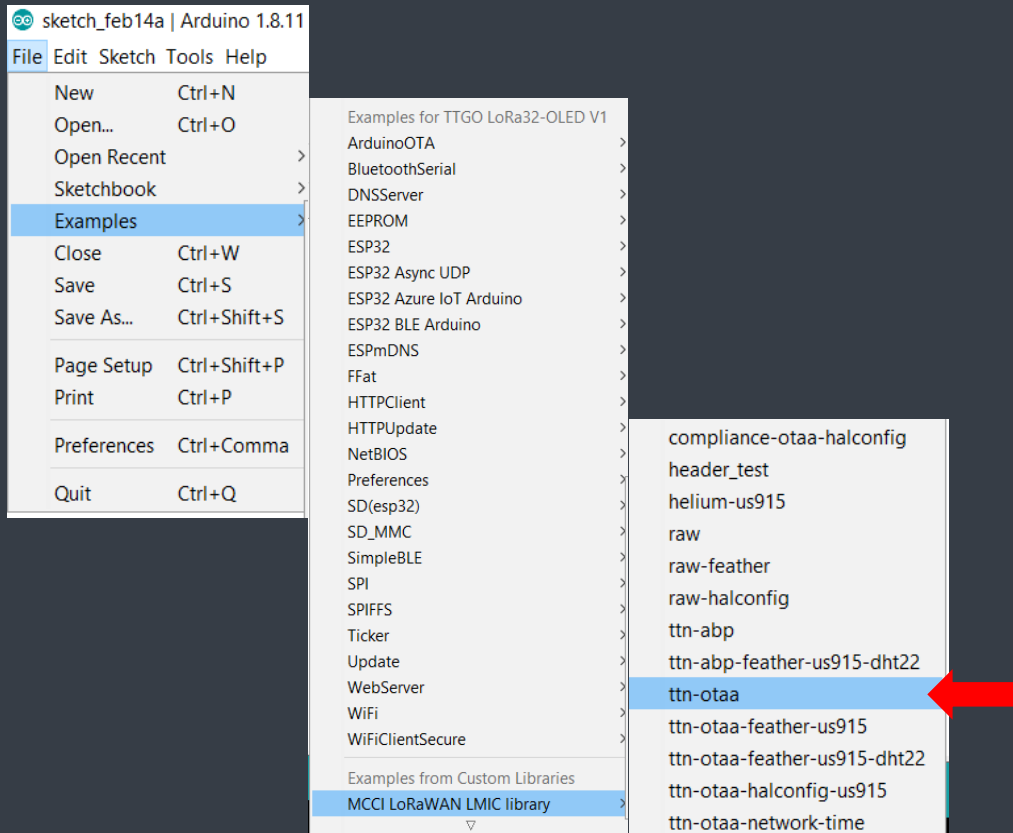
Once you have entered to the Website look for the “**Clone or download**” button, click on it and then select “**Download ZIP**”. Save the ZIP-folders wherever is easy for you to find them (we won’t be needing them for too long).

For the next step, open the Arduino IDE environment and go to the Sketch menu, then select “**Include Library**” and “**Add .ZIP Library**”. Here you will select the ZIP-folders downloaded in the step before and the libraries will be added to the Libraries folder of Arduino (After this step the ZIP-folders won’t be needed anymore).



THE CODE

- Inside the Arduino environment go to the File menu, select “**Examples**”, search for the **MCCI LoRaWAN LMIC library** and then click on the “**ttn-otaa**” example.



- The first three Libraries showed below are already included in the example, the other three need to be added as well as the **CayenneLPP lpp(51)** and **DHT12 dht12** instructions. They are important for the functioning of the sensor and encryption of the data.

```
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include <CayenneLPP.h>
#include <Dht12.h>
#include <Wire.h> //The DHT12 uses I2C communication.
CayenneLPP lpp(51);
DHT12 dht12;      //Preset scale CELSIUS and ID 0x5c.
```

- Note:** The **Wire.h** library is a default library of Arduino, we do not need to download it.

- Next thing that will come up in the program is this chunk of code, it needs to be removed or it will cause problems when trying to compile and upload to the board.

```
//  
// For normal use, we require that you edit the sketch to replace FILLMEIN  
// with values assigned by the TTN console. However, for regression tests,  
// we want to be able to compile these scripts. The regression tests define  
// COMPILE_REGRESSION_TEST, and in that case we define FILLMEIN to a non-  
// working but innocuous value.  
//  
#ifdef COMPILE_REGRESSION_TEST  
# define FILLMEIN 0  
#else  
# warning "You must replace the values marked FILLMEIN with real values from the TTN control panel!"  
# define FILLMEIN (#dont edit this, edit the lines that use FILLMEIN)  
#endif
```

- This part of the program plays a very important role but it requires a further explanation, for now we will leave it as it is.

```
// This EUI must be in little-endian format, so least-significant-byte  
// first. When copying an EUI from ttnctl output, this means to reverse  
// the bytes. For TTN issued EUIs the last bytes should be 0xD5, 0xB3,  
// 0x70.  
static const ul_t PROGMEM APPEUI[8]={ FILLMEIN };  
void os_getArtEui (ul_t* buf) { memcpy_P(buf, APPEUI, 8);}  
  
// This should also be in little endian format, see above.  
static const ul_t PROGMEM DEVEUI[8]={ FILLMEIN };  
void os_getDevEui (ul_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 ttnctl can be copied as-is.  
static const ul_t PROGMEM APPKEY[16] = { FILLMEIN };  
void os_getDevKey (ul_t* buf) { memcpy_P(buf, APPKEY, 16);}
```

- The value of **TX_INTERVAL** can be changed to increase or decrease the time the board will wait to send a payload. The given value represent the amount of seconds.

```
// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 60;
```

- The Pin mapping needs to be changed so that it matches the Pins of the TTGO LoRa32 V1.

PIN Mappings

Heltec Wifi LoRa 32, TTGO LoRa and TTGO LoRa32 V1:

ESP32		LoRa (SPI)	Display (I2C)	LED
GPIO5	SCK	SCK		
GPIO27	MOSI	MOSI		
GPIO19	MISO	MISO		
GPIO18	SS	NSS		
GPIO14		RST		
GPIO26		DIO0		
GPIO33		DIO1		
GPIO32		DIO2		
GPIO15			SCL	
GPIO4			SDA	
GPIO16			RST	
GPIO25				Heltec, TTGO LoRa32
GPIO2				TTGO LoRa

BEFORE

```
// Pin mapping
const lmico_pinmap lmico_pins = {
    .nss = 6,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 5,
    .dio = {2, 3, 4},
};
```



AFTER

```
// Pin mapping
const lmico_pinmap lmico_pins = {
    .nss = 18,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 14,
    .dio = {26, 33, 32},
};
```

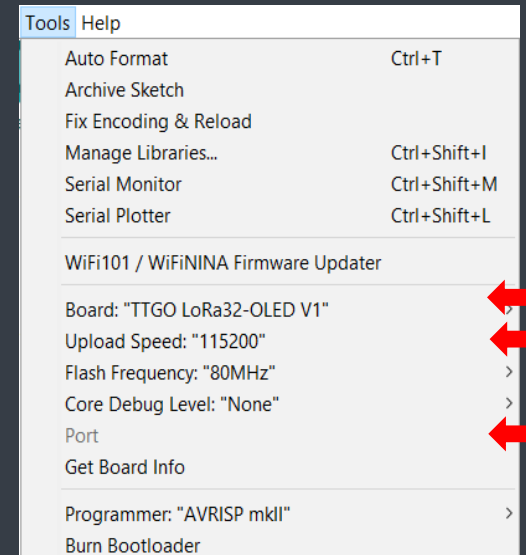
- **OPTIONAL:** These two chunks of code can be deleted or just be ignored, they have no influence on the functioning of the program.

```
/*  
|| This event is defined but not used in the code. No  
|| point in wasting codespace on it.  
||  
|| case EV_RFU1:  
||     Serial.println(F("EV_RFU1"));  
||     break;  
*/
```

```
/*  
|| This event is defined but not used in the code. No  
|| point in wasting codespace on it.  
||  
|| case EV_SCAN_FOUND:  
||     Serial.println(F("EV_SCAN_FOUND"));  
||     break;  
*/
```

- The **Wire.h** library has to be initialized in the **void setup()** with the instruction **Wire.begin()**.
- The value of **Serial.begin()** should correspond to the **upload speed**, we can see this value in the **Tools** menu.
- Inside the Tools menu the **"TTGO LoRa32-OLED V1"** board and the **port** that is being used have to be selected, or the program won't function.

```
void setup() {  
    Serial.begin(115200);  
    Serial.println(F("Starting"));  
  
    Wire.begin();  
    Serial.println(F("Wire begin"));  
}
```



- The original program was written to send “Hello, world!” through LoRa, since we will be sending information from a sensor, the next line of code won’t be needed anymore.

```
static const ul_t PROGMEM APPKEY[16] = { FILLMEIN };  
void os_getDevKey (ul_t* buf) { memcpy_P(buf, APPKEY, 16);}  
  
static uint8_t mydata[] = "Hello, world!";  
  
static osjob_t sendjob;
```

- The **void do_send(osjob_t* j)** function will contain the code and data wished to send. Start by writing inside of the else brackets, the following function from the Cayenne library and the temperature and humidity readings (which will be stored into variables).

```
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 {  
        lpp.reset();  
        float temp = dht12.readTemperature();  
        float hum = dht12.readHumidity();  
    }  
}
```

- Write then the following lines of code to print the values of temperature and humidity into the serial monitor, encrypt the values and set the data that will be sent by the board.

```
Serial.printf("T:%2.2f°C H:%0.2f%%\n", temp, hum);  
lpp.addTemperature(1, temp);  
lpp.addRelativeHumidity(2, hum);  
LMIC_setTxData2(1, lpp.getBuffer(), lpp.getSize(), 0);  
Serial.println(F("Packet queued"));  
}  
// Next TX is scheduled after TX_COMPLETE event.  
}
```

BAND AND VERSION CONFIGURATION

- In the **Libraries** folder of Arduino, look for the “**Arduino-lmic-master**” folder. Inside of it you will find another folder called “**project_config**” that contains the file “**lmic_project_config.h**”. Open and edit this file so that it contains what it is shown in the picture.
- This is the configuration for choosing a LoRaWAN version, defining the region and frequency the program will be working with and selecting the transceiver our ESP32 module possesses.
- Our program is configured to work with the designated frequency for Europe (868MHz), it uses the LoRaWAN version 1.0.2 and it has selected the sx1276 radio transceiver.

C lmic_project_config.h X

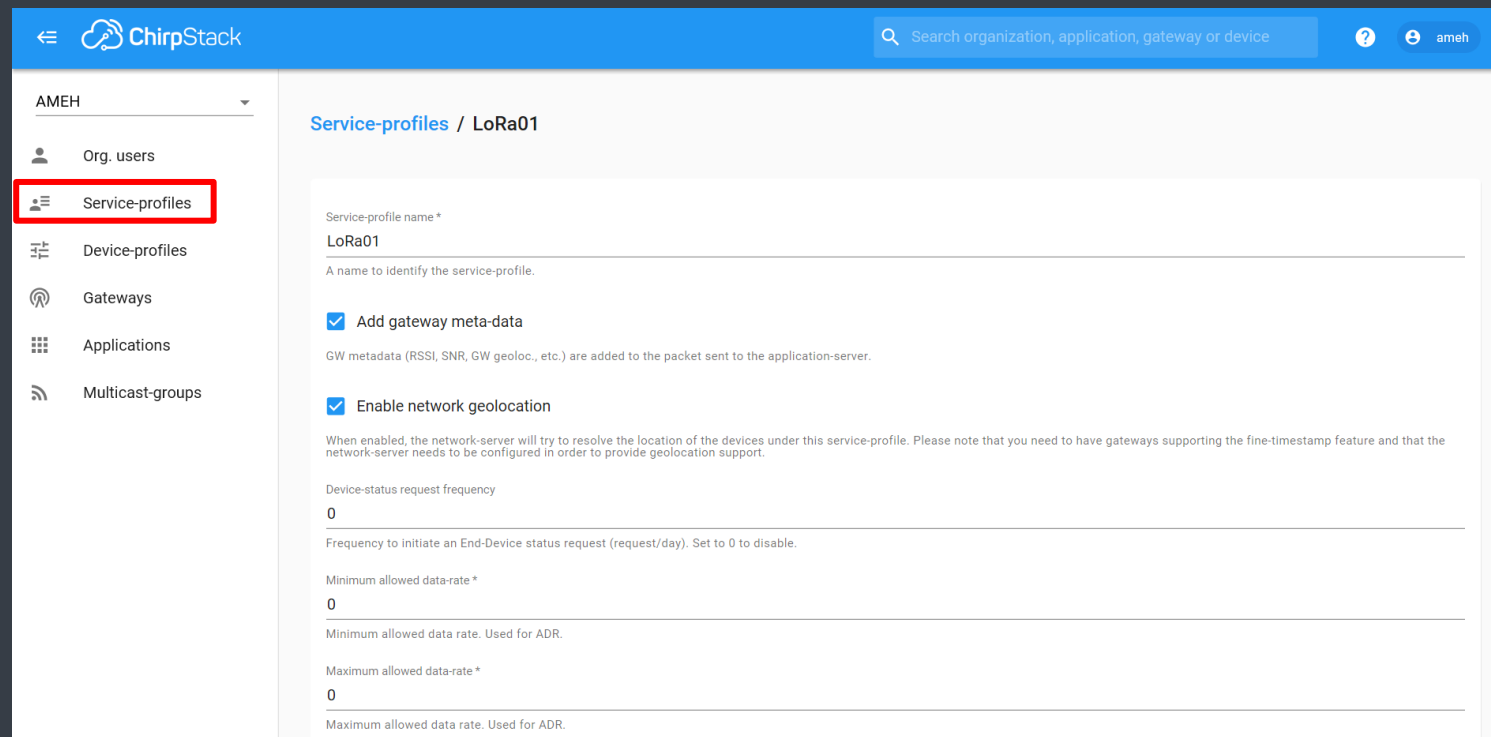
```
C: > Projects > libraries > arduino-lmic-master > project_config > C lmic_project_config.h
1 // project-specific definitions
2 #define CFG_eu868 1
3 //#define CFG_us915 1
4 //#define CFG_au915 1
5 //#define CFG_as923 1
6 // #define LMIC_COUNTRY_CODE LMIC_COUNTRY_CODE_JP /* for as923-JP */
7 //#define CFG_kr920 1
8 //#define CFG_in866 1
9 #define CFG_sx1276_radio 1
10 //#define LMIC_USE_INTERRUPTS
11 #define LMIC_LORAWAN_SPEC_VERSION LMIC_LORAWAN_SPEC_VERSION_1_0_2
12 #define DISABLE_PING
13 #define LMIC_ENABLE_DeviceTimeReq 1
14 #define LMIC_PRINTF_TO Serial
```

EUIS AND KEY INFORMATION

- **Device EUI**: It is a global end device ID in IEEE EUI64 address space that uniquely identifies the end device. The user can derive their own DevEUI.
- **Application Key**: The AppKey is an AES128 root key specific to the end device. Whenever an end device joins a network via over the air activation(OTAA), the AppKey is used to derive the session keys NwkSKey and AppSKey specific for that end device to encrypt and verify network communication and application data. The AppKey should be unique for each device. The user can derive their own AppKey.
- **Application EUI**: It is a global application ID in IEEE EUI64 address space that uniquely identifies the entity able to process the JoinReq frame. The AppEUI is stored in the end-device before the activation procedure is executed. The AppEUI can be different for each device or it can also be same for all device. It also depends on what kind of application server you are using. The user can derive their own AppEUI.

SETTING UP THE SERVER

- For this project a ChirpStack server was used to receive the data from the board. After logging into the server a **Service-profile** needs to be created if there no existing one yet. Click in “**Service-profiles**” and create a new one with the following specifications (the name given is completely optional):



The screenshot shows the ChirpStack web interface. The top navigation bar is blue with the ChirpStack logo on the left, a search bar in the center, and a user profile 'ameh' on the right. A left sidebar contains a menu with items: 'AMEH' (selected), 'Org. users', 'Service-profiles' (highlighted with a red box), 'Device-profiles', 'Gateways', 'Applications', and 'Multicast-groups'. The main content area is titled 'Service-profiles / LoRa01'. It contains a form for configuring a service profile. The 'Service-profile name' field is filled with 'LoRa01'. Below this, there are two checked checkboxes: 'Add gateway meta-data' and 'Enable network geolocation'. The 'Device-status request frequency' is set to '0'. The 'Minimum allowed data-rate' and 'Maximum allowed data-rate' are both set to '0'. Each input field has a descriptive text below it.

ChirpStack

Search organization, application, gateway or device

ameh

AMEH

Org. users

Service-profiles

Device-profiles

Gateways

Applications

Multicast-groups

Service-profiles / LoRa01

Service-profile name *

LoRa01

A name to identify the service-profile.

☒ Add gateway meta-data

GW metadata (RSSI, SNR, GW geoloc., etc.) are added to the packet sent to the application-server.

☒ Enable network geolocation

When enabled, the network-server will try to resolve the location of the devices under this service-profile. Please note that you need to have gateways supporting the fine-timestamp feature and that the network-server needs to be configured in order to provide geolocation support.

Device-status request frequency

0

Frequency to initiate an End-Device status request (request/day). Set to 0 to disable.

Minimum allowed data-rate *

0

Minimum allowed data rate. Used for ADR.

Maximum allowed data-rate *

0

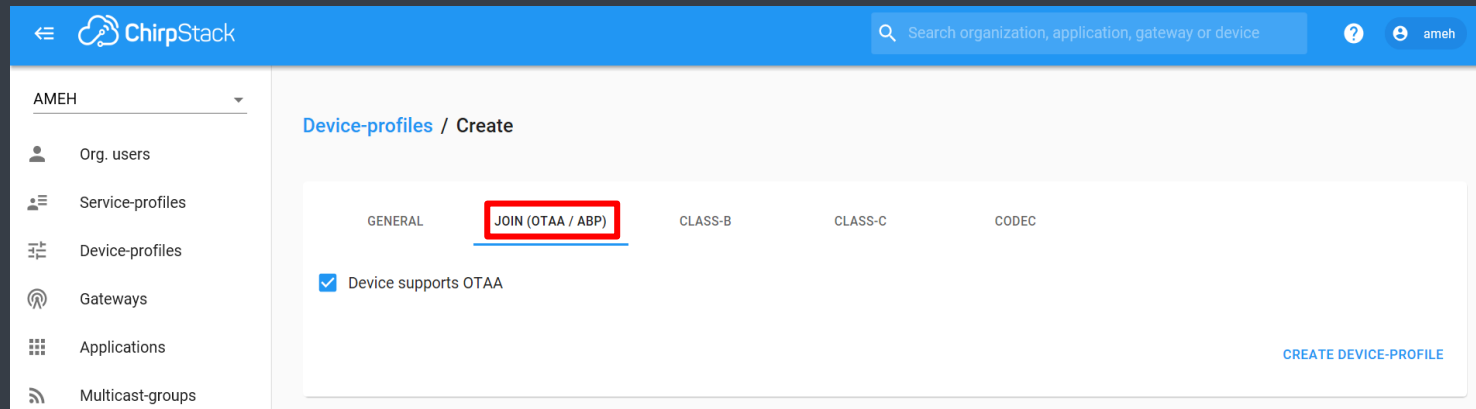
Maximum allowed data rate. Used for ADR.

- As following click on “**Device-profiles**” and create a new one, give in the following specifications and select the created Service-profile for the “**Network server**” field. There are still other configurations that need to be done, so do not create the profile yet.

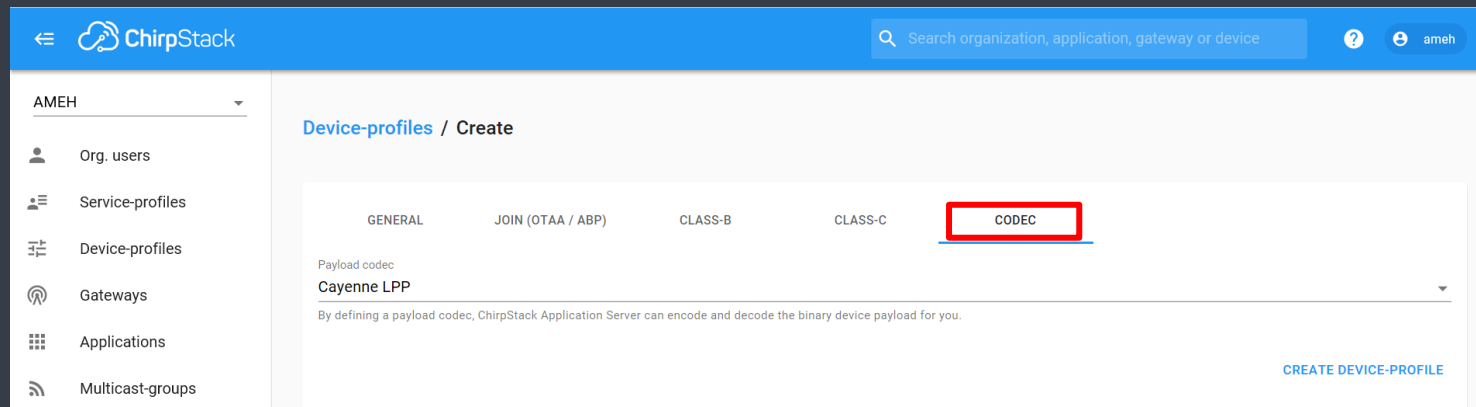
The screenshot shows the ChirpStack web interface. The top navigation bar is blue with the ChirpStack logo and a search bar. The left sidebar is white with a list of menu items: AMEH, Org. users, Service-profiles, Device-profiles (highlighted with a red box), Gateways, Applications, and Multicast-groups. The main content area is white and shows the 'Device-profiles' configuration page. The 'GENERAL' tab is selected and highlighted with a red box. The form contains the following fields:

- Device-profile name *: A name to identify the device-profile.
- Network-server *: Select network-server. The network-server on which this device-profile will be provisioned. After creating the device-profile, this value can't be changed.
- LoRaWAN MAC version *: 1.0.2. The LoRaWAN MAC version supported by the device.
- LoRaWAN Regional Parameters revision *: A. Revision of the Regional Parameters specification supported by the device.
- Max EIRP *: 0. Maximum EIRP supported by the device.
- Geolocation buffer TTL (seconds): 0. The time in seconds that historical uplinks will be stored in the geolocation buffer.
- Geolocation minimum buffer size: 0. The minimum buffer size required before using geolocation (when enabled in the Service Profile). Using multiple uplinks for geolocation can increase the accuracy of the geolocation results.

- Enable “**Device supports OTAA**” (over the air activation) and select “**Cayenne LPP**” as codec for the Payload. After this two steps click on “**create Device-profile**”:



The screenshot shows the ChirpStack web interface for creating a device profile. The left sidebar lists navigation options: Org. users, Service-profiles, Device-profiles, Gateways, Applications, and Multicast-groups. The main content area is titled "Device-profiles / Create" and features five tabs: GENERAL, JOIN (OTAA / ABP), CLASS-B, CLASS-C, and CODEC. The "JOIN (OTAA / ABP)" tab is selected and highlighted with a red box. Below the tabs, there is a checkbox labeled "Device supports OTAA" which is checked. A "CREATE DEVICE-PROFILE" button is located at the bottom right of the form.



The screenshot shows the same ChirpStack web interface, but now the "CODEC" tab is selected and highlighted with a red box. The "Payload codec" dropdown menu is open, showing "Cayenne LPP" as the selected option. Below the dropdown, a note states: "By defining a payload codec, ChirpStack Application Server can encode and decode the binary device payload for you." The "CREATE DEVICE-PROFILE" button remains at the bottom right.

- In order to receive data from the board a gateway needs to be created. Important here is the **Gateway ID**, most antennas and gateway devices have their own ID. If the device has no ID, one can be randomly generated by clicking in the arrow on the right side of the window.
- Select the created Service-profile for the “**Network server**” field, enable the “**Gateway discovery**” and then create the gateway.

ChirpStack

Search organization, application, gateway or device


AMEH

- Org. users
- Service-profiles
- Device-profiles
- Gateways**
- Applications
- Multicast-groups

Gateway name *

The name may only contain words, numbers and dashes.

Gateway description *

Gateway ID * MSB 

Network-server *

Select network-server

Select the network-server to which the gateway will connect. When no network-servers are available in the dropdown, make sure a service-profile exists for this organization.

Gateway-profile

Select gateway-profile

An optional gateway-profile which can be assigned to a gateway. This configuration can be used to automatically re-configure the gateway when ChirpStack Gateway Bridge is configured so that it manages the packet-forwarder configuration.

☒ Gateway discovery enabled

When enabled (and ChirpStack Network Server is configured with the gateway discover feature enabled), the gateway will send out periodical pings to test its coverage by other gateways in the same network.

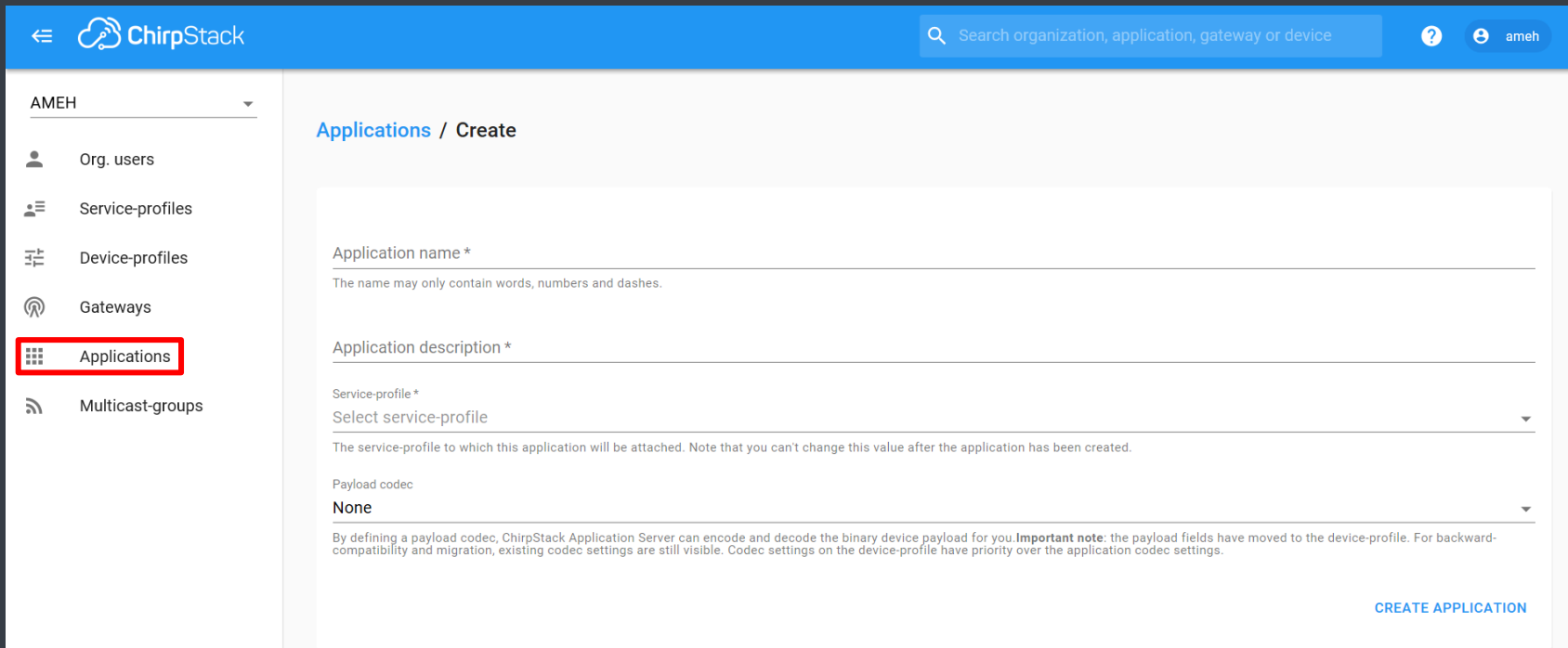
Gateway altitude (meters) *

0

When the gateway has an on-board GPS, this value will be set automatically when the network has received statistics from the gateway.

Generate random ID

- Go to “**Applications**” and create a new application. Write a name and a description, give in the Service-profile and select “**None**” as Payload codec.



The screenshot shows the ChirpStack web interface. The top navigation bar is blue with the ChirpStack logo and a search bar. The left sidebar contains a list of menu items: AMEH, Org. users, Service-profiles, Device-profiles, Gateways, Applications (highlighted with a red box), and Multicast-groups. The main content area is titled 'Applications / Create' and contains a form with the following fields:

- Application name ***: A text input field with a note below it: 'The name may only contain words, numbers and dashes.'
- Application description ***: A text input field.
- Service-profile ***: A dropdown menu with the text 'Select service-profile' and a note below it: 'The service-profile to which this application will be attached. Note that you can't change this value after the application has been created.'
- Payload codec**: A dropdown menu with 'None' selected and a note below it: 'By defining a payload codec, ChirpStack Application Server can encode and decode the binary device payload for you. **Important note:** the payload fields have moved to the device-profile. For backward-compatibility and migration, existing codec settings are still visible. Codec settings on the device-profile have priority over the application codec settings.'

A blue button labeled 'CREATE APPLICATION' is located at the bottom right of the form.

- Once the application is created, click on it and it will allow you to create devices inside of the chosen application. Some devices already have their own **Device EUI**. If there is no given EUI, a random one can be generated by clicking on the arrow button.
- It is important to change the **DevEUI** format from “**Most Significant Byte**” (**MSB**) to “**Least Significant Byte**” (**LSB**) before generating the EUI. This can be done by clicking on MSB.

The screenshot shows the ChirpStack web interface. The top navigation bar is blue with the ChirpStack logo on the left and a search bar, help icon, and user profile 'ameh' on the right. The left sidebar contains a dropdown menu for 'AMEH' and a list of navigation items: 'Org. users', 'Service-profiles', 'Device-profiles', 'Gateways', 'Applications', and 'Multicast-groups'. The main content area has a breadcrumb trail: 'Applications / Temperatursensor1 / Devices / Create'. Below this is a form with three tabs: 'GENERAL' (selected), 'VARIABLES', and 'TAGS'. The form contains the following fields: 'Device name *' with a note 'The name may only contain words, numbers and dashes.', 'Device description *', 'Device EUI *' with a red box around the 'MSB' button and a refresh icon, and a 'Device-profile *' dropdown menu. At the bottom, there is a checkbox for 'Disable frame-counter validation' and a note: 'Note that disabling the frame-counter validation will compromise security as it enables people to perform replay-attacks.' A 'CREATE DEVICE' button is located at the bottom right of the form.

- Once the device is created it should be visible every time the application where it was created is open. The **Device EUI** of each device will be also shown.

ChirpStack

Search organization, application, gateway or device

AMEH

- Org. users
- Service-profiles
- Device-profiles
- Gateways
- Applications
- Multicast-groups

Applications / Temperatursensor1

DEVICES APPLICATION CONFIGURATION INTEGRATIONS FUOTA

+ CREATE

Last seen	Device name	Device EUI	Link margin	Battery
a few seconds ago	M5Stack1		n/a	n/a
a minute ago	TTG01		n/a	n/a

Rows per page: 10 1-2 of 2

- By clicking on the device name a new window will be open. All the device information will be shown here as well as the **Application key**, which will be later needed for the over the air activation.
- Clicking on the eye symbol will allow you to see the Appkey.

ChirpStack

Search organization, application, gateway or device

AMEH

- Org. users
- Service-profiles
- Device-profiles
- Gateways
- Applications
- Multicast-groups

Applications / Temperatursensor1 / Devices / TTG01

DETAILS CONFIGURATION KEYS (OTAA) ACTIVATION DEVICE DATA LORAWAN FRAMES FIRMWARE

DELETE

Application key *

For LoRaWAN 1.0 devices. In case your device supports LoRaWAN 1.1, update the device-profile first.

Gen Application key

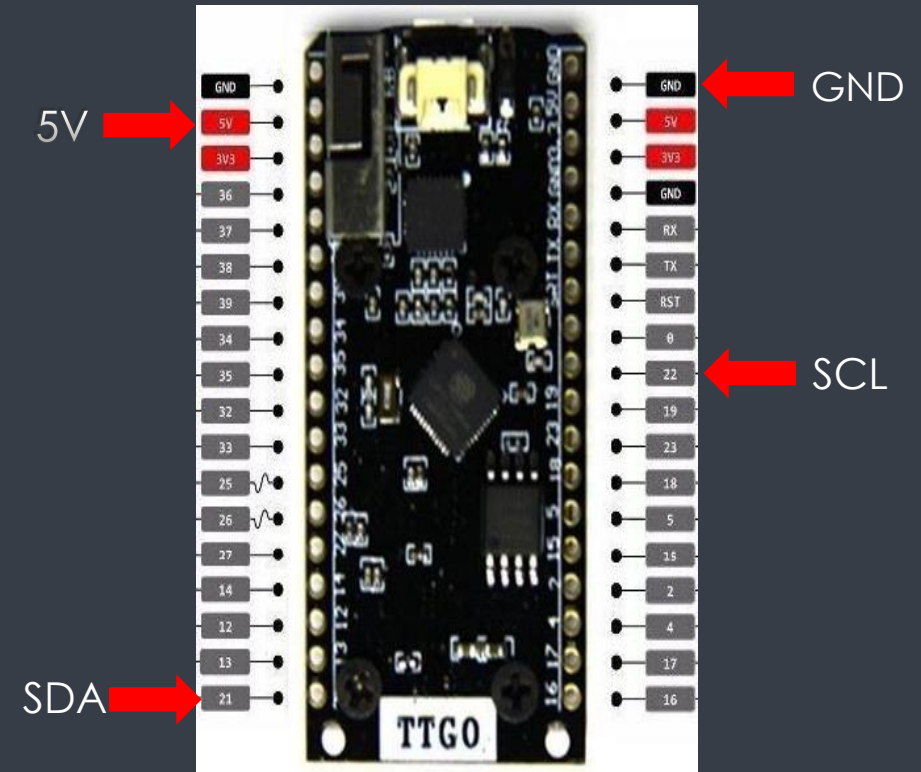
For LoRaWAN 1.0 devices. This key must only be set when the device implements the remote multicast setup specification / firmware updates over the air (FUOTA). Else leave this field blank.

SET DEVICE-KEYS

CONNECTING THE SENSOR



- Ground has to be connected to any of the “**GND**” pins in the board.
- The voltage wire should go to any of the “**5V**” pins.
- Connect “**SDA**” to pin 21.
- Connect “**SCL**” to pin 22.



APPLYING THE KEYS TO THE PROGRAM

- The **AppEUI** and **DevEUI** have to be written in little endian format this means least significant byte first. If the DevEUI was generated in LSB format then the EUI can be written in the order it appears. If not, the order has to be reversed. Here is an example:

DevEUI (MSB): ab12cd34ef56gh78

DevEUI (LSB): 78gh56ef34cd12ab

DevEUI (program): 0x78, 0xgh, 0x56, 0xef, 0x34, 0xcd, 0x12, 0xab

- Since the ChirpStack server does not provide us with the **AppEUI** one has to be made up. The format to write it, is the same as in the example above.

AppEUI: 0x45, 0x23, 0x01, 0x89, 0x67, 0x45, 0x23, 0x01

- The **AppKey** can be written in the format given by the server (MSB), it will only be needed to add “0x __,” every two characters, so that it can be accepted in the array.

Appkey: 0x11, 0x2a, 0x33, 0x4b, 0x55, 0x6c, 0x77, 0x8d,
0x99, 0x1e, 0x22, 0x3f, 0x44, 0x5g, 0x66, 0x7h

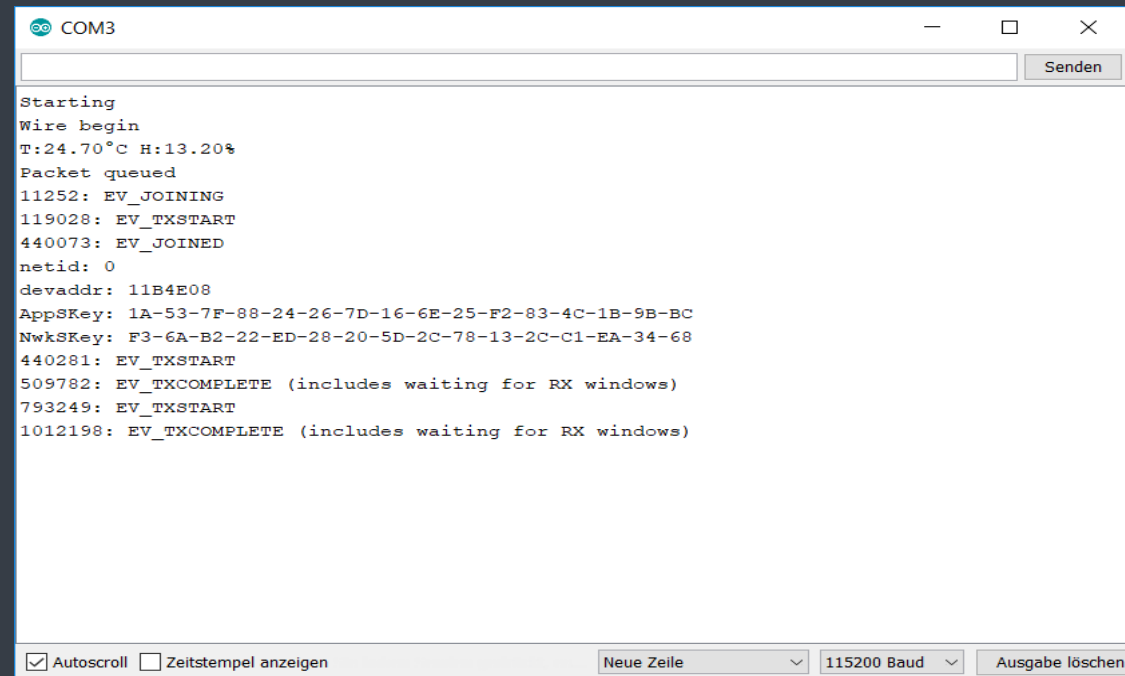
```
// This EUI must be in little-endian format, so least-significant-byte
// first. When copying an EUI from ttnctl output, this means to reverse
// the bytes. For TTN issued EUIs the last bytes should be 0xD5, 0xB3,
// 0x70.
static const u1_t PROGMEM APPEUI[8]={ 0x78, 0xgh, 0x56, 0xef, 0x34, 0xcd, 0x12, 0xab };
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}

// This should also be in little endian format, see above.
static const u1_t PROGMEM DEVEUI[8]={ 0x45, 0x23, 0x01, 0x89, 0x67, 0x45, 0x23, 0x01 };
void os_getDevEui (u1_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 ttnctl can be copied as-is.
static const u1_t PROGMEM APPKEY[16] = { 0x11, 0x2a, 0x33, 0x4b, 0x55, 0x6c, 0x77, 0x8d, 0x99, 0x1e, 0x22, 0x3f, 0x44, 0x5g, 0x66, 0x7h };
void os_getDevKey (u1_t* buf) { memcpy_P(buf, APPKEY, 16);}
```


RUNNING THE PROGRAM


- Make sure the board is connected to the PC with the USB to MicroUSB cable and that the correct upload speed and port are selected so that you can compile and upload the program. Once you have uploaded all the code, open the serial monitor and something like this should come up. The **device address** will be generated as well as the **AppSKey** and **NwkSKey**.

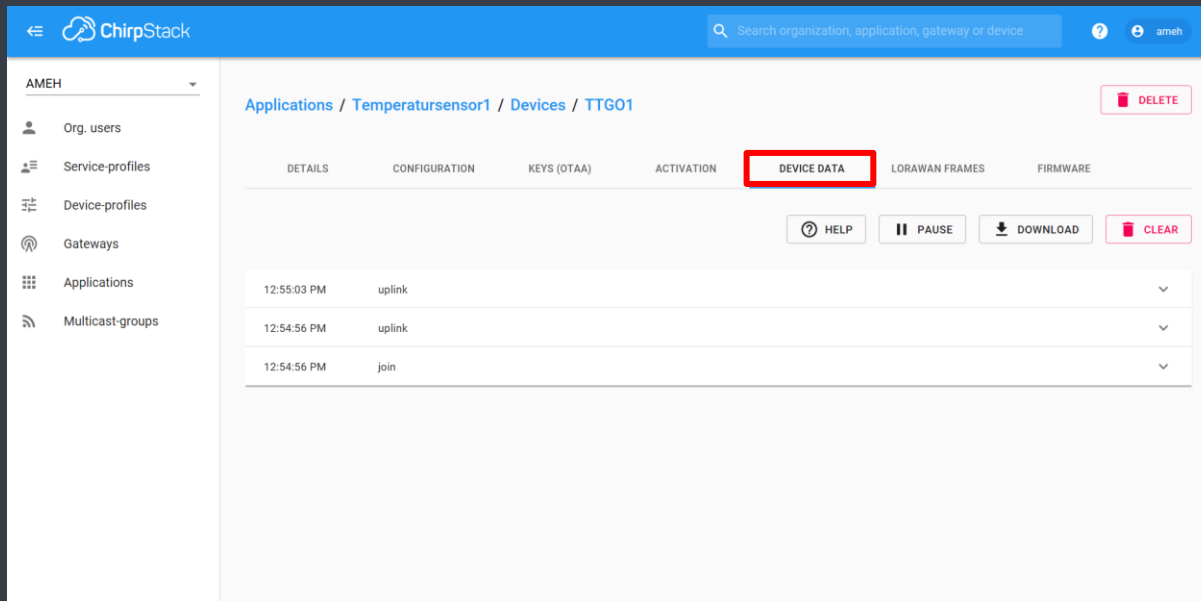


The screenshot shows a serial monitor window with the title 'COM3'. It features a text input field at the top with a 'Senden' button to its right. The main area displays the following text output:

```
Starting
Wire begin
T:24.70°C H:13.20%
Packet queued
11252: EV_JOINING
119028: EV_TXSTART
440073: EV_JOINED
netid: 0
devaddr: 11B4E08
AppSKey: 1A-53-7F-88-24-26-7D-16-6E-25-F2-83-4C-1B-9B-BC
NwkSKey: F3-6A-B2-22-ED-28-20-5D-2C-78-13-2C-C1-EA-34-68
440281: EV_TXSTART
509782: EV_TXCOMPLETE (includes waiting for RX windows)
793249: EV_TXSTART
1012198: EV_TXCOMPLETE (includes waiting for RX windows)
```

At the bottom, there is a status bar with several controls: a checked 'Autoscroll' checkbox, an unchecked 'Zeitstempel anzeigen' checkbox, a 'Neue Zeile' dropdown menu, a '115200 Baud' dropdown menu, and an 'Ausgabe löschen' button.

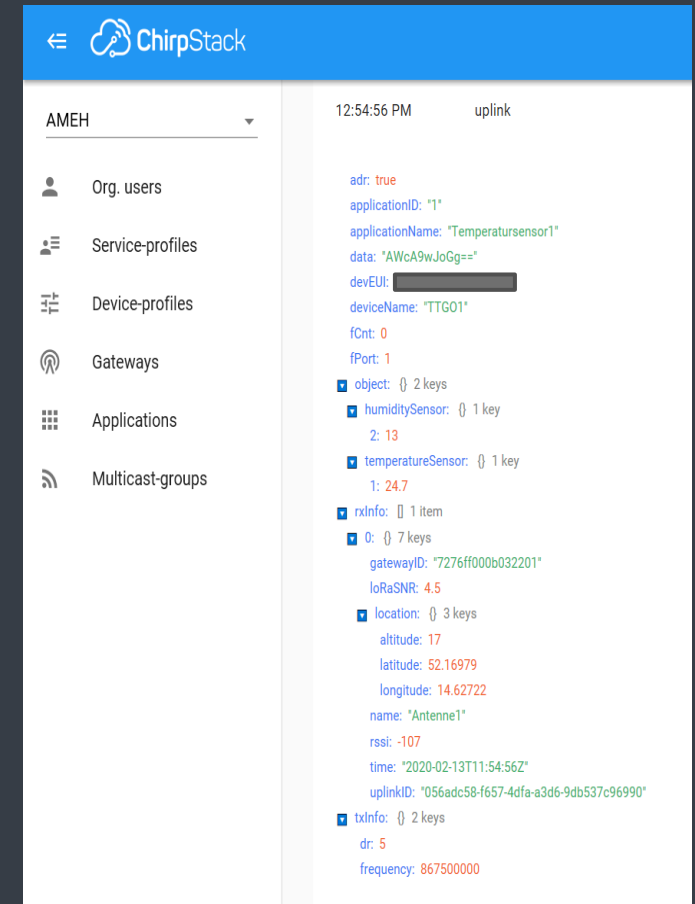
- Open the application and select the created device, if everything was correct the server should be receiving the data from the board. The data can be seen in the menu “**Device Data**”. The data displayed should look like this 



The screenshot shows the ChirpStack web interface. The left sidebar contains a menu with items: Org. users, Service-profiles, Device-profiles, Gateways, Applications, and Multicast-groups. The main content area is titled 'Applications / Temperatursensor1 / Devices / TTG01'. Below the title is a tabbed interface with tabs: DETAILS, CONFIGURATION, KEYS (OTAA), ACTIVATION, **DEVICE DATA** (highlighted with a red box), LORAWAN FRAMES, and FIRMWARE. The 'DEVICE DATA' tab displays a table of data points:

Time	Direction	Action
12:55:03 PM	uplink	▼
12:54:56 PM	uplink	▼
12:54:56 PM	join	▼

Below the table are buttons: HELP, PAUSE, DOWNLOAD, and CLEAR. A 'DELETE' button is also present in the top right corner of the main content area.



The screenshot shows the ChirpStack web interface. The left sidebar contains a menu with items: Org. users, Service-profiles, Device-profiles, Gateways, Applications, and Multicast-groups. The main content area is titled 'Applications / Temperatursensor1 / Devices / TTG01'. Below the title is a tabbed interface with tabs: DETAILS, CONFIGURATION, KEYS (OTAA), ACTIVATION, **DEVICE DATA** (highlighted with a red box), LORAWAN FRAMES, and FIRMWARE. The 'DEVICE DATA' tab displays a table of data points:

Time	Direction	Action
12:55:03 PM	uplink	▼
12:54:56 PM	uplink	▼
12:54:56 PM	join	▼

Below the table are buttons: HELP, PAUSE, DOWNLOAD, and CLEAR. A 'DELETE' button is also present in the top right corner of the main content area.

SOURCES

- <https://www.thethingsnetwork.org/forum/t/big-esp32-sx127x-topic-part-3/18436>
- <https://www.bjoerns-techblog.de/2019/02/adr-mit-the-things-network/>
- <https://www.thethingsnetwork.org/forum/t/solved-adafruit-feather-m0-to-connect-to-ttn-over-otaa-unknown-event-20/29990>
- <https://stackoverflow.com/questions/54096980/lorawan-deveui-appkey-and-appkey>
- <https://m5stack.com/products/mini-env-sensor-unit?variant=16804795940954>
- <https://github.com/mcci-catena/arduino-lmic/blob/master/README.md>