AN2201
Application note

Getting started guide for Move-X Cicerone and MAMWLE-based boards using Arduino IDE

Introduction

The purpose of this document is to guide the user in setting up the firmware development process using the Arduino IDE, the Integrated Desktop Environment for Arduino boards that provides all the tools needed for user-friendly and fast firmware development.
Given the success and popularity of the Arduino philosophy, Move-X provides **Move-Xduino**, the Arduino core that makes it possible to develop firmware for all the boards based on Move-X's MAMWLE module.

Note: the new Arduino IDE 2.0 is now available in stable release. Given the many improvements it provides over the previous version, we suggest the users to use the latest version available.

Move-Xduino also provides examples that make it easy for the users to turn their ideas into real things. These examples can be used for both official Move-X boards as well as custom-designed by the MAMWLE users.

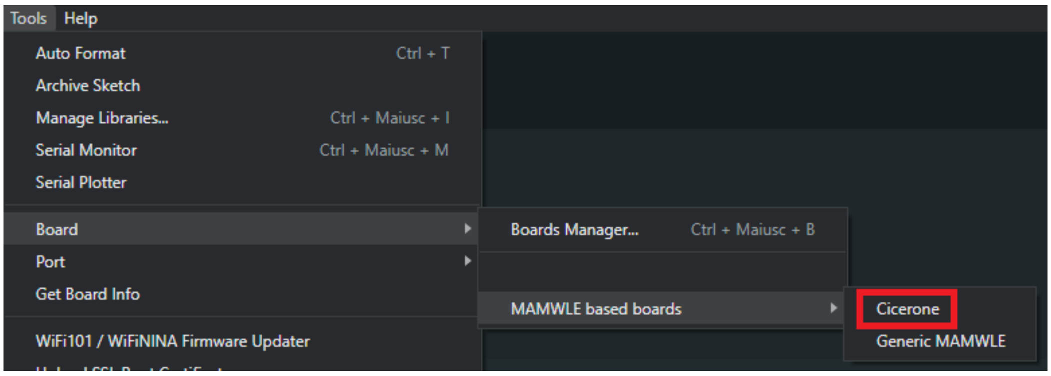
For more resources, news and links to official distributors for Move-X products, please visit Move-X website: https://www.move-x.it/.

Contents

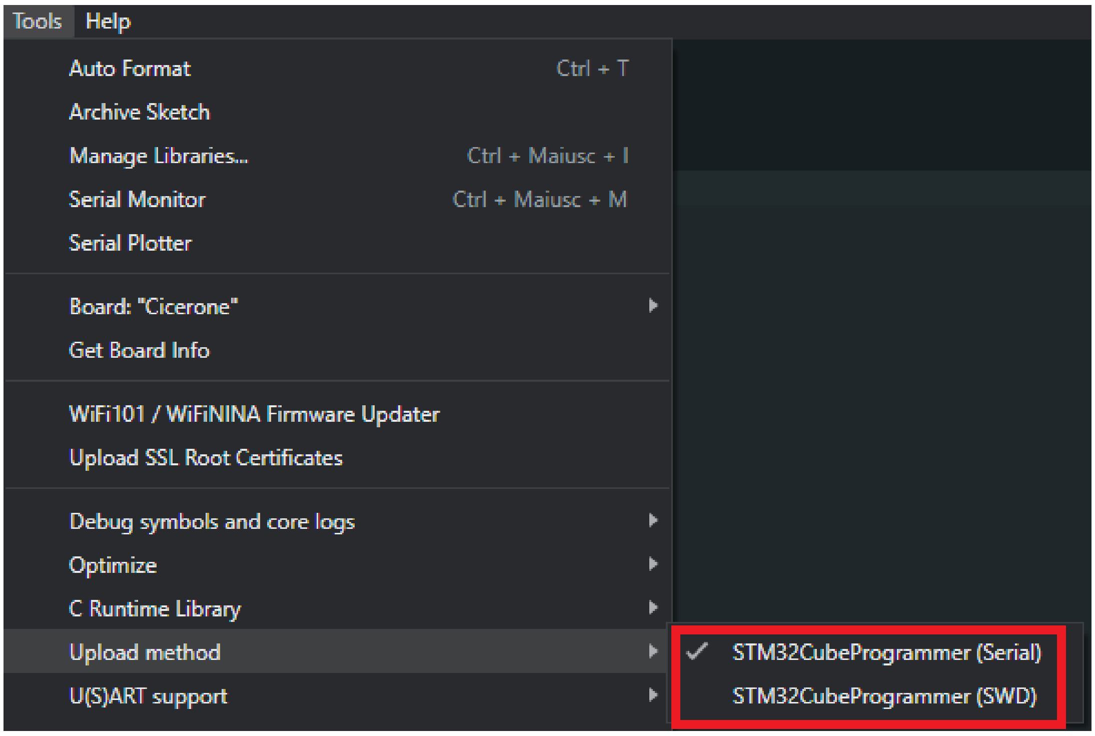
Setting up Arduino IDE for Move-X

Quick start guide to start developing your ideas within the Arduino IDE. The following guide is targeted to the basic “Blink” example for the Move-X Cicerone board. Steps:

1. Download and install the Arduino IDE. Downloads and instructions can be found in the official website: https://www.arduino.cc/en/software
2. Add Move-Xduino package to the Arduino IDE. You can find the package and installation instructions here: https://github.com/Move-X/Move-Xduino
3. Open Arduino IDE and select *File / Examples / 0..Basics / Blink*
4. Go to *Tools / Board* and select the Move-X Cicerone

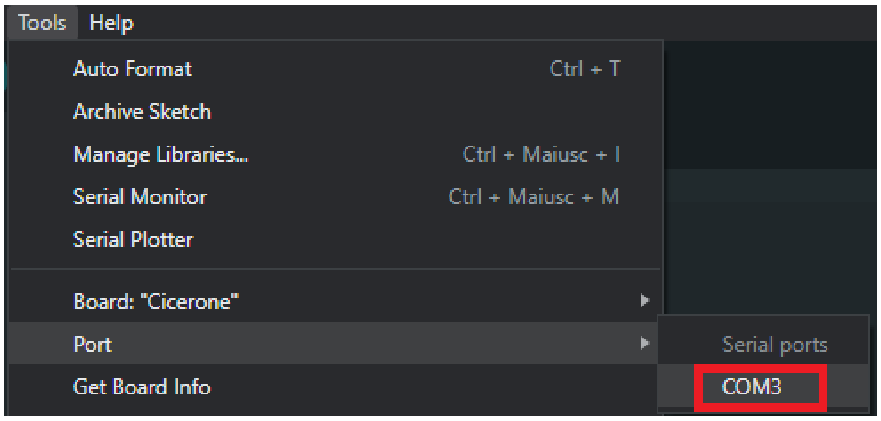

Tools
Help
Auto Format Ctrl + T
Archive Sketch
Manage Libraries... Ctrl + Maiusc + I
Serial Monitor Ctrl + Maiusc + M
Serial Plotter
Board
Port
Get Board Info
WiFi101 / WiFiNINA Firmware Updater
Boards Manager... Ctrl + Maiusc + B
MAMWLE based boards
Cicerone
Generic MAMWLE

5. Select the upload method: default is through Serial (USB), but it is also possible to use SWD debuggers such as ST-Link. To change this setting go to *Tools / Upload method*


Tools
Help
Auto Format Ctrl + T
Archive Sketch
Manage Libraries... Ctrl + Maiusc + I
Serial Monitor Ctrl + Maiusc + M
Serial Plotter
Board: "Cicerone"
Get Board Info
WiFi101 / WiFiNINA Firmware Updater
Upload SSL Root Certificates
Debug symbols and core logs
Optimize
C Runtime Library
Upload method
U(S)ART support
✓ STM32CubeProgrammer (Serial)
STM32CubeProgrammer (SWD)

6. Connect the board to your PC through USB cable

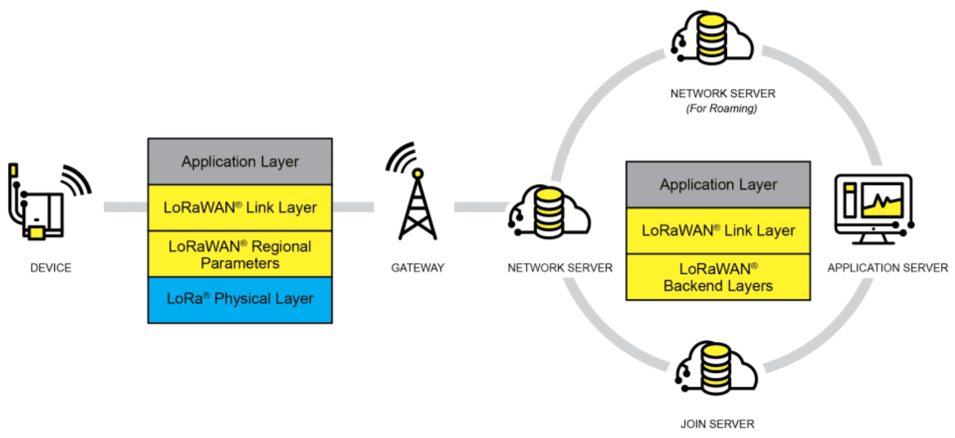
7. Select the port your board is connected to by going to *Tools* / *Port*. Port number is chosen by the OS of your PC, in our case it is named “COM3”



8. Compile and upload the example by clicking the *Upload* button
9. Check whether the onboard LED is blinking

What is LoRaWAN

LoRaWAN is a Low Power, Wide Area (LPWA) networking protocol designed to wirelessly connect battery operated 'things' to the internet in regional, national or global networks, and targets key Internet of Things (IoT) requirements such as bi-directional communication, end-to-end security, mobility and localization services.

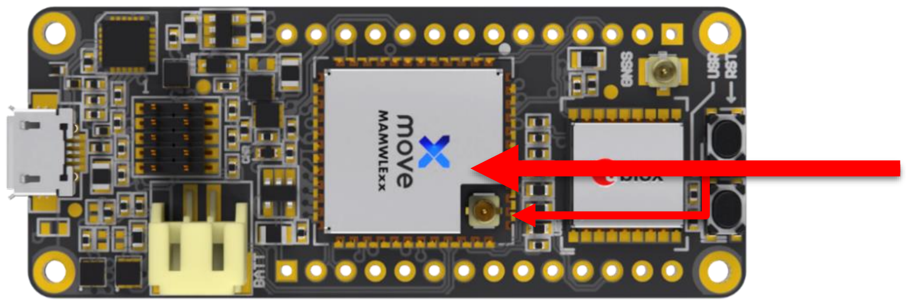


LoRaWAN network architecture is deployed in a star-of-stars topology in which gateways relay messages between end-devices and a central network server. The gateways are connected to the network server via standard IP connections and act as a transparent bridge, simply converting RF packets to IP packets and vice versa. The wireless communication takes advantage of the Long Range characteristics of the LoRa physical layer, allowing a single-hop link between the end-device and one or many gateways.

More info about LoRaWAN networks can be found at https://lora-alliance.org/about-lorawan/.

Setting up MAMWLE LoRa Module

The MAMWLE module by Move-X is a LoRa module with an STM32 programmable MCU inside.



To use the LoRa Module, as in figure (showing our Cicerone Board), you need to connect a **Passive** Antenna for LoRa frequency band (860 – 930 MHz) with a U.FL Female connector.

Example: LoRaWAN_End_Node

This example provides the user a reference for a LoRaWAN end node project capable of handling both the LoRaWAN stack and user application code. It is targeted for the Cicerone board by Move-X but can be easily ported to custom boards based on Move-X's MAMWLE module. Moreover, the example presumes that the user is already familiar with LoRawan network architecture.

> *Note: the user should provide valid LoRaWAN keys for the end-device, which are provided by the network/join server. There are network server vendors offering free options which are perfect for test purposes, examples are https://www.thethingsnetwork.org/ and https://www.loriot.io/. To make it work the end device must be placed within reach of a gateway connected to the chosen network server, if no gateway is available in the surroundings the user should install its own.*

The sketch implements a demo device that handles both uplinks and downlink: when the user button is pressed, a message with LED ON period time in milliseconds (0-255) is sent on LoRaWAN port 111. If the node receives a downlink on the same port with a payload length of one byte (0-255), the node updates the LED ON time with the received value.
The implemented end-device is compliant to LoRaWAN class A: the node listens for downlink only when it sends an uplink message (i.e. the button is pressed by the user). Class A behaves this way in order to maximize battery life by minimizing the time the node's radio is on waiting for incoming transmissions.
Log messages are printed through to the serial port (UART), so the user can monitor the sketch behavior using Arduino IDE's serial monitor.

LoRaWAN network EUI and keys

To join the LoRaWAN network valid parameters provided by the Network Server are required. The sketch is not targeted to a specific Network Server, so the user can choose his favorite one.
The LoRaWAN parameters required are *devEUI, appEUI, appKEY*. The user can enter his set of parameters in the global variable at the top of the sketch as a sequence of 8-bit HEX values, as in the example.

```
uint8_t devEUI[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
uint8_t appEUI[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
uint8_t appKEY[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00};
```

setup **function**

As usual for Arduino sketches here we do all the initializations:

- serial port (MAMWLE's USART2, baudrate 115200)
- built-in LED (LED on on HIGH level)
- interrupt for the onboard button (interrupt on rising edge)
- initialization of the LoRaWAN stack as follows:
    - region: EU868 (must be set by the user to match its actual region in the 868-915MHz band)
    - devEUI, joinEUI/appEUI, appKEY, networkKey (same as appKEY in the example), join type (OTAA in the example)

loop **function**

Contrary to how it normally happens for Arduino sketches, the *loop* function is not used for application code.

```
void loop()
{
  LoRaWAN.process();
}
```

Its purpose is to handle the LoRaWAN stack management process, and the user should avoid editing it: user's application code can be placed in the *userLoop* function instead.

> *Note: editing this function might cause the LoRaWAN stack to behave otherwise than expected.*

userLoop **function**

This has the same role as the *loop* function in traditional Arduino sketches. It handles the infinite loop where the user can place its code instead of using the *loop* function (which shouldn't be edited).
In this example the function performs the following actions:

- checks if the device already joined the LoRaWAN network
- if already joined it handles the LED status
    - if LED is on, waits until it has been turned on for *led_interval_on* milliseconds then turns it off

- if LED is off, waits until it has been turned off for *LED_INTERVAL_OFF* milliseconds then turns it on

```
    /* Update led */
    unsigned long currentMillis = millis();

    if   (currentMillis   -   previousMillis   >=   (led_on   ?   led_interval_on   :
LED_INTERVAL_OFF)) {
      previousMillis = currentMillis;

      if (!led_on)
      {
        analogWrite(LED_PIN, LED_INTENSITY);
        led_on = true;
      }
      else
      {
        analogWrite(LED_PIN, 0);
        led_on = false;
      }
    }
```

Note: the onboard LED is turned on in PWM mode using analogWrite with LED_INTENSITY value. This allows to reduce power consumption and avoid overstressing the eyes since the onboard LED is very bright.

The function code should avoid usage of blocking code such as *delay* and *delayMicroseconds* functions, active waiting for events, polling external components and long I/O sessions. These can be replaced by structuring the code in a more efficient way (see table).

Standard approach	**Replace with**

delay()
delayMicroseconds()
	millis()
	micros()

Active wait for events	State variables or finite state machines to track changes between consecutive executions
Polling inputs	Interrupts

Data I/O that requires long time	Split I/O session between consecutive executions using state variables or finite state machines

onUsrBtn function

This is the callback function executed when the onboard button is pressed, doing the following actions:

- logs to serial port
- starts the transmission of a LoRaWAN unconfirmed packet on *LED_LORA_PORT* with 1-byte payload corresponding to the value stored in the *led_interval_on* variable.

```
  if (LRW_OK != LoRaWAN.Send(TxPort, LRW_UNCONFIRMED_MSG, TxData, TxSize))
    Serial.println("Send() error!");
```

Function code should avoid blocking code as explained for *userLoop* function.

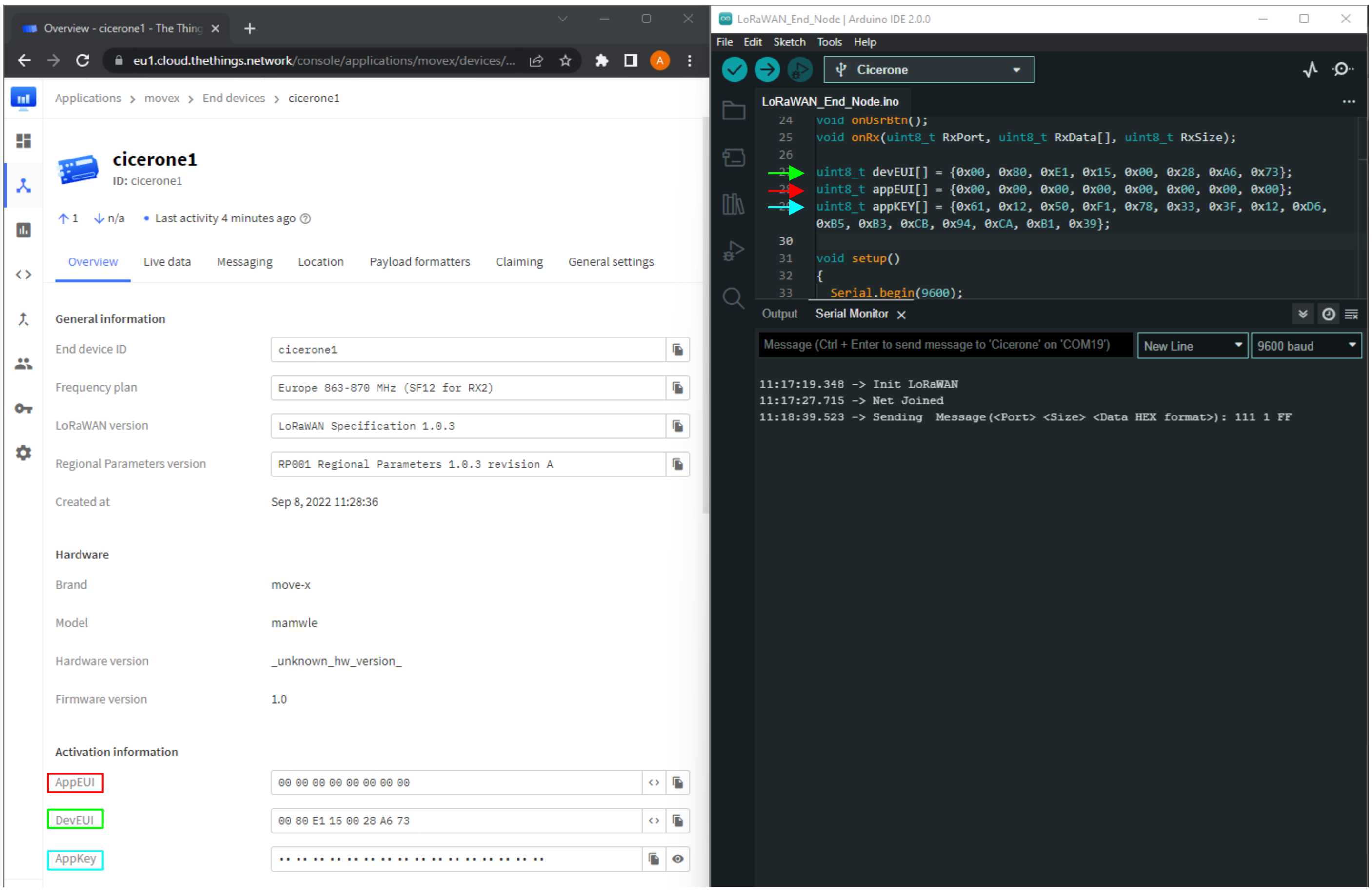
onRx function

This is the callback function executed when a LoRaWAN packet is received (incoming downlink with matching EUI and keys).
The code checks if the packet's port matches the *LED_LORA_PORT*. If this is true, the *led_interval_on* variable is updated with the value in the first payload byte, actually updating the LED's ON time from now on.
Function code should avoid blocking code as explained for *userLoop* function.

Testing uplinks and downlink

Here is an example output using The Things Network.

- Registered device and keys added to the sketch


Overview - cicerone1 - The Thing
eu1.cloud.thethings.network/console/applications/movex/devices/...
Applications > movex > End devices > cicerone1
Overview
Live data
Messaging
Location
Payload formatters
Claiming
General settings
LoRaWAN_End_Node | Arduino IDE 2.0.0
File Edit Sketch Tools Help
Cicerone
LoRaWAN_End_Node.ino
Output
Serial Monitor
Message (Ctrl + Enter to send message to 'Cicerone' on 'COM19')
New Line
9600 baud

cicerone1
ID: cicerone1

↑1 ↓n/a • Last activity 4 minutes ago

General information

End device ID	cicerone1
Frequency plan	Europe 863-870 MHz (SF12 for RX2)
LoRaWAN version	LoRaWAN Specification 1.0.3
Regional Parameters version	RP001 Regional Parameters 1.0.3 revision A
Created at	Sep 8, 2022 11:28:36

Hardware

Brand	move-x
Model	mamwle
Hardware version	_unknown_hw_version_
Firmware version	1.0

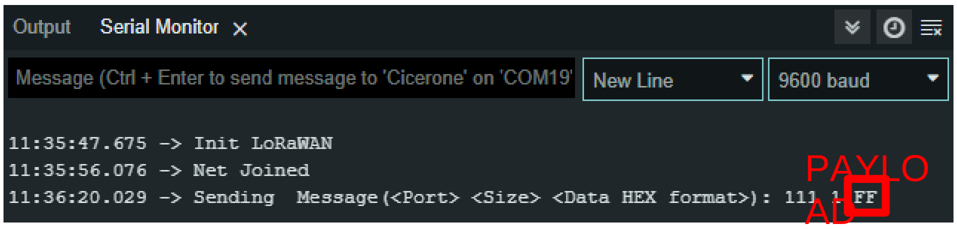
Activation information

AppEUI	00 00 00 00 00 00 00 00
DevEUI	00 80 E1 15 00 28 A6 73
AppKey	•• •• •• •• •• •• •• •• •• •• •• •• •• •• •• ••

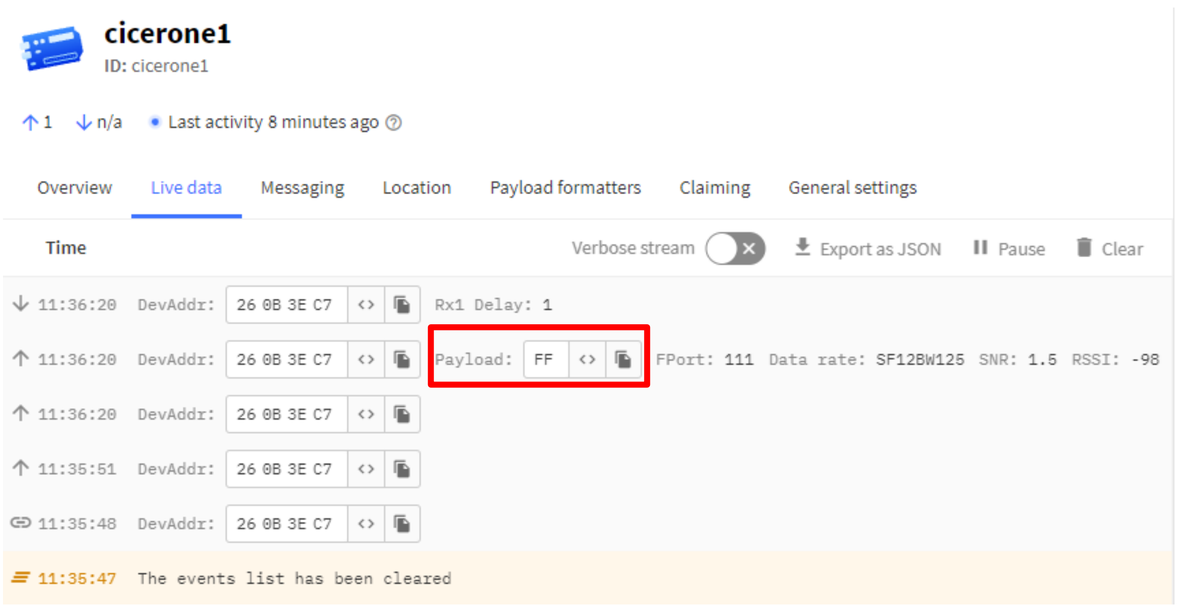
```
void onUsrBtn();
void onRx(uint8_t RxPort, uint8_t RxData[], uint8_t RxSize);

uint8_t devEUI[] = {0x00, 0x80, 0xE1, 0x15, 0x00, 0x28, 0xA6, 0x73};
uint8_t appEUI[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
uint8_t appKEY[] = {0x61, 0x12, 0x50, 0xF1, 0x78, 0x33, 0x3F, 0x12, 0xD6,
0xB5, 0xB3, 0xCB, 0x94, 0xCA, 0xB1, 0x39};

void setup()
{
  Serial.begin(9600);
```

```
11:17:19.348 -> Init LoRaWAN
11:17:27.715 -> Net Joined
11:18:39.523 -> Sending  Message(<Port> <Size> <Data HEX format>): 111 1 FF
```

- Serial output and network server live data for Join and uplink triggered by button pression


Output
Serial Monitor
Message (Ctrl + Enter to send message to 'Cicerone' on 'COM19'
New Line
9600 baud
PAYLOAD

```
11:35:47.675 -> Init LoRaWAN
11:35:56.076 -> Net Joined
11:36:20.029 -> Sending  Message(<Port> <Size> <Data HEX format>): 111 1 FF
```

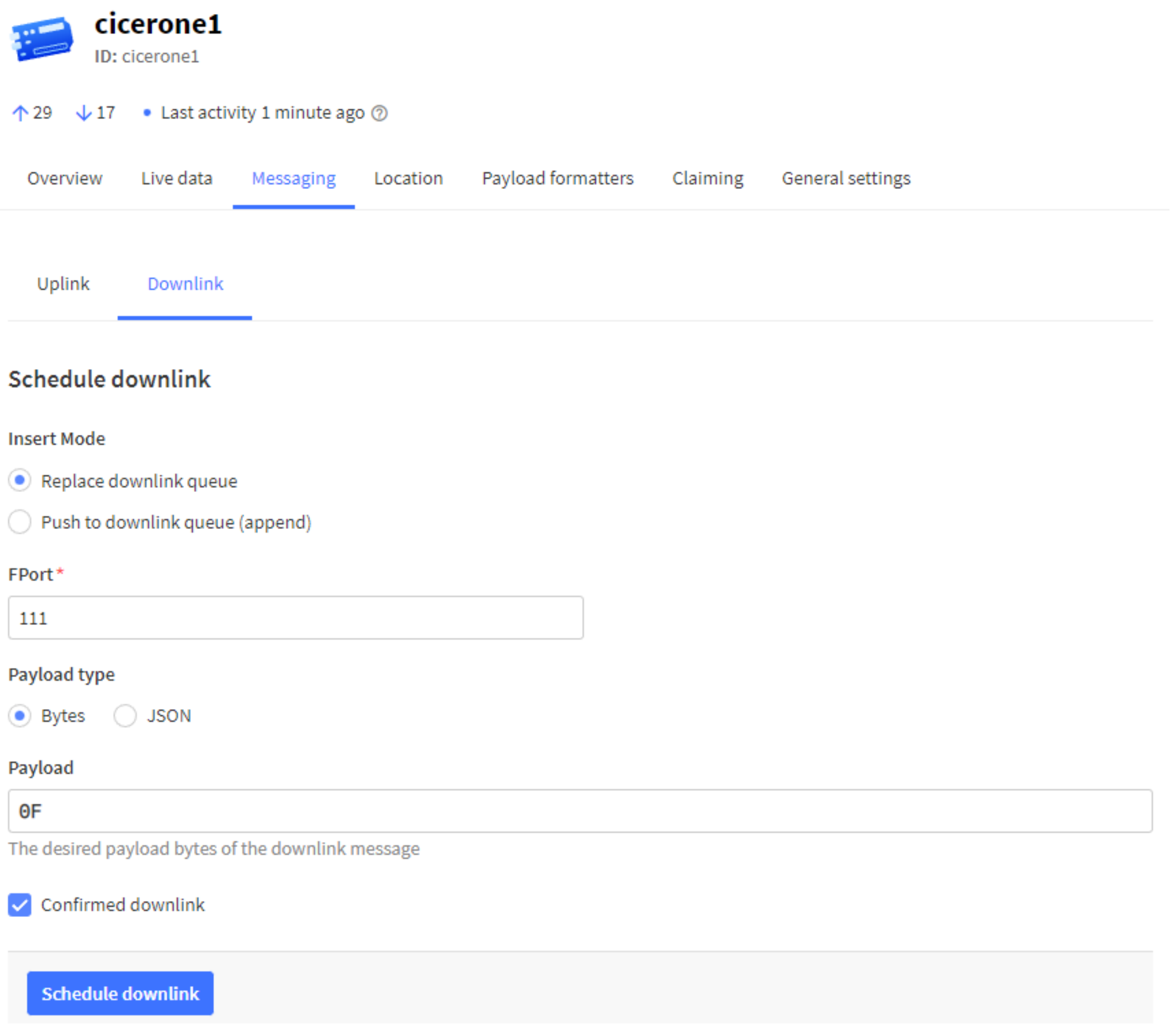


cicerone1
ID: cicerone1

↑1 ↓n/a • Last activity 8 minutes ago

```
Time
↓ 11:36:20  DevAddr: 26 0B 3E C7  Rx1 Delay: 1
↑ 11:36:20  DevAddr: 26 0B 3E C7  Payload: FF  FPort: 111  Data rate: SF12BW125  SNR: 1.5  RSSI: -98
↑ 11:36:20  DevAddr: 26 0B 3E C7
↑ 11:35:51  DevAddr: 26 0B 3E C7
  11:35:48  DevAddr: 26 0B 3E C7
  11:35:47  The events list has been cleared
```

- Schedule a downlink with payload 0x0F (15)


cicerone1
ID: cicerone1
↑29 ↓17 • Last activity 1 minute ago

Schedule downlink

Insert Mode
● Replace downlink queue
○ Push to downlink queue (append)

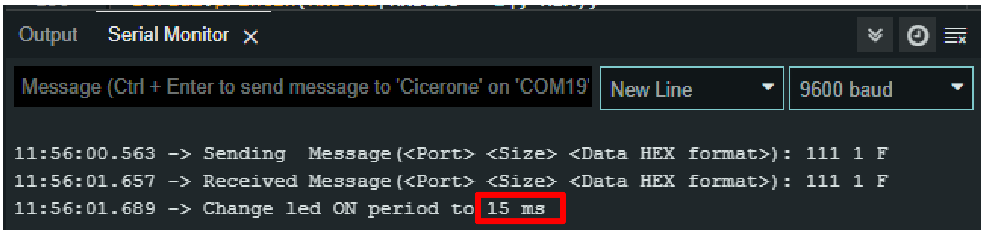
FPort*
111

Payload type
● Bytes ○ JSON

Payload
0F
The desired payload bytes of the downlink message

☑ Confirmed downlink

- Check received payload in the serial log


Output
Serial Monitor
Message (Ctrl + Enter to send message to 'Cicerone' on 'COM19'
New Line
9600 baud

```
11:56:00.563 -> Sending  Message(<Port> <Size> <Data HEX format>): 111 1 F
11:56:01.657 -> Received Message(<Port> <Size> <Data HEX format>): 111 1 F
11:56:01.689 -> Change led ON period to 15 ms
```

- With the new payload the onboard LED is on for 15 milliseconds (it was on for 255 milliseconds before)

Example: Low_Power_Mode

This example shows how to correctly use low power mode of MAMWLE module when using LoRaWAN library.

Low Power Mode

Most microcontrollers can be put in low-power mode to reduce power consumption. MAMWLE allows different modes, in order of consumption: RUN, Sleep, Low Power Run, Low Power Sleep, Stop0, Stop1, Stop2, Standby, Shutdown.
We will focus on Stop2 Mode as this is a compromise between low power and data retention.
Low Power Mode is entered every time no active processing is done. As the LoRaWAN stack is handled by a sequencer this one will put microcontroller in stop2 mode when no task has to be executed.

User Loop

One of the sequencer tasks is the user loop as seen in previous LoRaWAN_End_Node example. When calling the function LoRaWAN.attachLoop() with only one parameter (the pointer to the user loop function) the user loop function will be called as soon as the sequencer has no other tasks with higher priority. In this way there is no waiting time between two subsequent user loop executions. This approach does not allow microcontroller to be put in low power mode.

Typically, user wants to do some operations occasionally. In this case user loop has to be called every certain period time. To do so a second parameter must be specified in attachLoop(<function>, <period>), where <period> is in milliseconds.

In this example user loop function prints an incrementing counter each time is called, and then wait for two seconds to actively load the cpu.

```
void userloop(){
 count++;
 Serial.println("userloop() called " + String(count) + " times");
 delay(2000); // active cpu load (do not delay for more than 5 sec otherwise low power is never
entered)
}
```

Not retained Peripherals

One important aspect to consider is the peripheral retention. When microcontroller enters in low power mode not all the peripherals preserve their state/data so when exiting from low power they need a reinitialization. An example of not retained peripheral is the USART2, the one used for serial communication. If communication is handled in interrupt mode (non-blocking) the serial peripheral may be turned off before message is completely sent. To avoid these situations user can set some functions to be called just before and immediately after the low power mode request as in this example.

attachLPEnter()

This LibLoRaWAN function allows user to register a callback called just before entering low power mode. In this example it is used to flush content sent by serial

```
void onLPEnter(){
  Serial.print("*** Entering Low Power Mode...");
  Serial.flush(); // flush serial before entering STOP mode, otherwise print will be incomplete
}
```

attachLPExit()

This LibLoRaWAN function allows user to register a callback called immediately after exiting low power mode. In this examples it is used to reinitialize serial peripheral, automatically turned-off when low power mode is entered.

```
void onLPExit(){
  Serial.begin(BAUDRATE); // Need to reinit usart after STOP mode
  Serial.println("Exiting from Low Power Mode ***");
}
```

setup()

It is important to call attachLPEnter(), attachLPExit and attachLoop() before the LoRaWAN.begin()

User loop is called every 5 seconds.

```
void setup(){
 Serial.begin(BAUDRATE);

 LoRaWAN.attachLPEnter(onLPEnter);   // Callback on Low Power enter
 LoRaWAN.attachLPExit(onLPExit);     // Callback on Low Power exit
 LoRaWAN.attachLoop(userloop, 5000); // Call loop every 5 sec, sleep in the rest of time
 LoRaWAN.begin(true);
}
```

Testing

When entering low power mode, there is a relevant drop on the drawn current. Check this by using an amperometer in series on the power port used.

For Cicerone Board GNSS u-blox module has to be put in low power mode to maximize overall power consumption. Try Cicerone_Demo -> Deep_Sleep example.

Document revision

Revision 1.1 - 12/12/2022

Revision 1.0 - 16/9/2022