This repository contains LMIC-node, an example LoRaWAN application for a node for The Things Network. Get your node quickly up and running with LMIC-node.

LMIC-node

release v1.3.0 commits since v1.3.0 13

One example to rule them all

Contents

1 Introduction

LMIC-node is an example LoRaWAN application for a node that can be used with The Things Network. It demonstrates how to send uplink messages, how to receive downlink messages, how to implement a downlink command and it provides useful status information. With LMIC-node it is easy to get a working node quickly up and running. LMIC-node supports many popular (LoRa) development boards out of the box. It uses the Arduino framework, the LMIC LoRaWAN library and PlatformIO.

Basic steps to get a node up and running with LMIC-node:

- Select a supported board in `platformio.ini`.
- Select your LoRaWAN region in `platformio.ini`.
- Provide the LoRaWAN keys for your node (end device) in `lorawan-keys.h`.

- Compile and upload the firmware and you're ready to go!

1.1 What does it do?

- During startup LMIC-node sends some information to the serial port (if enabled) and display (if present and enabled). If OTAA activation is used LMIC-node will explicitly start a join to setup a network session.
- The main work like collecting input data and scheduling update messages is performed in the doWork job. This job runs at regular intervals (`DO_WORK_INTERVAL_SECONDS`).
- LMIC-node uses a counter to simulate a real sensor. The counter gets automatically incremented each time its value is read. The counter value is read by the doWork job and then transmitted via an uplink message to The Things Network (TTN).
- In addition LMIC-node also implements a downlink command to reset the counter. The command is sent to the node via a downlink message. When the 'reset counter' command is received by the node it will reset the counter value.
- While all this is happening LMIC-node outputs status information to the serial port for viewing on the serial monitor and also outputs information to the display (if present). The status information will show time, events (e.g. EV_JOINED, EV_TXCOMPLETE), uplink and downlink framecounters, RSSI and SNR of received downlink messages and when a downlink contains data the data will be displayed as bytes.
- The uplink messages can be viewed in the TTN Console. It is also possible to send the 'reset counter' downlink message from the console to the node. The effect of resetting the counter can be watched on the console as arriving uplink messages will show the new counter value.

Once the node is up and running you can start to explore and customize the source code to your own needs, e.g. add support for sensors. It is always easier to start with something that already works and then continue from there. Have fun!

1.2 Implemented features

- Send uplink messages.
- Receive downlink messages.
- Implements a downlink command.
- Provide detailed status feedback via serial port, LED and OLED display.
  Each of these output channels can be separately enabled/disabled (e.g. to save memory or power).
- Supports both OTAA and ABP activation. Simply switch via configuration option.
- LoRaWAN keys are placed in a separate file: `lorawan-keys.h`.
  - Protection against accidentally uploading LoRaWAN keys to public Git(Hub) repositories.
  - Easy to swap LoRaWAN keys for testing or for use with multiple devices.
- Requires only some configuration, but no programming or modification of source code to get it up and running.
- Configuration is done in a single project configuration file: `platformio.ini`.

- User modifiable code is clearly marked in the source code.
- Supports two different LMIC libraries: MCCI LoRaWAN LMIC library and IBM LMIC framework.
- Support for many popular (LoRa) development boards.
- Cross-platform, tested on STM32, SAMD21, ESP32, ESP8266, RP2040, MKL26Z64VFT4, ATmega32u4 and ATmega328 boards.
- Hardware dependencies are handled in separate Board Support Files (BSF).
- Built-in 'wait for serial port ready' with adjustable timeout and countdown visible on display. *Useful when using the serial monitor with boards with MCU with integrated USB support.*
- Abstraction of the serial port so code can print to `serial` without needing to know if it must print to `Serial` or `SerialUSB`.
- Use correct GPIO pins for onboard LED, display, LoRa hardware, I2C and SPI ports even if these are incorrectly defined in the BSP.
- Avoid hardware conflicts i.e. GPIO's shared between onboard LED, I2C, SPI and/or Vext.
- Explicitly initialize I2C and SPI interfaces with correct pins if default pins are incorrectly defined in the BSP.
- Select the proper subband for regions US915 and AU915.
- For LMIC debugging, for each board, LMIC_PRINTF_TO is defined for the correct serial port. *No need to set the `LMIC_PRINTF_TO` parameter to `Serial` or `SerialUSB` manually.*

1.3 Requirements

The following is required to use LMIC-node:

- **A supported** LoRa **development board** or development board with external Semtech SX127*x*, HopeRF RFM9*x* or HPDTek HPD1*x*A SPI LoRa module.
- **A computer with PlatformIO installed**. PlatformIO is used instead of Arduino IDE because more flexible, more powerful and it better supports cross-platform development. For installation see PlatformIO installation instructions.
- **Internet connection**. PlatformIO will automatically download and install packages (Arduino cores, toolchains and libraries) as needed.
  If PlatformIO is freshly installed the downloading may take some time. Once installed it will be possible to work offline.
- **USB cable**. For boards without onboard USB also a **USB to serial adapter** is needed (for STM32 optionally a STLink programmer).
- **Wiring**: Some LoRa development boards require manual wiring of LoRa DIO1 (see table below). When using a development board with external LoRa module then everything must be manually wired. For wiring details see the board's Board Support File (BSF).
- **Node registration**: A node (end device) must be created/registered in The Things Network (TTN) Console before it can be used. The LoRaWAN keys for the device must be copied from the TTN Console to file `lorawan-keys.h`. Registration is not further described here. For more information see

The Things Network documentation, The Things Stack documentation and The Things Network Forum.
- **Skills**: You should already be familiar with compiling and uploading basic Arduino sketches to your board and how to use a serial monitor.

Display:
LMIC-node supports the following display type: SSD1306 128x64 I2C OLED. These are the displays used on Heltec Wifi LoRa 32 and TTGO LoRa32 boards. When connecting an external display use this type. Use of other I2C displays is possible but requires modification of LMIC-node which is not further described here.

Not yet a requirement but this document assumes that you will be using The Things Network V3 as The Things Network V2 will cease operation by the end of 2021 and should not be used for new development.

2 Supported Boards

The following tables list the boards currently supported by LMIC-node.

Explanation of columns: MCU: microcontroller. Wiring required: yes means manual wiring of DIO1 is required. USB: has onboard USB. LED: yes: has onboard LED *and* is usable (no hardware conflicts). Display: yes means has onboard display. Board-id: board identifier as used by LMIC-node.

2.1 LoRa development boards

The following LoRa development boards have onboard LoRa support. Most have onboard USB that supports automatic firmware upload and serial over USB for serial monitoring. Some boards require manual wiring of the LoRa DIO1 port. For boards without onboard display an external display can be optionally connected. For details and wiring instructions see the board's BSF.

Board name	MCU	Wiring required	USB	LED	Display	Board-id
Adafruit Feather M0 RFMx LoRa	SAMD21G18	yes *1*	yes	yes	no	adafruit_feather_m0_lora
ST B-L072Z-LRWAN1 Discovery kit	STM32L072CZ	no	yes	yes	no	discovery_l072z_lrwan1
Heltec WiFi LoRa 32 V2	ESP32	no	yes	yes	yes	heltec_wifilora32_v2
Heltec WiFi LoRa 32 (1.x)	ESP32	no	yes	yes	yes	heltec_wifilora32
Heltec Wireless Stick	ESP32	no	yes	yes	yes *7	heltec_wireless_stick
Heltec Wireless Stick Lite	ESP32	no	yes	yes	no	heltec_wireless_stick_lite
Pycom LoPy4	ESP32	no	no *4*	no *5*	no	lopy4
BSFrance LoRa32u4 II *versions v1.0, v1.1, v1.2 and v1.3*	ATmega32u4	yes *2*	yes	yes	no	lora32u4II
TTGO LoRa32 V1.3	ESP32	no	yes	no	yes	ttgo_lora32_v1
TTGO LoRa32 V2.0	ESP32	yes *3*	yes	no *6*	yes	ttgo_lora32_v2
TTGO LoRa32 V2.1.6	ESP32	no	yes	no	yes	ttgo_lora32_v21
TTGO T-Beam *versions v0.5, v0.6 and v0.7*	ESP32	no	yes	yes	no	ttgo_tbeam
TTGO T-Beam *versions v1.0 and v1.1*	ESP32	no	yes	no	no	ttgo_tbeam_v1

**1*: DIO1 must be manually wired to GPIO6.
**2*: For versions 1.0, 1.1 and 1.2 DIO1 must be manually wired to GPIO5 (version 1.3 is already wired on the PCB).
**3*: DIO1 must be manually wired to GPIO33.
**4*: Requires USB to Serial adapter or Pycom Expansion Board which is explained further below.
**5*: Has onboard Neopixel RGB LED but is currently not supported by LMIC-node.
**6*: Either display (I2C) or LED can be used but not both at the same time. LED is default disabled.
**7*: Display (64x32) not supported by LMIC-node because resolution is too small.

2.2 Development boards with external SPI LoRa module

The following development boards require an external Semtech SX127*x*, HopeRF RFM9*x* or HPDTek HPD1*x*A SPI LoRa module *(Semtech SX1262 is currently not supported by the LMIC library)*. Most boards have onboard USB that supports automatic firmware upload and serial over USB for serial monitoring. An external display can be optionally connected. For details and wiring instructions see the board's BSF.

Board name	MCU	Wiring required	USB	LED	Display	Board-id
Adafruit QT PY	SAMD21E18	yes	yes	no *5*	no	adafruit_qt_py_m0
Black Pill STM32F103C8 128k	STM32F103C8T6	yes	yes *8*	yes	no	blackpill_f103c8_128k
Black Pill STM32F103C8 64k	STM32F103C8T6	yes	yes *8*	yes	no	blackill_f103c8
Blue Pill STM32F103C8 128k	STM32F103C8T6	yes	yes *8*	yes	no	bluepill_f103c8_128k
Blue Pill STM32F103C8 64k	STM32F103C8T6	yes	yes *8*	yes	no	bluepill_f103c8
Lolin D32 Pro	ESP32	yes	yes	yes	no	lolin_d32_pro
Lolin D32	ESP32	yes	yes	yes	no	lolin_d32
Lolin32	ESP32	yes	yes	yes	no	lolin32
NodeMCU-32S	ESP32	yes	yes	yes	no	nodemcu_32
NodeMCU V2 (aka v1.0)	ESP8266	yes	yes	yes	no	nodemcuv2
Raspberry Pi Pico	RP2040	yes	yes	yes	no	pico
Arduino Pro Mini (ATmega328 8mHz)	ATmega328	yes	yes	no	no	pro8mhzatmega328
SAMD21 M0-Mini	SAMD21G18	yes	yes	no	no	samd21_m0_mini
Teensy LC	MKL26Z64VFT4	yes	yes	yes	no	teensylc

5: Has onboard Neopixel RGB LED but is currently not supported by LMIC-node.
8: These boards have onboard USB but by default do not support firmware upload over USB or serial over USB. For upload use a STLink programmer or USB to serial adapter.

3 Details

This chapter contains detailed information about LMIC-node.

3.1 setup() function

During setup the following components and any corresponding libraries are initialized: serial port, display, LED, LoRa pin mappings, I2C, SPI, LMIC. Additionally other hardware like sensors and their libraries can be initialized (LMIC-node uses a counter to simulate a sensor).

The first step in `setup()` is:

```
// boardInit(InitType::Hardware) must be called at start of setup() before anything else.
bool hardwareInitSucceeded = boardInit(InitType::Hardware);
```

The `boardInit()` function is defined in the board's Board Support File.
If OTAA activation is used then during setup a join will be explicitly started to establish a connection with the network.
The last step in `setup()` is scheduling the first run of the `doWork` job.

3.2 doWork job

The `doWork` job runs every `DO_WORK_INTERVAL_SECONDS`.
To run the job the `doWorkCallback()` function is executed by the LMIC scheduler.
`doWorkCallback()` calls the `processWork()` function where the actual work is performed.
The first `doWork` run is started in `setup()`. On completion `doWork` reschedules itself for the next run.

When the node has joined and the `EV_JOINED` event is handled by the event handler, the next scheduled doWork job is cancelled and is re-scheduled for immediate execution. This is done to prevent that any uplink will have to wait until the current doWork interval ends. `processWork()` skips doing any work while the node is still joining. As a result sending the first uplink message may have to wait until the current doWork interval ends (max `DO_WORK_INTERVAL_SECONDS` seconds). Directly running the doWork job after a join prevents the in this case unnecessary and unwanted delay.

3.3 processWork() function

The `processWork()` function contains user code that performs the actual work like reading sensor data and scheduling uplink messages. In LMIC-node `processWork()` will skip doing any work if the node is still joining for two reasons:

1. To prevent unnecessary incrementing of the counter.
2. Uplink messages cannot yet be sent.

3.4 processDownlink() function

The `processDownlink()` function contains user code for processing a downlink message. `processDownlink()` is called from the event handler function when an EV_TXCOMPLETE event is handled and a downlink message was received.

3.5 Uplink messages

The counter is implemented as an unsigned 16 bit integer. The uplink payload consists of the counter value, 2 bytes in msb format (most significant byte first). The frame port number used for uplink messages is 10. Port 10 is used to demonstrate that other port numbers than the default 1 can be used.

3.6 Downlink messages

There are two types of downlink messages. Downlink messages containing user data and downlink messages containing MAC commands. MAC commands are sent by the network server to set or query network related settings.

When a downlink message is received its RSSI and SNR values will be displayed as well as the port number. If the port number is 0 and no data is shown then a MAC command was received.

If the port number is greater than 0 and user data was received the data will be displayed as a sequence of byte values. Contents of downlink data will only be output to the serial port and not to the display because the display is too small to fit all information on a single screen.

3.6.1 Reset-counter downlink command

The reset-counter downlink uses 100 as frame port number. The reset command is represented by a single byte with hex value 0xC0 (for Counter 0). When a downlink message is received on port 100, the length of the data is 1 byte and the value is 0xC0 then the `resetCounter()` function will be called and the counter will be reset to 0. If the received payload data is longer than a single byte then the reset-counter command will not be performed.

3.7 Status information

The following status information is shown:

3.7.1 Serial port and display

At the start:

- Timeout countdown value while waiting for serial port to become ready is shown on the display. This value is only shown for boards where `WAITFOR_SERIAL_SECONDS_DEFAULT` is defined in the BSF with a value unequal to 0.

In the header:

- Program title "LMIC-node".

- Device-id.
- Used LMIC library. MCCI or Classic. On display only an asterisk will be shown if Classic is used.
- If OTAA or ABP activation is used. On display only ABP will be shown if used.
- LMIC debug level if > 0. *(on serial port only)*
- DoWork job time interval (DO_WORK_INTERVAL_SECONDS). On display 'interval' label is abbreviated to "I".

When joined:

- Network-id. *(on serial port only, MCCI LMIC only)*
- Device address. *(on serial port only, MCCI LMIC only)*
- Sessions keys *(son erial port only, MCCI LMIC only)*

Continuously:

- During transmission of uplink and downlink messages a transmit symbol will be shown in the top-right of the display. the transmit symbol and LED are called transmit indicators. For MCCI LMIC transmit indicators are set to on while handling the EV_TXSTART event. Classic LMIC does not generate this event however. For Classic LMIC the transmit indicators will be set to on after an uplink has been successfully scheduled (which is less precise).
- A notification when the doWork job is started. *(on serial port only)*
- A notification when input data was collected *(on serial port only)*
- Message with Counter value. On display label is abbreviated to "Ctr".
- A notification "packet queued" when an uplink message is scheduled.
- A notification if a packet cannot be scheduled because transmission or reception of another message is still pending.
- An error message if scheduling of an uplink message failed.
- Events generated by the LMIC library (e.g. EV_JOINED, EV_TXCOMPLETE).
- When event EV_TXCOMPLETE is received the frame counters are printed.
- For each downlink message the following will be shown:
  - Message that downlink was received
  - RSSI and SNR values
  - Frame port number
  - If data was received, length of data
  - If data was received, the data is shown as as byte *(serial port only)*
  - Message if reset-counter command was received *(serial port only*)
- Notification when counter is reset.

For events and notifications a timestamp (`ostime`) will be shown. LMIC uses values of the type ostime_t to represent time in ticks. The rate of these ticks defaults to 32768 ticks per second (but may be configured at compile time to any value between 10000 ticks per second and 64516 ticks per second).

3.7.2 LED

The LED is a transmit indicator similar to the transmit symbol on the display. For a description when the LED is on and off see the description for the transmit symbol above.

3.8 User modifiable code

LMIC-node will work out of the box without having to do any programming or modifying of source code. However, LMIC-node will only do a few tricks: Send counter value via uplink messages, handle reset-counter downlink command and show detailed status information.

To make LMIC-node do other, more useful things e.g. reading values from a temperature and humidity sensor or from a water-level sensor and sending these via uplink messages, requires modifying and extending the source code.

Most of the source code is boiler plate code that does not need to be changed. Code for things like adding support for sensors or implement other downlink commands is called *User Code*. In `LMIC-node.cpp` three sections in the source code are marked as *User Code*. In `platformio.ini` one section is marked for User Code where additional libraries for User Code can be added. Try to put your user code in these sections (this will prevent accidentally messing things up).

If you are aware of what you are doing you are of course free to change every single line of code to your needs, but if this is new to you it might be safer to restrict modifications to the user code sections.

Reading sensors (etc), preparing uplink payload and scheduling an uplink message for transmission can be done in function `processWork()`. Handling of downlink messages and adding your own downlink commands can be done in function `processDownlink()`.

The User Code sections in `LMIC-node.cpp` are marked as follows:

```
//  █ █ █▀▀ █▀▀ █▀█   █▀▀ █▀█ █▀▄ █▀▀   █▀▄ █▀▀ █▀▀ █ █▀█
//  █ █ ▀▀█ █▀▀ █▀▄   █   █ █ █ █ █▀▀   █▀▄ █▀▀ █ █ █ █ █
//  ▀▀▀ ▀▀▀ ▀▀▀ ▀ ▀   ▀▀▀ ▀▀▀ ▀▀  ▀▀▀   ▀▀  ▀▀▀ ▀▀▀ ▀ ▀ ▀


const uint8_t payloadBufferLength = 4;    // Adjust to fit max payload length


//  █ █ █▀▀ █▀▀ █▀█   █▀▀ █▀█ █▀▄ █▀▀   █▀▀ █▀█ █▀▄
//  █ █ ▀▀█ █▀▀ █▀▄   █   █ █ █ █ █▀▀   █▀▀ █ █ █ █
//  ▀▀▀ ▀▀▀ ▀▀▀ ▀ ▀   ▀▀▀ ▀▀▀ ▀▀  ▀▀▀   ▀▀▀ ▀ ▀ ▀▀
```

The User code section in `platformio.ini` is marked as follows:

```
;   ▀▀▀ Add additional libraries for User Code below this line ▀▀▀
```

Names of libraries needed for User Code can be specified below above line.

3.9 Board-id

LMIC-node uses a *board-id* to identify a specific type of board. *board-id* is similar to PlatformIO's *board* but latter is limited to BSP's defined in Arduino cores. Unfortunately there does not exist a dedicated BSP for each board supported by LMIC-node. Therefore LMIC-node defines its own board identifiers.

board-id is kept identical or as similar as possible to PlatformIO's *board*. For simplicity and consistency *board-id* only uses underscores and lowercase characters (e.g. `heltec_wifi_lora_32_v2` and `ttgo_lora32_v2`) while PlatformIO's *board* uses hyphens, underscores and mixed case characters (e.g. `heltec_wifi_lora_32_V2` and `ttgo-lora32-v2`). Board-id `lora32u4II` is an exception, the `II` is uppercase because it is a Roman numeral.

If a proper *board* definition and BSP for some version of a development board do currently not exist then LMIC-node defines its own board with its own *board-id*. In which case it will use the closest matching *board* in PlatformIO and add a separate BSF for *board-id* where the differences can be properly handled. In this case *board-id* will be the same as *board* but suffixed with a version e.g. `ttgo_t_beam_v1`. In this example there only exists a board `ttgo-t-beam` in PlatformIO but no `ttgo-t-beam-v1` while there are important hardware differences between v0.x and v1.x versions of these boards.

3.10 Device-id

LMIC-node uses a device-id to identify a device. The device-id is used for display purposes in status information that is output to the serial port and display. Device-id's allow different devices to be easily recognized when they have different device-id's, because their device-id is shown on the display (if present) and/or serial monitor (serial port). The length of a device-id should be limited to max 16 characters long, otherwise it will not fit on a display.

When creating a device in the TTN console one must specify a unique device identifier for the device. The device-id used in LMIC-node is only used for display purposes (and is not the same as the device identifier in the TTN Console) but it is useful to use the same device-id for LMIC-node as the device identifier in the TTN console (or at least make it similar). That will make it easier to recognize traffic in the TTN console because the device identifier displayed on the console and the device-id displayed on the node's display (or serial monitor) will match.

In the BSF a default device-id (`DEVICEID_DEFAULT`) is defined per board type. The default device-id can be overriden by defining `DEVICEID` in file `lorawan-keys.h`, or defining `ABP_DEVICEID` in `lorawan-keys.h`. If defined latter will be used when ABP activation is used.

`lorawan-keys.h` can contain both the keys used for OTAA activation as well as the keys used for ABP activation. Keys used for OTAA and ABP are different and they have different names. Only the keys for the used actvation type (OTAA or ABP) need to be specified.

Tip: For testing purposes it is possible to create two different devices in the TTN console for the same hardware device, one for OTAA activation and the other for ABP activation. Both sets of keys can be added to lorawan-keys.h and for both a different device-id can be added to lorawan-keys.h. This way a single hardware device can be used for both OTAA and ABP (only one at a time). All that to needs to be done to switch the device from OTAA to ABP is to enable the `-D ABP_ACTIVATION` setting in the `[common]` section in `platformio.ini` (or vice versa) and then recompile and upload the firmware.

3.11 platformio.ini

`platformio.ini` is the project configuration file. This file contains all configuration settings like program options, names of libraries and build options. It contains below sections:

- **[platformio]**
  Contains a list of board names and their corresponding board-id ('environment definitions').
  This is used to select a board (by uncommenting the line with the board-id to be selected).

- **[common]**
  Contains settings used for all boards.

- **[mcci_lmic]**
  Contains MCCI LoRaWAN LMIC library specific settings.
  MCCI LoRaWAN LMIC library is used for boards with 32-bit MCU.

- **[classic_lmic]**
  Contains IBM LMIC framework library specific settings.
  IBM LMIC framework is used for boards with 8-bit AVR MCU.
  These boards have less available memory and Classic LMIC uses less memory resources than MCCI LMIC.
  It is not possible to use LMIC-node with MCCI LMIC on these boards due to insufficient memory.
  Be aware that Classic LMIC is not fully LoRaWAN compliant and should not be used with ABP activation on The Things Network V3. IBM LMIC framework (Clasic LMIC) was recently deprecated and is not further maintained.

- **[env:<*board-id*>]**
  Board specific sections (called *Environments* in PlatformIO terminology). E.g. [env:lolin32]. A board specific section contains settings that are specific for one type of board. It contain settings like which LMIC library to use and program options like USE_SERIAL, USE_LED and USE_DISPLAY.

Comments in platformio.ini start with a semicolon (;) character. To uncomment a line remove the semicolon prefix. To comment a line add a semicolon prefix.

3.12 lorawan-keys.h

File `lorawan-keys.h` contains the LoRaWAN keys for a node. The keys are placed in a separate file for:

- Protection against accidentally uploading LoRaWAN keys to public Git(Hub) repositories.

- Prevent that source code needs to be modified for changing keys.
- Make it easy to swap LoRaWAN keys for testing purposes or for use with multiple devices.

The keys use three different formats (lsb, msb and uint32_t). lsb: least significant byte, msb: most significant byte. The TTN console provides options to copy keys in different formats. Use the correct formats as shown in below extract from `lorawan-keys_example.h`:

```
// Optional: If DEVICEID is defined it will be used instead of the default defined in the BSF.
// #define DEVICEID "<deviceid>"

// Keys required for OTAA activation:

// End-device Identifier (u1_t[8]) in lsb format
#define OTAA_DEVEUI 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

// Application Identifier (u1_t[8]) in lsb format
#define OTAA_APPEUI 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

// Application Key (u1_t[16]) in msb format
#define OTAA_APPKEY 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00


// -----------------------------------------------------------------------------

// Optional: If ABP_DEVICEID is defined it will be used for ABP instead of the default defined
in the BSF.
// #define ABP_DEVICEID "<abp-deviceid>"

// Keys required for ABP activation:

// End-device Address (u4_t) in uint32_t format.
// Note: The value must start with 0x (current version of TTN Console does not provide this).
#define ABP_DEVADDR 0x00000000

// Network Session Key (u1_t[16]) in msb format
#define ABP_NWKSKEY 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00

// Application Session K (u1_t[16]) in msb format
#define ABP_APPSKEY 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00
```

The `lorawan-keys.h` file is located in folder `keyfiles`.

All files with name pattern `*lorawan-keys.h` and all files in folder `keyfiles` (except for file `lorawan-keys_example.h`) are excluded from the Git(Hub) repository (defined in file `.gitignore`) to prevent that LoRaWAN keys will be accidentally committed to a public repository.

`loarawan-keys_example.h` is included as example for `lorawan-keys.h`.

3.13 Board Support Files (BSF)

A Board Support File (BSF) isolates hardware dependencies for a board into a single file: the BSF. There is a separate BSF per board type.

A Board Support Package (BSP) provides support for a specific board and provides standard pin definitions for a board. BSP's are part of an Arduino core, which contains a BSP for each supported board.

A BSF acts like a mini Board Support Package (BSP). A BSF adds functionality on top of the BSP and can also contain corrections if pins are incorrectly defined in the BSP.

A BSF has the same name as the board-id, is prefixed with `bsf_` and suffixed with `.h` extension. BSF's are located in the `src/boards` folder.
Exceptions:

- `lora32u4II` the `II` is uppercase because it is a Roman numeral.
- `blackpill_f103c8_128k` and `bluepill_f103c8_128k` use the same BSF as their 64k versions.

A BSF contains important information about a board (including connection/wiring details), contains a URL to PlatformIO's documentation for the board and provides the following functionality:

- Defines the default device-id.
- Defines 'wait for serial port' timeout default value (if needed).
- Defines if LMIC timing should be relaxed (if needed).
- Defines a reference named `serial` that points to the correct serial port (Serial or SerialUSB).
- Defines and initializes the `led` object.
- Defines the `display` object.
- Defines the `lmic_pins` pin mapping structure used by the LMIC library.
- Contains the `boardInit()` function.
- Uses correct GPIO pins for onboard LED, display, LoRa hardware, I2C and SPI ports even if these are incorrectly defined in the BSP in the Arduino core.
- Avoids hardware conflicts i.e. GPIO's shared between onboard LED, I2C, SPI and/or Vext.

The `boardInit(initType)` function provides a generic mechanism for performing custom hardware initialization. It is called twice during setup(). The `initType` parameter indicates which initialization code must be executed.

- `InitType::Hardware`:
    - Explicitly initializes I2C and SPI interfaces with correct pins if default pins are incorrectly defined in the BSP (if needed).
    - Special hardware initialization (if needed, e.g. initializing T-Beam v1.x power management chip).
- `InitType::PostInitSerial`:

- Inserts a delay after serial port initialization to prevent losing the first output printed to the serial port (if needed).

Example of the `boardInit()` function for a board that does not require custom hardware initialization:

```
bool boardInit(InitType initType)
{
    // This function is used to perform board specific initializations.
    // Required as part of standard template.

    // InitType::Hardware        Must be called at start of setup() before anything else.
    // InitType::PostInitSerial  Must be called after initSerial() before other initializations.

    bool success = true;
    switch (initType)
    {
        case InitType::Hardware:
            // Note: Serial port and display are not yet initialized and cannot be used use
here.
            // No actions required for this board.
            break;

        case InitType::PostInitSerial:
            // Note: If enabled Serial port and display are already initialized here.
            // No actions required for this board.
            break;
    }
    return success;
}
```

3.14 Payload formatters

Payload formatter functions are located in the `payload-formatters` folder.

3.14.1 Uplink decoder

LMIC-node comes with a JavaScript payload formatter function for decoding the uplink messages so the counter value gets displayed in 'Live data' on the TTN Console. The `decodeUplink()` function can be found in folder `payload-formatters` in file `lmic-node-uplink-formatters.js`.

```
function decodeUplink(input) {
    var data = {};
    var warnings = [];

    if (input.fPort == 10) {
        data.counter = (input.bytes[0] << 8) + input.bytes[1];
    }
    else {
        warnings.push("Unsupported fPort");
    }
    return {
        data: data,
        warnings: warnings
    };
}
```

In the TTN Console this function should be added to the device (or application) as uplink payload formatter function. When this function is installed, the counter value will become visible in uplink messages in 'Live data' on the TTN Console.

3.15 External libraries

LMIC-node uses the following external libraries:

Library name	Category	Repository URL
MCCI LoRaWAN LMIC library	LoRaWAN	https://github.com/mcci-catena/arduino-lmic
IBM LMIC framework	LoRaWAN	https://github.com/matthijskooijman/arduino-lmic
U8g2	Display	https://github.com/olikraus/u8g2
EasyLed	LED	https://github.com/lnlp/EasyLed

4 Settings

4.1 Board selection

In platformio.ini the board must be selected. Select only a single board by uncommenting the line with its board-id. Comment the line starting with "<platformio.ini board selector guard>". The guard is explicitly added to prevent that PlatformIO will compile LMIC-node for ALL listed boards, when no board is selected.

Warnings:

- When the guard is disabled and no board is selected then PlatformIO will compile for ALL listed boards!
- The Serial Monitor can only be started when a board has been selected in `platformio.ini`.

```
[platformio]
default_envs =
    <platformio.ini board selector guard> Comment this line and uncomment one board-id below:

    ; LoRa development boards with integrated LoRa support:

    ; Board-id                            Board name
    ;---------                            ----------
    ; adafruit_feather_m0_lora          ; Adafruit Feather M0 LoRa
    ; disco_l072cz_lrwan1               ; Discovery B-L072Z-LRWAN1
    ; heltec_wifi_lora_32_v2            ; Heltec Wifi LoRa 32 V2
    ; heltec_wifi_lora_32               ; Heltec Wifi LoRa 32
    ; heltec_wireless_stick_lite        ; Heltec Wireless Stick Lite
    ; heltec_wireless_stick             ; Heltec Wireless Stick
    ; lopy4                             ; Pycom Lopy4
    ; lora32u4II                        ; BSFrance LoRa32u4 II v1.0, v1.1, v1.2, v1.3
    ; ttgo_lora32_v1                    ; TTGO LoRa32 v1.3
    ; ttgo_lora32_v2                    ; TTGO LoRa32 v2.0
    ; ttgo_lora32_v21                   ; TTGO LoRa32 v2.1.6
    ; ttgo_t_beam                       ; TTGO T-Beam v0.5, v0.6, v0.7
    ; ttgo_t_beam_v1                    ; TTGO T-Beam v1.0, v1.1

    ; Development boards that require an external SPI LoRa module:

    ; Board-id                            Board name
    ;---------                            ----------
    ; adafruit_qt_py_m0                 ; Adafruit QT Py
    ; blackpill_f103c8_128k             ; Black Pill 128k
    ; blackpill_f103c8                  ; Black Pill  64k
    ; bluepill_f103c8_128k              ; Blue Pill 128k
    ; bluepill_f103c8                   ; Blue Pill  64k
    ; lolin_d32_pro                     ; Lolin D32 Pro
    ; lolin_d32                         ; Lolin D32
    ; lolin32                           ; Lolin32
    ; nodemcu_32s                       ; NodeMCU-32S
    ; nodemcuv2                         ; NodeMCU V2
    ; pico                              ; Raspberry Pi Pico
    ; pro8mhzatmega328                  ; Arduino Pro Mini 3.3V 8Mhz
    ; samd21_m0_mini                    ; SAMD21 M0-Mini
    ; teensylc                          ; Teensy LC
```

4.2 Common settings

```
[common]

monitor_speed = 115200

build_flags =
    -D DO_WORK_INTERVAL_SECONDS=60

    ; -D ABP_ACTIVATION                ; Use ABP instead of OTAA activation
    ;
    ; -D WAITFOR_SERIAL_SECONDS=10     ; Can be used to override the default value (10)
    ;                                    Only used for boards with default set to != 0 in BSF
    ;
    ; -D STM32_POST_INITSERIAL_DELAY_MS=1500  ; Workaround for STM32 boards. Can be used
    ;                                           to override value (milliseconds) in BSF

lib_deps =
    olikraus/U8g2                      ; OLED display library
    lnlp/EasyLed                       ; LED library
;   ██ Add additional libraries for User Code below this line ██
```

monitor_speed
Sets the monitor speed for the serial port. 115200 bps should be fine for most purposes and does not need to be changed.

DO_WORK_INTERVAL_SECONDS
Defines the interval for when the doWork job runs where the actual work is done. Be aware that this is also the interval that uplink messages will be sent. The interval should not exceed TTN's fair use policy and regulatory constraints.

ABP_ACTIVATION
If enabled will use ABP activation instead of OTAA activation (default).

WAITFOR_SERIAL_SECONDS
Can be used to overrule the default value defined in BSF for testing purposes but normally not needed to change this. This setting only has effect for boards where a default value (>0) is already defined and will not have effect for other boards. 'Wait for serial' only is useful when USB functionality is provided by the MCU.

A 'wait for serial' delay of 10 seconds is configured by default in boards where USB support is integrated into the MCU (instead of using onboard USB to serial). Waiting until the serial (over USB) port is actually ready (via 'wait for serial') prevents the first output printed to the serial port getting lost.

A positive value will wait with a countdown delay (visible on display) for the number of seconds specified. A value of 0 will not wait. A value of -1 will wait indefinitely.

Be aware that the serial port must not only be ready but also a serial monitor needs to be connected. The countdown gives some time to start the serial monitor if not already started. A timeout value prevents that the node waits indefinitely if not connected to a computer. It will continue when the countdown ends.

STM32_POST_INITSERIAL_DELAY_MS For STM32 boards a delay is inserted after initializing the serial port. This is a workaround to prevent that the first output send to the serial port gets lost. This value is defined in STM32 boards BSF but can be overridden in platformio.ini (the override option was added for testing purposes).

4.3 LoRaWAN library settings

4.3.1 MCCI LoRaWAN LMIC library settings

```
[mcci_lmic]
; LMIC-node was tested with MCCI LoRaWAN LMIC library v3.3.0.
; Some changes have been announced for future versions of the MCCI library
; which may be incompatible with LMIC-node. In case of problems just
; use mcci-catena/MCCI LoRaWAN LMIC library@3.3.0 below which will
; explicitly use v3.3.0 of the library.
; Perform PlatformIO: Clean after changing library version and
; in case of issues remove the old version from .pio/libdeps/*.

; Note: LMIC_PRINTF_TO is defined for each board separately
;       in the board specific sections. Don't define it in this section.

lib_deps =
    ; Only ONE of below LMIC libraries should be enabled.
    mcci-catena/MCCI LoRaWAN LMIC library           ; MCCI LMIC library (latest release)
    ; mcci-catena/MCCI LoRaWAN LMIC library@3.3.0   ; MCCI LMIC library v3.3.0

build_flags =
    ; Use platformio.ini for settings instead lmic_project_config.h.
    -D ARDUINO_LMIC_PROJECT_CONFIG_H_SUPPRESS

    ; Ping and beacons not supported for class A, disable to save memory.
    -D DISABLE_PING
    -D DISABLE_BEACONS

    ; -D LMIC_DEBUG_LEVEL=1            ; 0, 1 or 2

    ; -D CFG_sx1272_radio=1            ; Use for SX1272 radio
    -D CFG_sx1276_radio=1              ; Use for SX1276 radio
    -D USE_ORIGINAL_AES                ; Faster but larger, see docs
    ; -D LMIC_USE_INTERRUPTS           ; Not tested or supported on many platforms
    ; -D LMIC_ENABLE_DeviceTimeReq=1   ; Network time support

    ; --- Regional settings -----
    ; Enable only one of the following regions:
    ; -D CFG_as923=1
    ; -D CFG_as923jp=1
    ; -D CFG_au915=1
    ; -D CFG_cn490=1                   ; Not yet supported
    ; -D CFG_cn783=1                   ; Not yet supported
    ; -D CFG_eu433=1                   ; Not yet supported
    -D CFG_eu868=1
    ; -D CFG_in866=1
    ; -D CFG_kr920=1
    ; -D CFG_us915=1
```

Regional setting

--- Regional settings --- is where the regional setting needs to be selected. Uncomment the line with the appropriate region setting (and comment the other region settings). Only one region may be selected. By default CFG_eu868 (Europe) is selected.

LMIC_DEBUG_LEVEL
This can be uncommented to allow debug information from the LMIC library to be printed. Possible values are 0, 1, 2 and 3 where 0 provides no debugging information and 3 provides the most information.
Be aware that enabling debug will increase memory requirements.

Other settings are listed for information but are not further explained here. For more information see the MCCI LoRaWAN LMIC library documentation.

4.3.2 IBM LMIC framework settings

```
[classic_lmic]
; IMPORTANT:
; This library was recently DEPRECATED and is no longer maintained.
; It is not fully LoRaWAN compliant (e.g. in handling of MAC commands)
; and is therefore less suitable for use with The Things Network V3.
;
; Region, radio and debug settings CANNOT be changed in platformio.ini.
; They must be configured in file: config.h in the following location:
; .pio/libdeps/<board-id>/IBM LMIC framework/src/lmic
;
; When making changes to config.h:
; CONFIG.H MUST BE CHANGED FOR EACH BOARD SEPARATELY!
; (By default libraries are installed per project per build config/board.)

lib_deps =
    matthijskooijman/IBM LMIC framework   ; [Deprecated] Classic LMIC library

build_flags =
    ; DEFAULT VALUES defined in config.h:
    ; CFG_sx1276_radio 1
    ; CFG_eu868 1
    ; LMIC_DEBUG_LEVEL 0

    ; Ping and beacons not supported for class A, disable to save memory.
    -D DISABLE_PING
    -D DISABLE_BEACONS
```

Region, radio and debug settings CANNOT be changed in `platformio.ini`.
When making changes to `config.h` these must be changed separately per board!
File `config.h` is located in these folders: `.pio/libdeps/<board-id>/IBM LMIC framework/src/lmic`

Above settings cannot be changed in platformio.ini because they are defined in config.h in the library source code. The settings must be changed separately per board because PlatformIO downloads libraries separately for each board.

Do NOT use ABP activation when using the IBM LMIC framework library.
On The Things Network V3 this will cause a downlink message for EVERY uplink message because it does NOT properly handle MAC commands.

4.4 Board specific settings

All options listed below are specified for each board separately. This is done for flexibility and to provide the optimum configuration for each board out of the box.

Example board section for BSFrance LoRa32u4 II board:

```
[env:lora32u4II]
; BSFrance LoRa32u4 II V1.0, V1.1, V1.2, V1.3 (ATmega32u4).
; No display.
; Board versions V1.0 to V1.2 require manual wiring of DIO1 to GPIO5.
; 8-bit AVR MCU with limited memory, therefore Classic LMIC is used.
; See limitations described in the [classic_lmic] section.
platform = atmelavr
board = lora32u4II
framework = arduino
monitor_speed = ${common.monitor_speed}
lib_deps =
    ${common.lib_deps}
    ${classic_lmic.lib_deps}     ; [Deprecated] IBM LMIC Framework
build_flags =
    ${common.build_flags}
    ${classic_lmic.build_flags}  ; [Deprecated] IBM LMIC Framework
    -D BSFILE=\"boards/bsf_lora32u4II.h\"
    -D MONITOR_SPEED=${common.monitor_speed}
    -D USE_SERIAL
    -D USE_LED
    ; -D USE_DISPLAY             ; Requires external I2C OLED display
```

Example board section for Blue Pill board with external LoRa module:

```
[env:bluepill_f103c8]
; Bluepill F103C8 (64k) (STMF103C8T6).
; No display.
; Select preferred upload protocol below.
platform = ststm32
board = bluepill_f103c8
framework = arduino
; upload_protocol = serial
upload_protocol = stlink
; upload_protocol = jlink
monitor_speed = ${common.monitor_speed}
lib_deps =
    ${common.lib_deps}
    ${mcci_lmic.lib_deps}
build_flags =
    ${common.build_flags}
    ${mcci_lmic.build_flags}
    -D BSFILE=\"boards/bsf_bluepill_f103c8.h\"
    -D MONITOR_SPEED=${common.monitor_speed}
    -D _GNU_SOURCE
    -D LMIC_PRINTF_TO=Serial
    -D USE_SERIAL
    -D USE_LED
    ; -D USE_DISPLAY             ; Requires external I2C OLED display
```

By default all USE_ options for a board are enabled. If the option is not enabled by default then it is either not supported (e.g. due to hardware conflict) or in case of USE_DISPLAY requires an external display to be connected. In latter case the USE_DISPLAY option can be enabled after an external display is connected.

Each of these options uses memory. In case of memory constrainted 8-bit boards disabling some of these option may free some memory for other use.

USE_SERIAL
If enabled status output will be send to the serial port to be viewed on serial monitor.

USE_LED
If enabled it will use the onboard LED. The LED will be on during Tx/Rx transmission and off otherwise. For some boards the onboard LED cannot be used because of a hardware conflict, because the LED type is not supported (e.g. WS2812 RGB LED) or because there is no onboard user LED. If the onboard LED cannot be used then this option is disabled by default and the line will be commented to make this clear.

USE_DISPLAY
If enabled status output will be send to the display.

upload_protocol
For some boards it may be necessary to change the upload protocol, e.g. for bluepill change it from `stlink` to `serial` if a USB to serial adapter is used for upload instead of a STLink programmer.

LMIC library

By default all boards with 32-bit MCU are configured to use the MCCI LoRaWAN LMIC library (MCCI LMIC) because this is the library that should be used. It is the most LoRaWAN compliant LMIC library for Arduino and it is actively maintained.

By default all boards with 8-bit MCU are configured to use the IBM LMIC framework library (Classic LMIC). These boards have limited memory capacity and Classic LMIC uses less memory than MCCI LMIC. Because of better LoRaWAN compliance and improved functionality MCCI LMIC is preferred but there is not enough memory to use LMIC-node with MCCI LMIC on these boards.

The other entries in the board sections should stay unchanged.

4.5 Board Support Files

Each BSF contains the following settings. Except for DEVICEID_DEFAULT these settings are not enabled for each board. Values shown are examples.

```
#define DEVICEID_DEFAULT "pro-mini"  // Default deviceid value

// Wait for Serial
// Can be useful for boards with MCU with integrated USB support.
#define WAITFOR_SERIAL_SECONDS_DEFAULT 10   // -1 waits indefinitely

// LMIC Clock Error
// This is only needed for slower 8-bit MCUs (e.g. 8MHz ATmega328 and ATmega32u4).
// Value is defined in parts per million (of MAX_CLOCK_ERROR).
// Value 30000 was determined empirically.
#define LMIC_CLOCK_ERROR_PPM 30000
```

DEVICEID_DEFAULT

If no `DEVICEID` is defined and no `ABP_DEVICEID` is defined when ABP activation is used then `DEVICEID_DEFAULT` is used.
There is no need to change this value in the BSF. `DEVICEID` and `ABP_DEVICEID` can be defined in file `lorawan-keys.h` where they are kept nicely together with the LoRaWAN keys of the same device.

WAITFOR_SERIAL_SECONDS_DEFAULT

Defines the default value used for the 'wait for serial' timeout. This value is only defined in the BSF for boards where USB support is integrated into the MCU. It has no use for other boards.
If there is need to change this value then don't change the default value in the BSF and change `WAITFOR_SERIAL_SECONDS` in the `[common]` section in `platformio.ini` instead.

LMIC_CLOCK_ERROR_PPM

Setting this value will cause LMIC timing to be less strict. This is only needed for some boards (usually 8-bit AVR based boards). This value should normally not be changed especially for 32-bit MCU boards.
If there are any issues with timing then increasing this value could possibly help. All boards supported by

LMIC-node have already been tested and for boards that need it LMIC_CLOCK_ERROR_PPM is already enabled with a value that has proven to be sufficient.

For more information about Clock Error consult the MCCI LoRaWAN LMIC library documentation.

5 Instructions

For prerequisites see 1.3 Requirements

5.1 Select your board

In `platformio.ini`:

Step 1a: Select exactly one supported board by uncommenting the line with its board-id and name.

Step 1b: Comment the guard line to disable it. This is the line that starts with `<platformio.ini board selector guard>`.

5.2 Select your LoRaWAN region

For MCCI LMIC (default for 32-bit boards):
Step 2: In `platformio.ini` select you LoRaWAN region.
or
For Classic LMIC (default for 8-bit boards):
Step 2: In `config.h` select your LoRaWAN region (if not CFG_eu868). See IBM LMIC framework settings for details.

If you are not sure which of the above to select then check the section for your board in `platformio.ini`.

5.3 Provide the LoRaWAN keys for your node

Step 3a: If this is the first time: In the `keyfiles` folder copy file `lorawan-keys_example.h` to `lorawan-keys.h`.

Step 3b: In file `lorawan-keys.h` add the LoRaWAN keys for the device that were copied from the TTN Console.

Step 3c: Optional: if preferred, set values for `DEVICEID` and `ABP_DEVICEID` in `lorawan-keys.h` (otherwise default value will be used).

5.4 Compile and upload

Step 4a: Compile the code.

Step 4b: Start the serial monitor (if USE_SERIAL is enabled).

Step 4c: Upload the code (and manually reset the device if needed).

Your node should now be up and running.

The node will be running but to be able to see the counter value in uplink messages displayed on the console, an uplink decoder function needs to be installed (see next step).

5.5 Add uplink decoder function in TTN Console

Step 5: In the TTN Console, add the `decodeUplink()` JavaScript function as uplink payload formatter to the device. As formatter type select `JavaScript` and the function must be pasted in the field `Formatter parameter`.

The `decodeUplink()` function can be found in folder `payload-formatters` in file `lmic-node-uplink-formatters.js`.

When this function is installed, the counter value will be shown in uplink messages in 'Live data' on the TTN Console.

6 Additional information

6.1 Pinout diagrams

Pinout diagrams for most supported boards can be found here: https://github.com/lnlp/pinout-diagrams

6.2 The Things Network Forum

The following topics on The Things Network Forum contain useful additional information:

- Big ESP32 SX127x topic part 3
- TTGO T-Beam
- Big STM32 boards topic
- Big LoRa32u4 boards topic

6.3 Not yet tested

All supported boards have been tested except below boards for which hardware was not available.

- Heltec Wireless Stick and Wireless Stick Lite.
- BSFrance LoRa32u4 II v1.1 and v1.3 *(v1.0 and v1.2 were tested)*.
- TTGO LoRa32 v1.3.
- TTGO LoRa32 v2.1.6.
- Lolin D32 *(Lolin D32 Pro was tested)*.

6.4 Known issues

There are no known issues in this release.

7 Tips

7.1 Serial Monitor

If a development board has a MCU with built-in USB support (e.g. ATmega32u4 and SAMD21) and the serial monitor is connected, resetting the board (e.g. with onboard reset button) will temporarily disconnect the serial port and crash the serial monitor. The serial monitor will then have to be manually restarted and the output will be lost. For boards that use a USB to serial adapter (either onboard or external) this problem does not exist.

To prevent issues with the serial monitor it is best to start the serial monitor *before* uploading a new sketch. After uploading, PlatformIO will automatically switch back from the upload window to the serial monitor and the serial monitor will contine without crashing. If you connect the serial monitor after uploading you will most probably lose the first output, in which case you may be tempted to press the reset button to restart the output, but in case of MCU with built-in USB support this will only crash the serial monitor.

For boards with MCU with built-in USB support LMIC-node by default waits with a 10 seconds timeout for the serial port to become ready so the first output does not get lost. If not yet connected the serial monitor needs to be connected (or restarted) within these 10 seconds. The countdown stops when the serial port becomes ready or when the countdown ends (whichever is first). The value of the countdown period can be changed as needed by enabling and changing the value of (`WAITFOR_SERIAL_SECONDS` in `platformio.ini`). A value 0 will disable the 'wait for serial' countdown, a value of -1 will wait indefinitely and not contiunue until the serial port is ready. The serial port is detected as ready only when a serial monitor is connected.

7.2 Antenna

- **The antenna should always be connected when the device is powered!**
  Powering the board with antenna disconnected may damage the LoRa radio.
- For best results the antenna should be used in upright (vertical) position.

7.3 Distance to gateway

- The distance between the node and a gateway should be 3 meters at minimum. If the distance is less than 3 meters this can cause RF communication issues.

8 Example output

Below are some examples of status output from serial monitor and display.

8.1 Serial Monitor

Example output from Raspberry Pi Pico with RFM95 SPI LoRa module.
This is the first serial output after reset. It shows a join, the first two uplinks, then an uplink followed by a downlink containing a counter reset command, and then the next uplink where it is visible that the counter value has been reset. After the third uplink it shows that two downlinks have been received. One was for the reset counter command, the other downlink was for a MAC command that cannot be output because the sequence of up and downlinks all occur before the EV_TXCOMPLETE event is generated.

```
LMIC-node

Device-id:     rpi-pico
LMIC library:  MCCI
Activation:    OTAA
Interval:      60 seconds

000000087328:  Event: EV_JOINING

000000088779:  doWork job started
000000115173:  Event: EV_TXSTART
000000437671:  Event: EV_JOINED
               Network Id: 19
               Device Address: 260B9BA3
               Application Session Key: BD-40-9E-1F-9C-36-BC-1C-0F-ED-A9-4C-90-27-84-8A
               Network Session Key:     80-FC-6E-99-9C-67-63-3F-4D-61-70-E1-2C-1D-D6-8E

000003838779:  doWork job started
000003841831:  Input data collected
               COUNTER value: 1
000003842567:  Packet queued
000003844154:  Event: EV_TXSTART
000004225330:  Event: EV_TXCOMPLETE
               Up: 1,  Down: 0

000007588779:  doWork job started
000007591830:  Input data collected
               COUNTER value: 2
000007592585:  Packet queued
000007594181:  Event: EV_TXSTART
000007975350:  Event: EV_TXCOMPLETE
               Up: 2,  Down: 0

000011338779:  doWork job started
000011341830:  Input data collected
               COUNTER value: 3
000011342539:  Packet queued
000011344159:  Event: EV_TXSTART
000011725339:  Event: EV_TXCOMPLETE
               Up: 3,  Down: 0

000015088780:  doWork job started
000015091771:  Input data collected
               COUNTER value: 4
000015092467:  Packet queued
000015094084:  Event: EV_TXSTART
000015413725:  Event: EV_TXCOMPLETE
               Up: 4,  Down: 2
               Downlink received
               RSSI: -68 dBm,  SNR: 7.2 dB
               Port: 100
               Length: 1
               Data: C0
```

```
                Reset cmd received
000015417949:  Counter reset

000018838781:  doWork job started
000018841831:  Input data collected
                COUNTER value: 1
000018842563:  Packet queued
000018844177:  Event: EV_TXSTART
000019225356:  Event: EV_TXCOMPLETE
                Up: 5,  Down: 2
```

8.2 Display

To be added.

9 Release History

Release v1.3.0

- Added
  - Separate LMIC_PRINTF_TO definitions for each board that uses MCCI LMIC.
  - Subband selection for US915 and AU915 for OTAA (fixes issue for AU915).
  - Parameterization for `initLmic()` and `setAbpParameters()`.
  - Custom event handler for MCCI LMIC to capture `EV_RXSTART`.
  - User code markers in `setup()`.
  - Description for `setup()`, `processWork()` and `processDownlink()` functions and `doWork` job in `README.md`.
- Changed
  - Replace 'Arduino Zero (USB)' (`zerousb`) board with 'SAMD21 M0-Mini' (`samd21_m0_mini`).
  - For Adafruit Feather M0 LoRa board, change `lmic_pins.rssi_cal` from 10 to 8.
  - For BSFrance LoRa32u4 II board, change `lmic_pins.rssi_cal` from 10 to 8.
  - Set Tx indicators off at `EV_JOINED`, `EV_JOIN_TXCOMPLETE` and `EV_TXCANCELLED`.
  - Change Raspberry Pi Platform (GitHub URL) to official release (name, no URL).
  - Remove `#include <Arduino.h>` from LMIC-node.cpp.
- Fixed
  - Do not schedule uplinks while still joining.
    `processWork()` will be skipped when still joining (counter will not be read).
  - Incorrect DIO1 pin mapping (5 -> 6) for Adafruit Feather M0 LoRa board.
  - Incorrect #if statement for `LMIC_setDrTxpow()` in `initLmic()`.
  - AU915 joining fails because proper subband is not selected.
  - LMIC_PRINTF_TO not working (properly) (at least on Windows) for boards with ARM based MCU.
  - LMIC_PRINTF_TO not working for boards using SerialUSB.

- Known issues
  - None.

Release v1.2.0

- Added
  - Support for Raspberry Pi Pico board.
  - Support for Teensy LC board.
  - New pinout diagrams repository.
  - Link to pinout diagrams in `README.md`.
  - Links to forum topics in `README.md`.
  - Example output for serial monitor in `README.md`.
  - Improvements in `README.md`.
  - Specify more version numbers for supported boards in `README.md` and platformio.ini.
  - Compile error when `USE_LED` is defined for TTGO T-Beam v1.x boards.
- Changed
  - Prefix BSF file names with `bsf_`.
  - Rename board-id `ttgo_tbeam` to `ttgo_t_beam`.
  - Rename board-id `ttgo_tbeam_v1` to `ttgo_t_beam_v1`.
  - Rename header guards in BSF files that slipped a previous rename action.
  - Order of some chapters in `README.md`.
- Fixed
  - Casing in `#include "LMIC-node.h"` statement in .cpp and BSF files.
- Known issues
  - LMIC debug output currently does not work with Arduino Zero (USB) and Raspberry Pi Pico boards.

Release v1.1.0 - Maintenance release

- Update `README.md` and add Release History.
- Add support for SX1276 Low port and SX1272 in RSSI calculation.
- Add LMIC library and LMIC debug level (if >0) to print output.
- Add option to make possible to override in `platformio.ini`, the value of `STM32_POST_INITSERIAL_DELAY_MS` defined in BSF of STM32 boards.
- Add compile error when ABP is used with Classic LMIC because not compliant, generates downlink message for every uplink message because it does not properly handle MAC commands.
- For STM32 BSF's surround postInitSerial delay with `#ifdef USE_SERIAL`.
- In Adafruit Feather M0 LoRa BSF change DIO1 from pin 5 to pin 6 because that is what Adafruit instructs in their tutorial.
- Fix `ABP_DEVICEID` not being used if defined in `lorawan-keys.h`.

- Fix offset for RSSI calculation when using MCCI LMIC.
- Fix printEvent() error when no `USE_DISPLAY` and no `USE_SERIAL` are defined.
- Known issue: The MCCI LoRaWAN LMIC library has a problem with SerialUSB on the Arduino Zero (USB) board so LMIC debug output cannot be used.

Release v1.0.0 - Initial release

- First release of LMIC-node.
