How to create a LoRaWAN® LBM application with STM32CubeWL

Introduction

This application note guides the user through all the steps required to build and setup a LoRaWAN® Network with End Devices and the Relay feature based on the STM32WL series microcontroller with the new LoRaWAN® LBM stack (*since STM32CubeWL v1.5.0*).

LoRa® is a type of wireless telecommunication network designed to allow long-range communications at a very-low bitrate and to enable long-life battery-operated sensors. LoRaWAN® defines the communication and security protocol that ensures the interoperability with the LoRa® network.

The firmware in the STM32CubeWL LBM Package is compliant with the LoRa Alliance® specification protocol named LoRaWAN® and the LoRaWAN® Relay feature. It has the following main features:

- Application integration ready
- Easy add-on of the low-power LoRa® solution
- Extremely low CPU load
- No latency requirements
- Small STM32 memory footprint
- Low-power timing services

The firmware of the STM32CubeWL MCU Package is based on the STM32Cube HAL drivers.

This document provides customer application examples on the NUCLEO-WL55JC development board with STM32WL55JC (order codes NUCLEO-WL55JC1 for high-frequency band and NUCLEO-WL55JC2 for low-frequency band) and B-WL5M-SUB1 connectivity expansion board with STM32WL5M.

To fully benefit from the information in this application note and to create an application, the user must be familiar with the STM32 series microcontrollers, the LoRa® technology, and understand system services such as low-power management and task sequencing.

1 General information

The STM32CubeWL package runs runs on STM32WL5x/Ex series microcontrollers based on the Arm® Cortex®-M processor.

Note: Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere.

The Arm word and logo are trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved.

arm

Table 1. Acronyms and terms

Acronym	Definition
ADR	Adaptive Data Rate
HAL	Hardware abstraction layer
LoRa®	Long range radio technology
LoRaWAN®	LoRa® wide-area network
LBM	LoRaWAN® Basic Modem
<Device>	This must be replaced with the of device: End_Device, End_Device_Relay and Relay
Migration note	This indicates notes for user that already used a STM32CubeWL FW package version up to v1.4.x

Reference documents

[1] TS001-1.0.4 LoRaWAN® Link Layer 1.0.4 Specification - 2020, October

[2] RP002-1.0.3 LoRaWAN® Regional Parameters Specification - 2021, May

[3] TS001-1.0.1 LoRaWAN® Relay Mechanism Specification - 2022, February

[4] RP002-1.0.4 LoRaWAN® Regional Parameters Specification - 2022, September

[5] User manual *Description of STM32WL HAL and low-layer drivers* (UM2642)

[6] Semtech GitHub repository on https://github.com/Lora-net/SWL2001/

LoRa® standard

Refer to documents [1], [2], [3] and [4] for more details on LoRaWAN® and Relay recommendations.

STM32CubeWL firmware packages

The table below lists the applications using the NUCLEO-WL55JC1, NUCLEO-WL55JC2, and/or B-WL5M-SUB1 boards.

Table 2. LoRaWAN® projects list

Module name	Project module	NUCLEO-WL55JC1(1)	NUCLEO-WL55JC2(2)	B-WL5M-SUB1
LoRaWAN	LoRaWAN_End_Node_LBM	Yes	No	Yes
	LoRaWAN_End_Node_Relay_LBM	Yes	No	Yes
	LoRaWAN_Relay_LBM	Yes	No	Yes

1. *High-frequency band*
2. *Low-frequency band*

2 STM32CubeWL overview

The firmware of the STM32CubeWL MCU Package includes the following resources (see Figure 1):

- Board support package:
  - STM32WL_Nucleo drivers
  - B-WL5M-SUBG1 drivers
- STM32WLxx_HAL_Driver
- Middleware:
  - LoRaWAN® containing:
    - LoRaWAN® stack APIs
    - LoRaWAN® core implementation
    - LoRa® basic modem additional porting functions
  - SubGHz_Phy containing:
    - PHY layer for LoRaWAN® LBM stack
    - SubGHz_Phy layer containing the radio and radio_driver interfaces (not LoRaWAN®)
- LoRaWAN applications:
  - LoRaWAN_End_Node_LBM
  - LoRaWAN_End_Node_Relay_LBM
  - LoRaWAN_Relay_LBM
- SubGHz_Phy application:
  - SubGHz_Phy_PingPong (SingleCore and DualCore)
  - SubGHz_Phy_Per (SingleCore)
  - SubGHz_Phy_AT_Slave (SingleCore)
  - SubGHz_Phy_LrFhss (SingleCore)

Figure 1. Project file structure

DT80950v1

3 SubGHz HAL driver

This section focuses on the SubGHz HAL (other HAL functions such as timers or GPIO are not detailed).

The SubGHz HAL is directly on top of the sub-GHz radio peripheral.

The SubGHz HAL driver is based on a simple one-shot command-oriented architecture (no complete processes). Therefore, no LL driver is defined.

This SubGHz HAL driver is composed of the following main parts:

- Handle, initialization and configuration data structures
- Initialization APIs
- Configuration and control APIs
- MSP and event callbacks
- Bus I/O operation based on the SUBGHZ_SPI (intrinsic services)

As the HAL APIs are mainly based on the bus services to send commands in one-shot operations, no functional state machine is used except the RESET/READY HAL states.

3.1 SubGHz resources

The following HAL SubGHz APIs are called at the initialization of the radio:

- Declare a SUBGHZ_HandleTypeDef handle structure.
- Initialize the sub-GHz radio peripheral by calling the `HAL_SUBGHZ_Init(&hUserSubghz)` API.
- Initialize the SubGHz low-level resources by implementing the `HAL_SUBGHZ_MspInit()` API:
  - PWR configuration:
    - Enable wake-up signal of the sub-GHz radio peripheral.
  - NVIC configuration:
    - Enable the NVIC radio IRQ interrupts.
    - Configure the sub-GHz radio interrupt priority.

The following HAL radio interrupt is called in the `stm32wlxx_it.c` file:

- `HAL_SUBGHZ_IRQHandler` in the SUBGHZ_Radio_IRQHandler.

3.2 SubGHz data transfers

The **Set** command operation is performed in polling mode with the `HAL_SUBGHZ_ExecSetCmd();` API.

The **Get status** operation is performed using polling mode with the `HAL_SUBGHZ_ExecGetCmd();` API.

The read/write register accesses are performed in polling mode with the following APIs:

- `HAL_SUBGHZ_WriteRegister();`
- `HAL_SUBGHZ_ReadRegister();`
- `HAL_SUBGHZ_WriteRegisters();`
- `HAL_SUBGHZ_ReadRegisters();`
- `HAL_SUBGHZ_WriteBuffer();`
- `HAL_SUBGHZ_ReadBuffer();`

4 BSP STM32WL Nucleo boards

This BSP driver provides a set of functions to manage radio RF services, such as RF switch settings and control, TCXO settings, and DC-DC settings.

Note: The radio Middleware (SubGHz_Phy) interfaces the radio BSP via `radio_board_if.c/h` interface file. When a custom user board is used, it is recommended to perform one of the following:

- *First option*
  - *Copy the `BSP/STM32WLxx_Nucleo/` directory.*
  - *Rename and update the user BSP APIs with:*
    - *User RF switch configuration and control (such as pin control or number of ports)*
    - *User TCXO configuration*
    - *User DC-DC configuration*
  - *Replace in the IDE project the STM32WLxx_Nucleo BSP files by the user BSP files.*
- *Second option*
  - *Disable `USE_BSP_DRIVER` in `Core/Inc/platform.h` and implement the BSP functions directly into `radio_board_if.c`.*

4.1 Frequency band

Two types of Nucleo board are available on the STM32WL series:

- NUCLEO-WL55JC1: high-frequency-band, tuned for frequency between 865 MHz and 930 MHz
- NUCLEO-WL55JC2: low-frequency-band, tuned for frequency between 470 MHz and 520 MHz

If the user tries to run a firmware compiled at 868 MHz on a low-frequency-band board, very poor RF performances are expected.

The firmware does not check the band of the board on which it runs.

4.2 RF switch

The STM32WL Nucleo board embeds an RF 3-port switch (SP3T) to address, with the same board, the following modes:

- High-power transmission
- Low-power transmission
- Reception

Table 3. BSP radio switch

Function	Description
`int32_t BSP_RADIO_Init(void)`	Initializes the RF switch.
`int32_t BSP_RADIO_ConfigRFSwitch(BSP_RADIO_Switch_TypeDef Config)`	Configures the RF switch.
`int32_t BSP_RADIO_DeInit (void)`	De-initializes the RF switch.
`int32_t BSP_RADIO_GetTxConfig(void)`	Returns the board configuration: high power, low power, or both.

The RF states versus the switch configuration are given in the table below.

Table 4. RF states versus switch configuration

RF state	FE_CTRL1	FE_CTRL2	FE_CTRL3
High-power transmission	Low	High	High
Low-power transmission	High	High	High
Reception	High	Low	High

4.3 RF wake-up time

The sub-GHz radio wake-up time is recovered with the following API.

Table 5. BSP radio wake-up time

Function	Description
`uint32_t BSP_RADIO_GetWakeUpTime(void)`	Returns `RF_WAKEUP_TIME` value.

The user must start the TCXO by setting the command `RADIO_SET_TCXOMODE` with a timeout depending of the application.

The timeout value can be updated in `radio_conf.h`. The default template value is the following:

```
#define RF_WAKEUP_TIME                    1U
```

4.4 TCXO

Various oscillator types can be mounted on the user application. On the STM32WL Nucleo boards, a temperature compensated crystal oscillator (TCXO) is used to achieve a better frequency accuracy.

Table 6. BSP radio TCXO

Function	Description
`uint32_t BSP_RADIO_IsTCXO (void)`	Returns `IS_TCXO_SUPPORTED` value.

The TCXO mode is defined by the STM32WL Nucleo BSP by selecting `USE_BSP_DRIVER` in `Core/Inc/platform.h`.

If the user wants to update this value (no Nucleo board compliant), or if the BSP is not present, the TXCO mode can be updated in `radio_board_if.h`. The default template value is the following:

```
#define IS_TCXO_SUPPORTED                 1U
```

4.5 Power regulation

Depending on the user application, a LDO or an SMPS (also named DC-DC) is used for power regulation. An SMPS is used on the STM32WL Nucleo boards.

Table 7. BSP radio SMPS

Function	Description
`uint32_t BSP_RADIO_IsDCDC (void)`	Returns `IS_DCDC_SUPPORTED` value.

The DC-DC mode is defined by the STM32WL Nucleo BSP by selecting `USE_BSP_DRIVER` in `Core/Inc/platform.h`.

If the user wants to update this value (no Nucleo board compliant), or if the BSP is not present, the DC-DC mode can be updated in `radio_board_if.h`. The default template value is defined below:

```
#define IS_DCDC_SUPPORTED                    1U
```

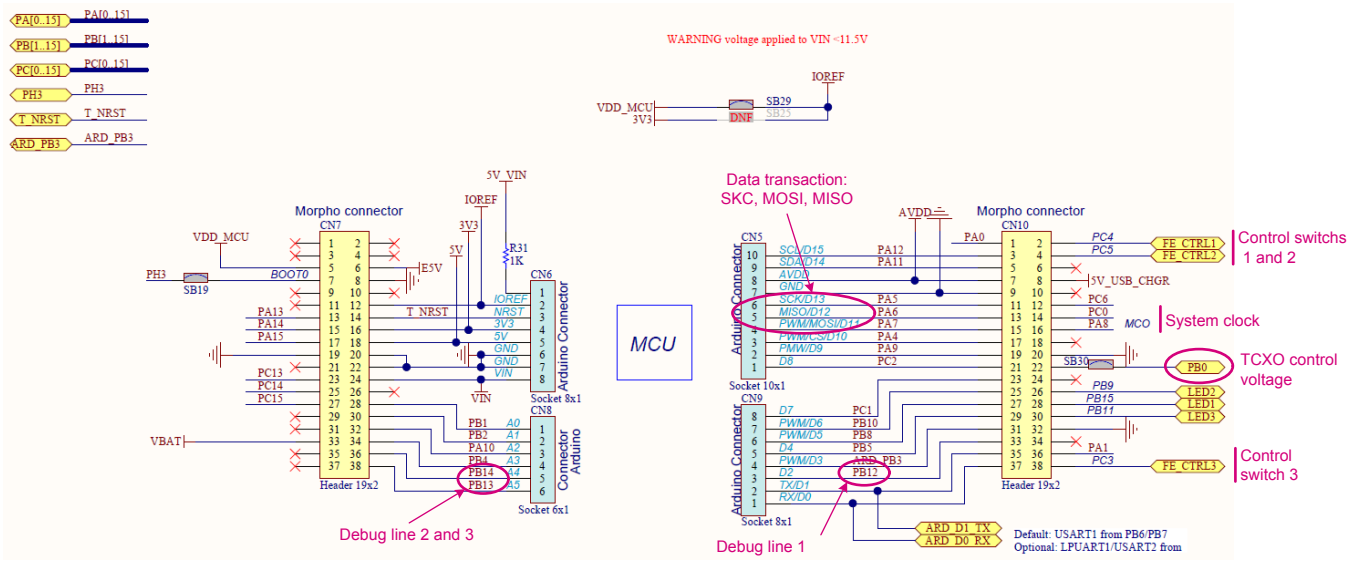
The SMPS on the board can be disabled by setting `IS_DCDC_SUPPORTED = 0`.

4.6 STM32WL Nucleo board schematic

The figure below details the STM32WL Nucleo board (MB1389 reference board) schematic, highlighting some useful signals:

- Control switches on PC4, PC5 and PC3
- TCXO control voltage pin on PB0
- Debug lines on PB12, PB13 and PB14
- System clock on PA8
- SCK on PA5
- MISO on PA6
- MOSI on PA7

Figure 2. NUCLEO-WL55JC schematic



5 BSP B-WL5M-SUBG1 boards

The BSP driver provides a set of functions to manage radio RF switch settings. It also gives additional components available on B-WL5M-SUBG1 such as external flash memory, LEDs, and sensors (environmental, motion, etc.).

5.1 RF switch

The B-WL5M-SUBG1 board embeds a RF switch integrated in the STM32WL5MOC. There is only a read API to return the board configuration.

Table 8. BSP radio switch

Function	Description
`int32_t BSP_RADIO_GetTxConfig(void)`	Returns the board configuration: high power, low power, or both.

5.2 External components

The B-WL5M-SUBG1 board embeds multiples components:

- One User PushButton
- 3 LEDs
- Temperature/barometer sensors
- Accelerometer, gyroscope and magnetometer sensors
- External flash memory

For more details about the defined APIs, refer to the User Manual document in the BSP directory.

6 LoRaWAN® stack description

The firmware of the STM32CubeWL MCU package includes STM32WL resources such as:

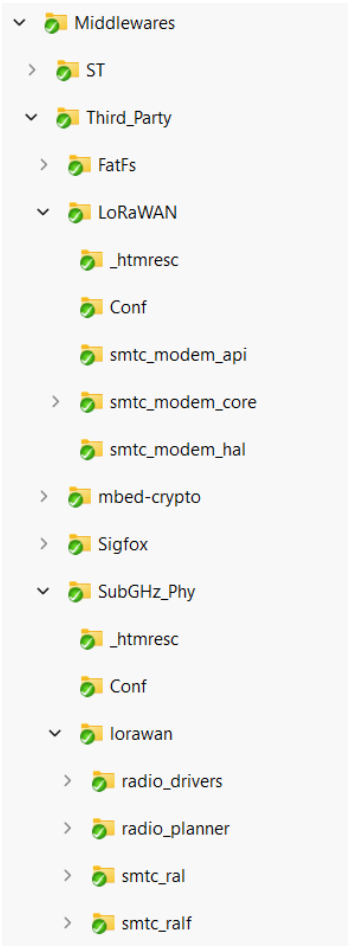
- STM32WLxx_Nucleo drivers
- B-WL5M-SUBG1 drivers
- STM32WLxx HAL drivers
- LoRaWAN® Middleware
- SubGHz physical layer Middleware
- LoRaWAN application example
- Utilities

The LoRaWAN® stack available in this STM32CubeWL MCU package is based on the LoRa® Basics Modem SW architecture provided by Semtech in [6].

Migration note: LoRaWAN® stack has a different structure and source code. Take care on API, structure and configuration sections.

It is composed by LoRaWAN® Middleware and SubGHz Middleware (for LoRaWAN® protocol) and its structure is shown in the figure below.

Figure 3. LoRaWAN®stack architecture



DT80971v1

The LoRaWAN® stack Middleware for STM32 series microcontrollers is split into several modules:

Table 9. LoRaWAN® stack description

Module	Description	Location
LoRaMAC layer	Implements the LoRaMAC stack specification.	`Middlewares\Third_Party\LoRaWAN\smtc_modem_core\lr1mac`
LoRaWAN® stack API interface	Implementes the LoRaWAN® API interface for LoRaWAN® stack Core and Relay features.	`Middlewares\Third_Party\LoRaWAN\smtc_modem_api`
Region layer	Implements the regional parameters specification.	`Middlewares\Third_Party\LoRaWAN\smtc_modem_core\lr1mac\src\smtc_real`
Relay feature	Implements Relay feature both for Relay device and End-Node with Relay feature.	`Middlewares\Third_Party\LoRaWAN\smtc_modem_core\lr1mac\src\relay`
Certification support	Implements certification tests support.	`Middlewares\Third_Party\LoRaWAN\smtc_modem_core\lorawan_packages\lorawan_certification`
LoRa® crypto	Implements AES/CMAC algorithms and interface with SecureEngine element.	`Middlewares\Third_Party\LoRaWAN\smtc_modem_core\smtc_modem_crypto`
LoRa® utilities	Implements the common utility functions.	`Middlewares\Third_Party\LoRaWAN\smtc_modem_core\modem_utilities`

Several LoRaWAN® features are implemented in compliance with the LoRa® alliance protocol specifications:

- From the link layer specification:
  - On-board LoRaWAN® Class A, Class B and Class C protocol stack
  - Device activation through OTAA for End-Device, Relay and End-Device Relay (both to the GW and through the Relay device)
  - Adaptive data-rate support
- From the regional parameters specification:
  - EU 868 MHz ISM band ETSI compliant
  - US 915 MHz ISM band FCC compliant
  - KR 920 MHz ISM band defined by Korean government
  - RU 864 MHz ISM band defined by Russian regulation
  - CN470 MHz ISM band defined by Chinese government
  - CN470 MHz ISM band defined by Chinese government (LoRaWAN® regional parameters v1.0)
  - AS 923 MHz ISM band defined by Asian governments
  - AU 915 MHz ISM band defined by Australian government
  - IN 865 MHz ISM band defined by Indian government

Migration note: EU433 and CN779 are no more supported.

Additionally, the LoRaWAN® stack integrates:

- Certification solution in accordance with the specifications described as below
- NVM context management to prevent power off loss of context
- Low-power integration by standby/sleep radio states

6.1 LoRaWAN® specifications version

The Link Layer specification, the Regional Parameters specifications and the Relay specification are defined by the LoRa® Alliance.

The LoRaWAN® stack implements:

- LoRaWAN Link Layer 1.0.4 Specification (TS001-1.0.4)
- LoRaWAN 2-1.0.3 Regional Parameters Specification (RP002-1.0.3) + LoRaWAN 2-1.0.4 Regional Parameters Specification (RP002-1.0.4) for the Relay feature
- LoRaWAN Relay 1.0.1 Specification (TS011-1.0.1)

6.2 LoRaWAN® commissioning

Depending of the LoRaWAN® version used, each End-Device has to be personalized and activated with some unique identifier and some Network Keys shared with your preferred LoRaWAN® network.

Activation of an End-Device can be achieved in two ways via:

- Over-The-Air Activation (OTAA) when an End-Device is deployed or reset. This is valid for the three types of devices (End-Device, End-Device-Relay and Relay)
- Over-The-Air Activation (OTAA) when an End-Device-Relay is deployed or reset and will be connected to GW through Relay device.

It is required to personalize the End-Device with:

- A device unique identifier (DevEUI)
- An application/join identifier (JoinEUI)
- A network root key (NwkKey)
- An application root key (AppKey)

Migration note: ABP activation is no more supported.

6.3 LoRaWAN® pre-certification

The system including the NUCLEO-WL55JC board and the STM32CubeWL firmware modem application has been verified by LoRaWAN® LCTT tool with pre-certification tests and passed for Class A, Class C and Relay tests with EU868, IN865, AU915, AS923, and US915 bands.

6.4 Required STM32 peripherals to drive the radio

Sub-GHz radio

The sub-GHz radio peripheral is accessed through the `stm32wlxx_hal_subghz` HAL.

The sub-GHz radio issues an interrupt through SUBGHZ_Radio_IRQHandler NVIC, to notify a TxDone or RxDone event. More events are listed in the product reference manual.

RTC

The RTC (real-time clock) calendar is used as 32-bit counter running in all power modes from the 32 kHz external oscillator. By default, the RTC is programed to provide 1024 ticks (subseconds) per second. The RTC is programed once at hardware initialization (when the MCU starts for the first time). The RTC output is limited to a 32-bit timer that corresponds to about a 48-day period.

Caution: *When changing the tick duration, the user must keep it below 1 ms.*

RNG

The RNG (random number generator) is used as a 32-bit true random number generator, produced by an analog entropy source conditioned by a NIST SP800-90B approved conditioning stage.

By default, the RNG is disabled for power consumption concerns. Every time a random number must be generated, it is enabled/disabled by the *HAL_RNG_Init/HAL_RNG_Init.*

Caution: *Radio number generator MUST not be used, due to error on radio operations.*

7 LoRaWAN Middleware description

The LoRaWAN LBM Middleware is made by an internal scheduler that manages all activities necessary at all levels from the PHY up to the App.

Migration note: LoRaWAN® stack has a different structure and source code. Take care on API, structure and configuration sections.

7.1 LoRaWAN Middleware initialization

The initialization of the LoRaWAN® stack layer is done through the *smtc_modem_init* API, that initializes both the LoRaWAN® stack and the associated callbacks (see the table below).

This is the first operation to be done to receive the *SMTC_MODEM_EVENT_RESET* event.

All other configuration or operation MUST be done after the reception of the *SMTC_MODEM_EVENT_RESET* event.

Table 10. LoRaWAN® Middleware initialization

Function	Description
`void smtc_modem_init( Callbacks_t *handlerCallbacks)`	Initializes the LoRaWAN® stack (see Section 7.3: Application callbacks)

7.2 LoRaWAN Middleware MAC layer APIs

The LoRaWAN® LBM stack has an internal supervisor and scheduler that acts as manager of all activities of the stack.

The LoRaWAN® stack provides these following services:

- In general, the LoRaWAN® stack uses the *EventCallback* to notify MW events to the application (for example, join, TX/ RX data, alarm).

Table 11. Stack event callback API

Function	Description
`void EventCallback (void);`	Callback event raised internally by the stack. This the principle way to interact with the middleware based on the notification rand the state of the stack.

- Enable Relay feature for Relay device to manage ED Relay devices. This is disabled by default as specified from the LoRa® alliance.

Table 12. Enable Relay feature API for Relay device

Function	Description
smtc_modem_return_code_t smtc_modem_relay_rx_enable(void)	Enable Relay feature for Relay device. It uses a default configuration.

- Enable Relay feature an End-Device that can connect to the GW through a Relay device. This is enabled by default as specified from the LoRa® alliance. (It is disable in certification mode).

Table 13. Enable Relay feature API for End-Device with Relay feature

Function	Description
`smtc_modem_return_code_t smtc_modem_relay_tx_enable( uint8_t stack_id, const smtc_modem_relay_tx_config_t* relay_config );`	Enable Relay feature for End-Device to communicate through the GW.

- Request to join to the network

Table 14. Join network API

Function	Description
`smtc_modem_return_code_t smtc_modem_join_network( uint8_t stack_id )`	Request to join to the network.

- Request to send a data packet

Table 15. Send Tx data API

Function	Description
`smtc_modem_return_code_t smtc_modem_request_uplink( uint8_t stack_id, uint8_t f_port, bool confirmed, const uint8_t* payload, uint8_t payload_length )`	Request to send TX data.

- Run the LoRaWAN® stack FSM. This function must be called in loop. It returns an amount of ms after which the function must at least be called again. This time is also the Idle time for which the device can sleep. It is used for low power management in the applications.

Table 16. LoRaWAN® FSM engine

Function	Description
`uint32_t smtc_modem_run_engine(void)`	Run the FSM engine of the stack.

- Schedule an application operation.

Table 17. User App schedule event

Function	Description
`smtc_modem_return_code_t smtc_modem_alarm_start_timer( uint32_t alarm_s )`	Schedule user alarm event in seconds. An *SMTC_MODEM_EVENT_ALARM* event is received at application level when timer expires. *Note: This time is considered in Idle time calculation.*

7.3 Application callbacks

Callbacks in the tables below are used for all LoRaWAN applications.

Table 18. LoRaWAN® Middleware initialization

Function	Description
`void EventCallback (void);`	Callback event raised internally by the stack. This the principle way to interact with the Middleware based on the notification rand the state of the stack
`void RestoreContext (const modem_context_type_t ctx_type, uint32_t offset, uint8_t* buffer, const uint32_t size );`	Restore the NVM Data context from the flash
`void StoreContext( const modem_context_type_t ctx_type, uint32_t offset, const uint8_t* buffer, const uint32_t size );`	Store the NVM Data context to the flash
`uint32_t GetRandomValue(void);`	Get random value using the RNG module
`uint8_t GetBatteryLevel (void);`	Get the current battery level
`int16_t GetTemperatureLevel (void);`	Get the current temperature
`void SystemReset (void);`	Will be called to reset the system

8 LoRaWAN application

This application measures the battery level and the temperature of the MCU. These values are sent periodically to the LoRa® network using the LoRa® radio in Class A at 868 MHz.

Focus on the configuration described below to setup the application.

Migration note: LoRaWAN® stack has a different structure and source code. Take care on API, structure and configuration sections.

Sequencer is no more used, because the LBM stack supports an internal scheduler.Take the new lora_app.c file as reference source code.

8.1 LoRaWAN application description

There are three applications available under *\Projects\NUCLE-WL55JC\Applications\LoRaWAN:*

LoRaWAN_End_Device_LBM:

Acts as normal LoRaWAN End Device Class A device.

This application measures the battery level and the temperature of the MCU. These values are sent periodically to the LoRa® network using the LoRa® radio in class A at 868 MHz.

LoRaWAN_End_Device_Relay_LBM:

Acts as LoRaWAN End Device with the Relay feature enabled by default. The Relay feature parameters configuration could be changed in the *lora_app.c* function before to enable the functionalities. See the Table 20. LoRaWAN API for details.

An End Device with Relay feature is able to communicate with the Relay through WOR protocol and with the Network Server through the Relay as specified in [3] and [4].

This application measures also the battery level and the temperature of the MCU. These values are sent periodically to the LoRa® network using the LoRa® radio in Class A at 868 MHz though the Relay device if It is present and configured.

LoRaWAN_Relay_LBM:

Acts as LoRaWAN® Class A device with the possibility to enable the Relay feature though API (see Table 20. LoRaWAN API for details) or through Network Server commands. The Relay feature is disabled by default.

A Relay device with Relay feature communicates with the Gateway as Class A device, is able to communicate with an ED with Relay feature using the WOR protocol and forwards the ED's packet to the NS as specified in [3] and [4].

This application measures also the battery level and the temperature of the MCU. These values are sent periodically to the LoRa® network using the LoRa® radio in class A at 868 MHz.

8.2 LoRaWAN user code section description

Four main functions are defined as example to implement and use the LoRaWAN® stack in *\Projects\<target>\Applications\LoRaWAN\LoRaWAN_<device>_LBM\LoRaWAN\App\lora_app.c.*

These functions contain example code which can be overwritten to handle the specific features of the application layer.

Caution: *Do not change the order of the LoRaWAN APIs call.*

Migration note: LoRaWAN® stack has a different structure and source code. Take care on API, structure and configuration sections.

Sequencer is no more used, because the LBM stack supports an internal scheduler. Take the new lora_app.c file as reference source code.

Table 19. LoRaWAN user function

App function	Description
`void LoRaWAN_Init (void)`	Initialize the LoRaWAN® application
`void LoRaWAN_Process (void)`	LoRaWAN application continuous process

App function	Description
`static void EventCallback(void)`	Callback implementation for all LoRaWAN® Middleware events
`static void SendTxData(void)`	Example of LoRaWAN Tx process with a generic content payload generation and a smtc_modem_request_uplink (...) call with the temporary payload generated buffer

This following table describes the main LoRaWAN APIs in use in the applications.

For details on the LoRaWAN LBM API please refer to Semtech repository in [6].

Migration note: LoRaWAN® stack has a different structure and source code. Take care on API, structure and configuration sections.

The LBM stack supports an internal scheduler. Take the new lora_app.c file as reference source code.

Table 20. LoRaWAN API

LoRaWAN® API	Description	App function call
smtc_modem_init(&modem_event_callback);	Init the LoRaWAN® stack and use modem_event_callback as event callback, please note that the callback will be called immediately after the first call to smtc_modem_run_engine because of the reset detection.	LoRaWAN_Init
smtc_modem_set_certification_mode(STACK_ID, CertMode);	Certification mode is disabled by default. It can be enabled setting LORAWAN_CERTIFICATION_MODE to true	LoRaWAN_Init
smtc_modem_set_crystal_error_ppm(BSP_CRYSTAL_ERROR);	BSP crystal accuracy could be set to a different value. By default is 10.	LoRaWAN_Init
smtc_modem_run_engine();	LoRaWAN central process engine	LoRaWAN_Process
smtc_modem_get_event(&current_event, &event_pending_count)	Read the event generated by the stack.	modem_event_callback
smtc_modem_set_deveui(stack_id, user_dev_eui)	Configure the DevEUI	
smtc_modem_set_joineui(stack_id, user_join_eui)	Configure the JoinEUI	
smtc_modem_set_appkey(stack_id, user_gen_app_key)	Configure the Application Key (Gen APP key)	
smtc_modem_set_nwkkey(stack_id, user_app_key)	Configure the Network Key (APP Key)	
smtc_modem_is_certification_port_disabled(STACK_ID)	Verify if the certification port 224 was deactivated by LoRaWAN command. *Note: Command added by ST.*	
smtc_modem_set_region(stack_id, ACTIVE_REGION)	Configure the region.	
smtc_modem_join_network(stack_id)	Start the join to a LoRaWAN® network. This could be done once all parameters are set.	
smtc_modem_get_downlink_data(rx_payload, &rx_payload_size, &rx_metadata, &rx_remaining)	Upon receiving a downlink with data for application, this could be retrieved (data and metadata) calling this function.	
smtc_modem_relay_tx_enable(stack_id, &relay_config)	Enable the Relay feature on End-Device with Relay ability. It is enabled by default using the LoRaWAN_End_Device_Relay APP. The configuration could be changed before the function call.	

LoRaWAN® API	Description	App function call
	Note: It is available only in the LoRaWAN_End_Device_Relay APP.	modem_event_callback
smtc_modem_relay_rx_enable();	Enable the Relay feature on device that can acts as Relay. It is disabled by default and can be enabled uncommenting the function call in *lora_app.c* file. The Relay configuration is the default suggested by LoRa® Alliance. *Note: It is available only in the LoRaWAN_Relay APP.* *Command added by ST.*	
smtc_modem_alarm_start_timer/ smtc_modem_alarm_clear_timer	Enable / disable a user alarm that will be managed by the stack with the other Middleware activities.	modem_event_callback

8.3 LoRaWAN® device App configuration

The following table shows which parameters could be configured in each application.
The ACTIVE_REGION in use is the EU868.

Table 21. Device configuration

Define	Definition	Location
USER_LORAWAN_DEVICE_EUI	End-Device IEEE Extended Unique	lora_app.c
USER_LORAWAN_JOIN_EUI	Application or join server IEEE EUI	
USER_LORAWAN_GEN_APP_KEY	Application root key used to derive application session keys	
USER_LORAWAN_APP_KEY	Network root key used to derive session keys at Join in OTAA	
ACTIVE_REGION	Default supported region is EU868	lora_app.h
APP_TX_DUTYCYCLE	Duty cycle value for TX packets in seconds	
LORAWAN_USER_APP_PORT	Application port for Data Packet. Ports 224 and 226 are reserved values	
BSP_CRYSTAL_ERROR	Crystal error of the MCU to fine adjust the rx window for lorawan. (ex: set 30 for a crystal error = 0.3%). Default value is 10	
LORAWAN_CERTIFICATION_MODE	Allows to enable certification mode for LCTT certification tests. By default is set to false and is disabled. **Migration note:** LORAWAN_FORCE_REJOIN_AT_BOOT is no more supported.	
LORAWAN_APP_DATA_BUFFER_MAX_SIZE	Maximum data buffer size	

8.3.1 Activation method and keys

The OTAA is supported by default. ABP is no supported.
\Projects\<target>\Applications\LoRaWAN\LoRaWAN_<device>_LBM\LoRaWAN\App\se-identity.h file contains commissioning data useful for device activation.

8.3.2 LoRaWAN® class

By default, the device works as Class A device. The Class C could be enabled with the proper LoRaWAN® MAC command sent by the Network Server.

8.3.3 TX trigger

There are two ways to generate an uplink action, with the *EventType* global variable in *\Projects\<target>\Applications\LoRaWAN\LoRaWAN_<device>_LBM\LoRaWAN\App\lora_app.c*:

- By timer
- By an external event

with the code:

```
static TxEventType_t EventType = TX_ON_TIMER;
```

where *TxEventType_t enum* is defined as follows:

```
typedef enum TxEventType_e {
       TX_ON_TIMER = 0, /* App data transmission issue based on timer */
       TX_ON_EVENT = 1, /* App data transmission by external event */
}TxEventType_t;
```

The *TX_ON_EVENT* feature uses the button 1 as event in the LoRaWAN_<device>_LBM application.

8.3.4 Duty cycle

The duty cycle value (in s) to be used for the application is defined in `\Projects\<target>\Applications\LoRaWAN\LoRaWAN_<device>_LBM\LoRaWAN\App\lora_app.h` with the code below (for example):

```
#define APP_TX_DUTYCYCLE 10 /* 10s duty cycle */
```

This value by default is different for each application based on the common usage and data load in real communication with and through the GW.

The default values are:

- 10s for LoRaWAN_End_Node_LBM App
- 1800s for LoRaWAN_End_Node_Relay_LBM App
- 1200s for LoRaWAN_Relay_LBM App

8.3.5 Application port

The application port to be used for the application is defined in `\Projects\<target>\Applications\LoRaWAN\LoRaWAN_<device>_LBM\LoRaWAN\App\lora_app.h`, with the code below (for example):

```
#define LORAWAN_APP_PORT 2
```

Note: LORAWAN_APP_PORT must not use port 224 that is reserved for certification and port 226 that is reserved for Relay feature.

8.3.6 Data buffer size

The size of the buffer sent to the network is defined in `\Projects\<target>\Applications\LoRaWAN\LoRaWAN_<device>_LBM\LoRaWAN\App\lora_app.h`, with the code below:

```
#define LORAWAN_APP_DATA_BUFFER_MAX_SIZE 242
```

This value defines the maximum payload size that the device can be sent. But the uplink/downlink payload size is also dependent of the data rate used. For more details, refer to maximum payload size table of Regional Parameters specification.

8.3.7 Adaptive data rate (ADR)

The ADR is enabled by default with a Static ADR Mode, related to static devices with ADR managed by Network Server.

Different profiles might be configured using the *smtc_modem_adr_set_profile* API. Please refer to doxigen documentation for specific profiles.

8.3.8 LoRa® band selection

The region and its corresponding band selection are defined in `\Projects\<target>\Applications\LoRaWAN\LoRaWAN_<device>_LBM\LoRaWAN\Target\lorawan_conf.h` with the code below:

```
#define REGION_AS923
#define REGION_AU915
#define REGION_CN470
#define REGION_EU868
#define REGION_KR920
#define REGION_IN865
#define REGION_US915
#define REGION_RU864
#define REGION_CN779_RP_1_0
```

Note: Several regions can be enabled at compilation on the same application.

Depending on the region, the default active region must be defined in `\Projects\<target>\Applications\LoRaWAN\ LoRaWAN_<device>_LBM \Core\Inc\sys_conf.h` with the code (example for Europe):

```
#define ACTIVE_REGION LORAMAC_REGION_EU868
```

When the REGION_AS923 is enabled, the specific profile AS923 profile MUST be configured within the ACTIVE_REGION define choosing between the possible selections below:

- LORAMAC_REGION_AS923_GRP1 (Default configuration. Freq offset = 0.0 MHz / Freq. range = 915-928 MHz)
- LORAMAC_REGION_AS923_GRP1 (Freq offset = -1.80 MHz / Freq. range = 915-928 MHz)
- LORAMAC_REGION_AS923_GRP1 (Freq offset = -6.60 MHz / Freq. range = 915-928 MHz)
- LORAMAC_REGION_AS923_GRP1 (Freq offset = -5.90 MHz / Freq. range = 917-920 MHz)

Migration note: The REGION_AS923_DEFAULT_CHANNEL_PLAN define is no more used.

8.3.9 NVM context management

The NVM context management is used to store the current LoRaWAN® stack configuration in ROM to prevent loss of information due to a power-off or a reset of the board. The NVM context management is enabled by default and is implemented in

`\Projects\<target>\Applications\LoRaWAN\LoRaWAN_<device>_LBM\LoRaWAN\App\lora_app.c`

There are three types of information: *CONTEXT_LORAWAN_STACK*, *CONTEXT_MODEM* and *CONTEXT_SECURE_ELEMENT*.

The proposed solution is to store these information in a pre-allocated ROM page of 2 Kbytes. The details on sections and size are shown in the table below.

Table 22. NVM context sections

Context type	Description	Start address	Size (bytes)
CONTEXT_LORAWAN_STACK	Contains the *lr1_mac_nvm_context_t* struct information typically of LoRaWAN® session.	0x0803F000	0x28
CONTEXT_MODEM	Contains the *modem_ctx_t* struct information used by the stack.	0x0803F028	0x10
CONTEXT_SECURE_ELEMENT	Contains the *soft_se_context_nvm_t* struct information with the security material information for LoRaWAN® session.	0x0803F038	0x2A0

The context store is automatically done by the LoRaWAN® Middleware based on the LoRa® specification.

If callbacks listed in Table 23. NVM context callback are not defined in `\Projects\<target>\Applications\LoRaWAN\LoRaWAN_<device>_LBM\LoRaWAN\App\lora_app.c`, the feature is considered as disabled.

Table 23. NVM context callback

Context type	Description
static void RestoreContext(const modem_context_type_t ctx_type, uint32_t offset, uint8_t *buffer, const uint32_t size);	Restores the NVM data stored in FLASH to a backup buffer in RAM for the specific context ctx_type.

Context type	Description
static void StoreContext(const modem_context_type_t ctx_type, uint32_t offset, const uint8_t *buffer, const uint32_t size);	Stores the NVM data from the RAM to the FLASH for the specific context ctx_type.

Migration note: Context information and management are differents. Oldest implementation is NOT compatible.

8.3.10 Debug switch

The debug mode is enabled in *\Projects\<target>*

`\Applications\LoRaWAN\ LoRaWAN_<device>_LBM \Core\Inc\sys_conf.h` with the code below:

```
#define DEBUGGER_ENABLED 1 /* ON=1, OFF=0 */
```

The debug mode enables the SWD pins, even when the MCU goes in low-power mode.

Note: In order to enable a true low-power, `#define DEBUGGER_ENABLED` must be reset.

Some additional defines activate monitoring (probes) of some internal RF signal for debug:

```
#define DEBUG_SUBGHZSPI_MONITORING_ENABLED 0
#define DEBUG_RF_NRESET_ENABLED_ENABLED 0
#define DEBUG_RF_HSE32RDY_ENABLED_ENABLED 0
#define DEBUG_RF_SMPSRDY_ENABLED 0
#define DEBUG_RF_LDORDY_ENABLED 0
#define DEBUG_RF_DTB1_ENABLED 0
#define DEBUG_RF_BUSY_ENABLED 0
```

8.3.11 Low-power switch

When the system is in idle, it enters the low-power Stop 2 mode.

This entry in Stop 2 mode can be disabled in *\Projects\<target>*

`\Applications\LoRaWAN\ LoRaWAN_<device>_LBM \Core\Inc\sys_conf.h` with the code below:

```
#define LOW_POWER_DISABLE 0 /* Low power enabled = 0, Low power disabled = 1 */
```

where:

- Low power enabled = 0 means the MCU enters to Stop 2 mode
  - Stop 2 is a Stop mode with low-power regulator and VDD12I interruptible digital core domain supply OFF. Less peripherals are activated than in low-power Stop 1 mode to reduce power consumption. Refer to document [5] for more details.
- Low power disabled = 1 means the MCU enters only in Sleep mode.

8.3.12 Trace level

The trace mode is enabled in *\Projects\<target>*

`\Applications\LoRaWAN\ LoRaWAN_<device>_LBM \Core\Inc\sys_conf.h` with the code below:

```
#define APP_LOG_ENABLED 1
```

The trace level is selected in *\Projects\<target>*

`\Applications\LoRaWAN\ LoRaWAN_<device>_LBM \Core\Inc\sys_conf.h` with the code below:

```
#define VERBOSE_LEVEL VLEVEL_M
```

The following trace levels are proposed:

- VLEVEL_OFF: all traces disabled
- VLEVEL_L: functional traces enabled
- VLEVEL_M: debug traces enabled
- VLEVEL_H: all traces enabled

8.4 Device configuration summary for LoRaWAN_<device>_LBM application

Table 24. **Options for LoRaWAN® application configuration**

Project module	Detail	Switch option	Definition	Location
LoRa stack	Unique device identification	LORAWAN_DEVICE_EUI	End-Device IEEE extended unique Identifier (EUI)	se-identity.h
	Join EUI	LORAWAN_JOIN_EUI	Application or join server IEEE EUI (only used in OTAA)	
	Generated App key	LORAWAN_GEN_APP_KEY	Application root key used to derive application session keys	
	App root key	LORAWAN_APP_KEY	Network root key used to derive session keys at Join in OTAA	
	Supported regions	REGION_EU868	Regions supported by the device	lorawan_conf.h
		REGION_US915		
		REGION_AS923		
		REGION_AU915		
		REGION_CN470		
		REGION_CN470_RP_1_0		
		REGION_IN865		
		REGION_RU864		
		REGION_KR920		
	Read keys	KEY_EXTRACTABLE	Defines the read access of the keys in the memory.	
	Optional class	LORAMAC_CLASSB_ENABLED	End-device class B capability. NOT yet supported.	
Application	Tx trigger	EventType = TX_ON_TIMER	Tx trigger method	lora_app.c
	Duty cycle	APP_TX_DUTYCYCLE	Time period in seconds between two Tx sent. *Note: This time is modified by randomicity and time distribution as LoRaWAN® specifications recommended).*	lora_app.h
	App port	LORAWAN_USER_APP_PORT	LoRa® port used by the Tx data frame. *Note: 224 or 226 must not be used.*	
	Maximum data buffer size	LORAWAN_APP_DATA_BUFFER_MAX_SIZE	App data buffer size definition	
Application	Initial region	ACTIVE_REGION	Region used at startup	lora_app.h

Project module	Detail	Switch option	Definition	Location
Application	Certification mode	CERTIFICATION_MODE	Certification mode. It must be set to “true” to execute all LCTT Certification tests (except the Retransmission Backoff test).	lora-app.h
	Debug	DEBUGGER_ENABLED	Enables SWD pins.	sys_conf.h
	Low power	LOW_POWER_DISABLE	Disables low-power mode.	
	Trace enable	APP_LOG_ENABLED	Enables the trace mode.	
	Trace level	VERBOSE_LEVEL	Sets the trace level.	

Revision history

Table 25. **Document revision history**

Date	Version	Changes
24-Mar-2026	1	Initial release.

Contents

1 General information 2
2 STM32CubeWL overview 3
3 SubGHz HAL driver 5
3.1 SubGHz resources 5
3.2 SubGHz data transfers 5
4 BSP STM32WL Nucleo boards 6
4.1 Frequency band 6
4.2 RF switch 6
4.3 RF wake-up time 7
4.4 TCXO 7
4.5 Power regulation 7
4.6 STM32WL Nucleo board schematic 8
5 BSP B-WL5M-SUBG1 boards 9
5.1 RF switch 9
5.2 External components 9
6 LoRaWAN® stack description 10
6.1 LoRaWAN® specifications version 12
6.2 LoRaWAN® commissioning 12
6.3 LoRaWAN® pre-certification 12
6.4 Required STM32 peripherals to drive the radio 13
7 LoRaWAN Middleware description 14
7.1 LoRaWAN Middleware initialization 14
7.2 LoRaWAN Middleware MAC layer APIs 14
7.3 Application callbacks 16
8 LoRaWAN application 17
8.1 LoRaWAN application description 17
8.2 LoRaWAN user code section description 17
8.3 LoRaWAN® device App configuration 19
8.3.1 Activation method and keys 19
8.3.2 LoRaWAN® class 19
8.3.3 TX trigger 19
8.3.4 Duty cycle 20
8.3.5 Application port 20
8.3.6 Data buffer size 20

8.3.7 Adaptive data rate (ADR) . . . 20
8.3.8 LoRa® band selection . . . 20
8.3.9 NVM context management . . . 21
8.3.10 Debug switch . . . 22
8.3.11 Low-power switch . . . 22
8.3.12 Trace level . . . 22
8.4 Device configuration summary for LoRaWAN_<device>_LBM application . . . 23
Revision history . . . 25
List of tables . . . 28
List of figures . . . 29

List of tables

Table 1. Acronyms and terms . . . 2
Table 2. LoRaWAN® projects list . . . 2
Table 3. BSP radio switch . . . 6
Table 4. RF states versus switch configuration . . . 7
Table 5. BSP radio wake-up time . . . 7
Table 6. BSP radio TCXO . . . 7
Table 7. BSP radio SMPS . . . 7
Table 8. BSP radio switch . . . 9
Table 9. LoRaWAN® stack description . . . 11
Table 10. LoRaWAN® Middleware initialization . . . 14
Table 11. Stack event callback API . . . 14
Table 12. Enable Relay feature API for Relay device . . . 14
Table 13. Enable Relay feature API for End-Device with Relay feature . . . 15
Table 14. Join network API . . . 15
Table 15. Send Tx data API . . . 15
Table 16. LoRaWAN® FSM engine . . . 15
Table 17. User App schedule event . . . 15
Table 18. LoRaWAN® Middleware initialization . . . 16
Table 19. LoRaWAN user function . . . 17
Table 20. LoRaWAN API . . . 18
Table 21. Device configuration . . . 19
Table 22. NVM context sections . . . 21
Table 23. NVM context callback . . . 21
Table 24. Options for LoRaWAN® application configuration . . . 23
Table 25. Document revision history . . . 25

List of figures

Figure 1. Project file structure 4
Figure 2. NUCLEO-WL55JC schematic 8
Figure 3. LoRaWAN®stack architecture 10