SEMTECH

WIRELESS & SENSING PRODUCTS

# LoRa Basics™ Modem User Manual

# Table of Contents

# List of Figures

# List of Tables

# 1. Introduction

LoRa Basics™ Modem is an easy-to-use software library that simplifies the development of LoRaWAN® end-nodes. By using LoRa Basics Modem in their solution, developers can work through an event driven interface, at a high level of abstraction without needing to delve into the details of the LoRaWAN Standard.

LoRa Basics Modem allows developers to seamlessly integrate the services provided by LoRa Cloud™ into their applications. LoRa Basics Modem can be used to generate messages compatible with the LoRa Cloud Modem & Geolocation Services, including periodic Device Management messages.

## 1.1 Scope

This document describes the LoRa Basics Modem library (SWL2001). This version of the document pertains to version 3.3.0 of the library.

It should be read in conjunction with:

- LoRa Basics Modem SDK User Manual and associated SDK (SWSD001)
- LoRa Basics Modem Porting Guide

## 1.2 Supported Transceivers

This version of LoRa Basics Modem supports the following transceivers:

- LR1110 with firmware 0x0308.
- LR1120 with firmware 0x0102.
- LR1121 with firmware 0x0102.
- SX1261.
- SX1262.
- SX1268.
- SX1280.
- SX1281.

## 1.3 New Features and Changes

The following new features and changes have been introduced since version 3.2.4 of the LoRa Basics Modem library.

### 1.3.1 New Features

- Added support for LR1121.
- Now any of the LoRa Basics Modem define values can be changed by means of the *EXTRAFLAGS* parameter when issuing the make command.
- Added a port to the Nucleo-L073 board using the LL driver for minimal flash usage.
- Added a porting tool in the main examples to help during MCU porting.

### 1.3.2 Changes

- Updated LR11xx driver used to version v2.3.0.
- Updated SX126x driver used to version v2.2.0.
- Added response code assert in exti example.
- Removed temperature from exti example and replaced with 32-bit counter.
- Removed unused uart4-related functions and calls from *smtc_hal_l4*.
- Added randomness before call to any modem task that perfoms an uplink.
- Capped alarm timer at 864000s, i.e. 10 days.
- Set minimal reception window size to 16ms instead of 6ms to avoid ping slot issues in FSK.

### 1.3.3 Bugs Fixed

- Corrected typo which prevented using environment variables to set the MCU parameters.
- Fixed example so EUI and Keys are not overriden when code is built using the *CRYPTO=LR11XX_WITH_CREDENTIALS* option.
- Removed ARM-specific flag from common.mk.
- Fixed size error in *smtc_secure_element_get_pin()*.
- Fixed issue in LBT when TCXO startup is greater than default *RP_MARGIN_DELAY* value (8ms).
- Corrected tx done timestamp with known padding delay to avoid issue with the following rx windows (mainly seen on US and AU regions).
- Fixed LoRaWAN Link Adr Request channel mask control case 5 missing impact of 500MHz bank.
- In main_exti example, fixed blue button missing pin in irq configuration.

A comprehensive list of changes is provided in the changelog file of the library.

## 1.4 Example Application

LoRa Basics Modem includes an example which serves to illustrate the use of the library and can also be used as a starting point when developing an application. This application is defined in the file *utilities/user_app/main_examples/main_exti.c*. The example application joins the LoRaWAN network and remains in sleep mode until the user button is pressed, at which time it makes a temperature measurement and sends an uplink with the measured value.

The application was developed for use in a Nucleo-L476RG or Nucleo-L073 development boards and supports the LR11xx, SX126x, SX128x transceivers.

To compile the example for a Nucleo-L476RG board, perform the following steps:

1. Edit file *utilities/user_app/main_examples/example_options.h*:
    1. Replace the values of *USER_LORAWAN_DEVICE_EUI*, *USER_LORAWAN_JOIN_EUI*, and *USER_LORAWAN_APP_KEY* with the correct values for your device and network.
    2. Replace the value of *MODEM_EXAMPLE_REGION* by the value corresponding to your region.
2. In the utilities directory, issue the command *make <TARGET>* where target will be either *sx128x*, *lr1110*, *lr1120*, *sx1261*, or *sx1262* depending on your transceiver.
3. Plug in the Nucleo L476RG board.
4. Drag the file *utilities/build_<TARGET>/app_<TARGET>.bin* to the *NODE_L476RG* storage device.

The example application outputs informational messages through the UART. These messages can be viewed by means of a terminal emulator using the serial parameters: 921600 baud, 8 data bits, no parity and 1 stop bit (921600 8N1).

# 2. LoRa Basics Modem Architecture

LoRa Basics™ Modem (LBM) employs a modular design in which microcontroller and transceiver interaction is carried out through the use of abstraction layers (see figure 2.1).

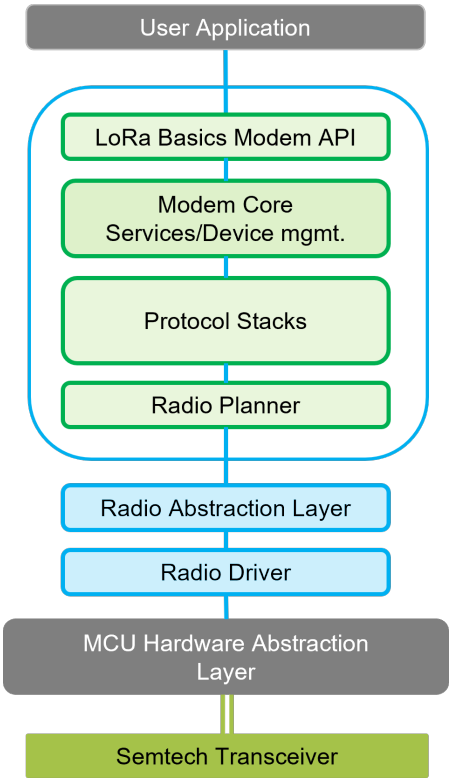


Fig. 2.1: LoRa Basics™ Modem Software Stack

This modular design is implemented as a set of components that interact through corresponding interfaces. From a high level perspective, an end-device application that uses LBM is formed by 3 main components:

- Hardware Abstraction Layer.
- Modem Core.
- Application Programming Interface.

## 2.1 Hardware Abstraction Layer

In LBM, hardware interaction is done through a Hardware Abstraction Layer. As a result, LBM can be ported to a new platform by implementing the functions defined in the Hardware Abstraction layer. Although LBM is designed to be platform independent, a minimum set of features is required from the Host MCU:

- Software MCU reset.
- A timer with 1ms resolution.
- SPI controller.
- A random number generator.
- Non-volatile storage for maintaining modem state.
- A dedicated (non-shared) GPIO MCU interrupt for the transceiver is recommended.

LBM has only been tested on 32-bit microcontrollers. As described in the next chapter, LBM is highly customizable and, as a result, Flash and RAM requirements will depend on the features included in the application. Worst-case stack usage is currently unknown however, to provide a rough guideline, the LBM Porting Guide lists RAM and Flash requirements for an example STM32L476 implementation.

Communication between LBM and the transceiver is done by means of an SPI bus and two signalling lines:

- SPI bus. LBM issues commands to, and reads responses from, the transceiver via the Host MCU SPI bus.
- Busy line. This line signals whether the transceiver is busy or ready to receive commands from the modem. It is high while the transceiver is busy and goes low when the transceiver is ready to receive commands.
- DIOx line. This line signals to the LBM that the transceiver has asynchronous event data pending.

To maintain platform independence, access to the above hardware peripherals is abstracted by means of an MCU Hardware Abstraction Layer (HAL). This Hardware Abstraction Layer is defined in the header file *smtc_modem_hal\smtc_modem_hal.h* and in the radio drivers. To port LBM to a given Host MCU, the functions listed in the HAL must be implemented. An example HAL implementation using the STM32L476 MCU as the target controller can be found in the LBM SDK (SWSD001).

Please refer to the LBM Porting Guide for further details.

## 2.2 Modem Core

The Modem Core component encapsulates the functionality of the LBM. This section describes some of the modules that constitute the Modem Core.

### 2.2.1 Modem Services

The Modem Services component provides support for LoRa Cloud™ services:

- Large File Upload.
- Reliable Octet Stream Encoding.
- Clock synchronization.
- Almanac Update.

Please refer to the LoRa Cloud documentation for further details of the above services.

### 2.2.2 Device Management

When using the LoRa Cloud services, an end device must generate periodic uplinks, which in turn must be submitted to LoRa Cloud by the Application Server. The payload of these end device uplinks must be formatted according to the LoRa Cloud requirements. When necessary, LoRa Cloud will also send downlinks to the device with information requests or with actions that must be performed by the device. Device management uplink formatting and scheduling, as well as device management downlink processing, is carried out by the LBM Device Management component.

### 2.2.3 Modem Supervisor

LBM requires concurrent execution of multiple tasks to respond to user requests and process asynchronous events. The Modem Supervisor schedules task execution (without real-time constraints) and manages the interaction between user requests and internal services.

### 2.2.4 Modem Crypto

LoRa Basics Modem includes a software-implemented Cryptographic Engine.

### 2.2.5 Radio Abstraction Layer

The Radio Abstraction Layer provides a generic interface for transceiver interaction. This interface is implemented in each of the transceiver-specific drivers. As a result, to use a specific transceiver, solution developers simply need to include the driver for the selected transceiver in their code.

## 2.3 Application Programming Interface

LBM provides an Application Programming Interface (API) for application development. The files that define the LBM API are located in the *smtc_modem_api* folder.

### 2.3.1 Event-Driven Software Interface

LBM uses an event-driven scheme to interact with the application. In this scheme, the result of many of the commands is an asynchronous event that must be handled by a callback provided by the application. The use of an event-driven scheme for application interaction allows the development of power-optimized applications.

### 2.3.2 Test API

The LBM API includes a set of test functions which are used to implement test functionality for regulatory conformance, certification, and functional testing. With the exception of the *smtc_modem_test_start* command, test commands are only available if test mode is active. Test mode can only be activated if the device has not yet joined a network and is not joining a network. Once test mode is active all other modem commands are disabled.

The test functions are defined in the *smtc_modem_api\smtc_modem_test_api.h* file.

### 2.3.3 LR11xx Extension

LR1110, LR1120 & LR1121 Transceivers are pre-provisioned during production with two identifiers:

- A globally unique *ChipEUI* number that identifies the device.
- A *SemtechJoinEUI* number that is re-used in a set of Semtech devices.

In addition to the above identifiers, LR1110, LR1120 & LR1121 Transceivers are also pre-provisioned with a unique Device Key, *DKEY*. With these numbers a Device PIN number, which is required to claim the device in the LoRa Cloud Join services, is calculated.

The LR11xx Extension of the LBM provides access to the LR1110, LR1120 & LR1121 provisioning functions. Please refer to the following for further details:

- LR1110 Transceiver User Manual.
- LR1120 Transceiver User Manual.
- LR1121 Transceiver User Manual.

- LoRa Cloud Join Server documentation.

# 3. LoRa Basics Modem Integration with LoRa Cloud

LoRa Cloud™ is a set of services that provide simple solutions to common tasks related to LoRaWAN® networks and LoRa®-enabled devices. These tasks include comprehensive device telemetry, device and application configuration, clock synchronization, and advanced data transport services with configurable robustness against packet loss and transparent data fragmentation. LoRa Cloud services simplify the process of developing managed endpoint solutions and make LoRaWAN technology more accessible to application developers.

Some of the services provided by LoRa Cloud are:

1. Periodic info messages
    - System status, firmware version
    - Charge, temperature
    - Downlink signal quality
    - Uptime, time since last downlink
    - Device EUI, Join EUI
    - Application-specific status bytes
2. Management commands
    - Mute, rejoin
    - Soft reset / factory reset
    - Set Adaptive Data Rate profile
    - Change reporting interval of periodic info messages
    - Retrieve Crash Log
3. Advanced protocols
    - Advanced Transport Services
        - Large file upload (LFU)
        - Reliable Octet Stream Encoding (ROSE)
    - Clock synchronization over-the-air

To integrate the LoRa Cloud services into a solution, the end device may submit periodic uplinks with information about its state, which are then used to update the representation of the device in the LoRa Cloud. These uplinks are processed by the LoRaWAN Network Server or the LoRaWAN Application Server, and subsequently submitted to the LoRa Cloud through calls to the LoRa Cloud API endpoints as illustrated in figure 3.1. Uplinks that must be submitted to the LoRa Cloud are identified by means of the port number.

When an uplink destined for the LoRa Cloud is received by the LoRaWAN Network Server or the LoRaWAN Application Server, the payload and other fields, such as the device EUI, must be extracted from the uplink and then submitted to the LoRa Cloud by means of a call to the appropriate LoRa Cloud API endpoint. The response obtained by the LoRaWAN Network Server or the LoRaWAN Application Server from the LoRa Cloud as a result of the API call may contain a downlink payload. If a downlink is present in the LoRa Cloud API call response, then said downlink must be sent back to the device.

A detailed description of the LoRa Cloud APIs can be found in the LoRa Cloud on-line documentation. Examples of the end device interaction with LoRa Cloud can be found in the LoRa Basics Modem SDK User Manual and associated SDK (SWSD001).

Creation of the uplinks required by the LoRa Cloud and processing of the resulting downlinks is done by the LBM Device Management component. The information that is sent to the LoRa Cloud, as well as the send periodicity, can be tuned through the Device Management fields and functions of the LBM Device Management module.

Semtech provides several LoRaWAN Application Server examples that illustrate, among other features, uplink and downlink processing:

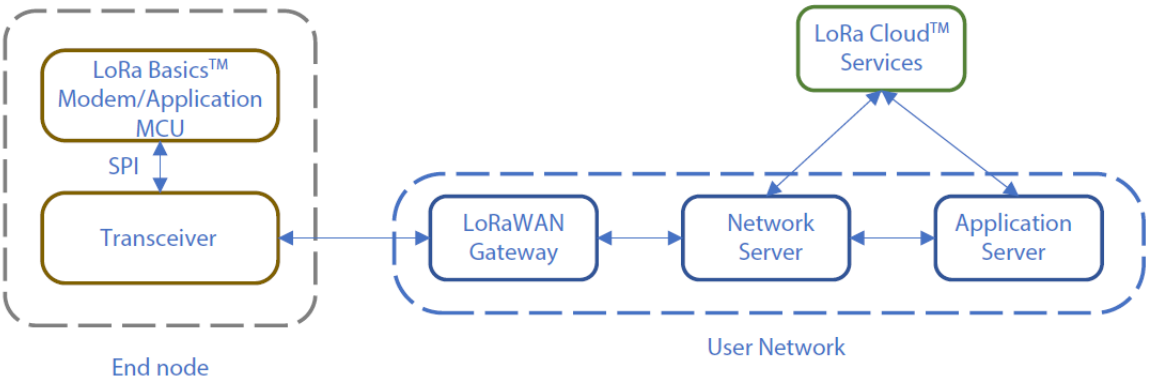


Fig. 3.1: LoRa Basics™ Modem Network Architecture

1. node-red-contrib-loracloud-utils provides a Node-RED based LoRaWAN Application Server with LoRa Cloud integration.

2. SWNW001 is an AWS Lambda based LoRaWAN Application Server. Developed in Python, this Application Server uses the *Serverless Framework <https://www.serverless.com/>* and the AWS IoT Core Wireless to interface between the LoRa Edge™ Tracker Reference Design and the LoRa Cloud.

3. SWNW003 is a LoRaWAN Application Server implemented as an Azure Serverless Function. Programmed in Python, SWNW003 uses an Azure IoT Hub to interface between the LoRa Edge Tracker Reference Design and the LoRa Cloud.

For a detailed description of the interaction between an end node and the LoRa Cloud Modem & Geolocation Services please refer to the LoRa Basics™ Modem and LoRa Edge™ documentation available in the LoRa Developer Portal.

# 4. LoRa Basics Modem Options

The LoRa Basics Modem library is highly customizable. The following sections list the compile time options. The list of compile time options can also be obtained by issuing the command *make help*.

## 4.1 Radio Targets

Support for the following transceivers can be selected at build time:

- *sx128x* - SX1280 & SX1281 Transceivers.
- *lr1110* - LR1110 Transceiver.
- *lr1120* - LR1120 Transceiver.
- *lr1121* - LR1121 Transceiver.
- *sx1261* - SX1261 Transceiver.
- *sx1262* - SX1262 Transceiver.
- *sx1268* - SX1268 Transceiver.

LoRa Basics™ Modem (LBM) should be built for a specific transceiver by using the *basic_modem_<TARGET>* parameter.

## 4.2 MCU Flags

MCU flags are passed to the compiler by means of the *MCU_FLAGS* build parameter. The flags are set as a single text string. For example, to build LoRa Basics Modem for an STM32L4 MCU the following *MCU_FLAGS* is used:

*MCU_FLAGS="-mcpu=cortex-m4 -mthumb -mfpu=fpv4-sp-d16 -mfloat-abi=hard"*

## 4.3 Regions

Support for one or more of the following regions can be selected at compile time:

- *AS_923* - AS923MHz ISM Band. Group selection is performed at runtime through the API.
- *AU_915* - AU915-928MHz ISM Band.
- *CN_470* - CN470-510MHz Band.
- *CN_470_RP_1_0* - CN470-510MHz Band. LoRaWAN Regional Parameters v1.0 version.
- *EU_868* - EU863-870MHz ISM Band.
- *IN_865* - IN865-867MHz ISM Band.
- *KR_920* - KR920-923MHz ISM Band.
- *RU_864* - RU864-870MHz ISM Band.
- *US_915* - US902-928MHz ISM Band.
- *WW_2G4* - Emulation of the LoRaWAN Standard for the 2.4GHz global ISM band.

The supported regions are selected through the *REGION* build option. If the user does not explicitly select one or more regions, all regions are included at compile time.

## 4.4 Regional Parameters

Two versions of the LoRaWAN Regional Parameters are supported:

- *RP2_101* - LoRaWAN Regional Parameters vRP002-1.0.1.
- *RP2_103* - LoRaWAN Regional Parameters vRP002-1.0.3 (Default). This option includes support for LR-FHSS.

The LoRaWAN Regional Parameters version is selected at compile time through the *RP_VERSION* compile option.

## 4.5 Cryptographic Engine Selection

LR11xx Transceivers include a Cryptographic Engine which provides a dedicated hardware accelerator for AES-128 encryption based algorithms. This Cryptographic Engine improves the power efficiency of cryptographic operations and reduces the code size of the software stack. Alternatively, LBM includes a software-implemented Cryptographic Engine. The selection of the Cryptographic Engine to use is done through the *CRYPTO* parameter. The following options are available:

- *SOFT* - Use the LoRa Basics Modem Cryptographic Engine (Default).
- *LR11XX* - Use the LR11xx Cryptographic Engine.
- *LR11XX_WITH_CREDENTIALS* - Use the LR11xx Cryptographic Engine with pre-provisioned EUIs and keys.

## 4.6 GNSS

LR1110 & LR1120 Transceivers include a GNSS receiver that allows fast and energy efficient indoor/outdoor geolocation. Use of the GNSS receiver can be enabled or disabled by means of the following parameter:

- *USE_GNSS=yes/no* - Enable or disable the use of the LR11xx GNSS receiver.

## 4.7 Debug Options

The following options can be enabled for debugging:

- *VERBOSE=yes/no* - Increase build verbosity (default: no).
- *DEBUG=yes/no* - Enable debugging options (default: no).
- *MODEM_TRACE=yes/no* - Choose to enable or disable modem trace print (default: yes).

## 4.8 Example

For example to compile LoRa Basics Modem for an STM32L4 CPU with an SX1262 transceiver, with support for US_915 region using regional parameters RP2-1.0.1, without modem trace print and with increase build verbosity the following command line should be used:

*make basic_modem_sx1262 MCU_FLAGS="-mcpu=cortex-m4 -mthumb -mfpu=fpv4-sp-d16 -mfloat-abi=hard" REGION=US_915 RP_VERSION=RP2_101 CRYPTO=SOFT MODEM_TRACE=no VERBOSE=yes*

# 5. Developing an Application with LoRa Basics™ Modem

This chapter describes the required sequences of commands to enable functionalities offered by LoRa Basics™ Modem. Reference implementations can be found in the LoRa Basics Modem SDK (SWSD001).

## 5.1 EXTRAFLAGS Usage

Build time parameters are passed to the compiler through the use of the EXTRAFLAGS parameter. For example, to change the internal radio planner margin delay value the following build command line can be used:

*make basic_modem_<TARGET> MCU_FLAGS=xxx EXTRAFLAGS="-DRP_MARGIN_DELAY=12"*

## 5.2 Porting

The following items must be implemented before starting to develop an application:

- The driver Hardware Abstraction Layer (HAL) corresponding to the selected transceiver
- The Radio Abstraction Layer (RAL) Board Support Package (BSP) corresponding to the selected transceiver
- The LoRa Basics Modem Hardware Abstraction Layer corresponding to the selected MCU

Please refer to the LoRa Basics™ Modem Porting Guide for further details.

## 5.3 Radio Selection

LoRa Basics Modem includes support for multiple transceivers. The driver for the specific transceiver used in a solution must be instantiated through a call to one of the following macros:

- *RALF_LR11XX_INSTANTIATE( ctx )*
- *RALF_SX126X_INSTANTIATE( ctx )*
- *RALF_SX128X_INSTANTIATE( ctx )*

The call to one of the above macros establishes the link between the RAL and the specific driver used in the solution.

## 5.4 Reset and Initialize the System

In order to reset the LoRa Basics Modem, the user first has to perform an initialization by calling *smtc_modem_init()*, where two parameters are required:

- The radio to be used
- The callback to get the event from the Modem

After the first call to *smtc_modem_run_engine()*, the event SMTC_MODEM_EVENT_RESET is triggered to let the user know that the modem is now ready to use. It is strongly recommended to not call any modem-related command between the initialization and the event. If you do, LoRa Basics Modem may have an undefined behaviour.

The application must call *smtc_modem_run_engine* periodically to advance the modem state machine.

## 5.5 Get Version Information

Various information can be fetched from the API:

- *smtc_modem_get_modem_version()* to get the LoRa Basics Modem version
- *smtc_modem_get_lorawan_version()* to get the LoRaWAN® version implemented in LoRa Basics Modem
- *smtc_modem_get_regional_params_version()* to get the LoRaWAN regional parameters version implemented in LoRa Basics Modem

## 5.6 Fetch an Event

The callback that is shared during initialization permits the user to be informed of incoming events. When the callback is called by LoRa Basics Modem, the user can retrieve the pending events by calling *smtc_modem_get_event()*.

The user can unstack pending events with successive calls to *smtc_modem_get_event()* until SMTC_MODEM_EVENT_NONE is returned.

## 5.7 Join a LoRaWAN® Network

Before initiating a join procedure, the user has to set some LoRaWAN® parameters - the order does not matter:

- LoRaWAN region with *smtc_modem_set_region()* (can only be changed when the modem is not joining or joined)
- LoRaWAN class with *smtc_modem_set_class()*

The user can configure the EUIs and key parameters with the following functions:

- DevEUI with *smtc_modem_set_deveui()*
- JoinEUI with *smtc_modem_set_joineui()*
- AppKey with *smtc_modem_set_nwkkey()*

The user can also configure the type of network they are using - private or public - by calling *smtc_modem_set_network_type()*.

Once all parameters are set, the user can start joining a LoRaWAN network by calling *smtc_modem_join_network()*. The user is informed of the evolution of the procedure via two events:

- SMTC_MODEM_EVENT_JOINED: a join accept was received; the join procedure is over
- SMTC_MODEM_EVENT_JOINFAIL: nothing was received after the join request; the join procedure keeps going

The user can cancel an ongoing join procedure or leave an already-joined network by calling *smtc_modem_leave_network()*.

## 5.8 LoRaWAN® Class B

Before enabling LoRaWAN Class B, LoRa Basics Modem has to be connected to a network in class A.

Once a network is joined, the two following actions have to be performed - the order does not matter:

- Enable the time synchronization service by calling *smtc_modem_time_start_sync_service()* with *SMTC_MODEM_TIME_MAC_SYNC* parameter
- Configure the ping slot periodicity by calling *smtc_modem_class_b_set_ping_slot_periodicity()* and request an update of the parameter to the LoRaWAN Network Server by calling *smtc_modem_lorawan_class_b_request_ping_slot_info()*

As soon as both events *SMTC_MODEM_EVENT_TIME* (with a status different from *SMTC_MODEM_EVENT_TIME_NOT_VALID*) and *SMTC_MODEM_EVENT_CLASS_B_PING_SLOT_INFO* (with a status equal to *SMTC_MODEM_EVENT_CLASS_B_PING_SLOT_ANSWERED*) are received, it is possible to switch to Class B by calling *smtc_modem_set_class()* with *SMTC_MODEM_CLASS_B* parameter.

LoRa Basics Modem is effectively in Class B when the event *SMTC_MODEM_EVENT_CLASS_B_STATUS* (with a status equal to *SMTC_MODEM_EVENT_CLASS_B_READY*) is received. The stack has acquired a beacon and started to open ping slots.

Class B downlinks are available through the event *SMTC_MODEM_EVENT_DOWNDATA* with status *SMTC_MODEM_EVENT_DOWNDATA_WINDOW_RXB*.

## 5.9 LoRaWAN® Multicast

Up to 4 multicast groups can be configured by calling *smtc_modem_multicast_set_grp_config()* with the following parameters for each group to be configured:

- ID
- Address
- Network session key
- Application session key

It is possible to read back the group address of a given group ID by calling *smtc_modem_multicast_get_grp_config()*.

All active multicast sessions have to be stopped before switching to class A (see hereafter to know how to stop a multicast session).

### 5.9.1 LoRaWAN® Multicast in Class B

When LoRa Basics Modem is set in Class B (see *LoRaWAN® Class B*) and multicast groups are configured (see *LoRaWAN® multicast*), each session can be started by calling *smtc_modem_multicast_class_b_start_session()* with the following parameters:

- Group ID
- Frequency
- Datarate
- Ping slot

Class B multicast downlink are available through the event *SMTC_MODEM_EVENT_DOWNDATA* with status *SMTC_MODEM_EVENT_DOWNDATA_WINDOW_RXB_MC_GRPx* (x being the multicast group ID).

It is possible to read back the status of the current session by calling *smtc_modem_multicast_class_b_get_session_status()*.

A multicast session can be stopped by calling *smtc_modem_multicast_class_b_stop_session()*.

### 5.9.2 LoRaWAN® Multicast in Class C

When LoRa Basics Modem is set in Class C and multicast groups are configured (see *LoRaWAN® multicast*), each session can be started by calling *smtc_modem_multicast_class_c_start_session()* with the following parameters:

- Group ID
- Frequency
- Datarate

If either the frequency or the datarate of the first session started is different from the one already used in class C, it will automatically switch to this new configuration and unicast messages cannot be received anymore. Then, if either the frequency or the datarate of one of the following sessions is different from the one already used, the session cannot be started. To be able to start this session, all already started sessions have to be stopped by calling *smtc_modem_multicast_class_c_stop_session()*.

Class C multicast downlinks are available through the event *SMTC_MODEM_EVENT_DOWNDATA* with status *SMTC_MODEM_EVENT_DOWNDATA_WINDOW_RXC_MC_GRPx* (x being the multicast group ID).

It is possible to read back the status of the current session by calling *smtc_modem_multicast_class_c_get_session_status()*.

A multicast session can be stopped by calling *smtc_modem_multicast_class_c_stop_session()*.

## 5.10 Send Data Over LoRaWAN

The user can send a standard uplink, while being connected, with the following functions:

- *smtc_modem_request_uplink()*
- *smtc_modem_request_emergency_uplink()* to send data with the highest priority, while ignoring duty cycle restriction
- *smtc_modem_request_empty_uplink()* to send an empty payload with an optional FPort field; can be used to create a downlink opportunity

## 5.11 Receive Data Over LoRaWAN

When a downlink is received that contains data for the application, the event *SMTC_MODEM_EVENT_DOWNDATA* is triggered with the payload - if any. There are also the following metadata:

- RSSI
- SNR
- Reception window
- FPort
- FPending bit
- Frequency
- Datarate

## 5.12 Manage a LoRaWAN Connection Lifecycle

Once a LoRaWAN network is joined, the user can configure several parameters while being connected:

- The ADR profile by calling *smtc_modem_adr_set_profile()*
- The connection timeout thresholds by calling *smtc_modem_connection_timeout_set_thresholds()*
- The number of transmissions for each unconfirmed uplink by calling *smtc_modem_set_nb_trans()*

---

**Note:** All the functions mentioned above can also be called before joining a network.

---

It is also possible to get information about the connection:

- Current status of the duty cycle limitation, if any, by calling *smtc_modem_get_duty_cycle_status()*
- Current status of the connection timeouts by calling *smtc_modem_connection_timeout_get_current_values()*

- Maximum size of the next uplink by calling *smtc_modem_get_next_tx_max_payload()*
- Available datarates by calling *smtc_modem_get_available_datarates()*
- Number of consecutive uplinks without downlink by calling *smtc_modem_lorawan_get_lost_connection_counter()*

The user can also request a link check by calling *smtc_modem_lorawan_request_link_check()*. This is a MAC command that will be responded to using a link check answer.

## 5.13 Get Time from the Network Server or Application Server

The user can configure LoRa Basics Modem to perform an automatic time synchronization:

- Configure the interval between two time requests with *smtc_modem_time_set_sync_interval_s()*
- Configure the delay since the last synchronization to consider the time as not valid anymore with *smtc_modem_time_set_sync_invalid_delay_s()*
- (Optional) Configure the ALCsync port with *smtc_modem_time_set_alcsync_fport()* if the service to be used is SMTC_MODEM_TIME_ALC_SYNC

The user must ensure that the interval is lower than the desynchronization delay. This delay takes into account the maximal time deviation tolerated by the targeted application and the clock drift. It is advised to configure the interval before starting the service. If done after, the first interval is set to 36 hours - the configured value is taken into account only for the next interval.

Once all parameters are set, the user can select and start the time synchronization service by calling *smtc_modem_time_start_sync_service()*. It can be stopped anytime by calling *smtc_modem_time_stop_sync_service()*.

If SMTC_MODEM_TIME_MAC_SYNC is set with *smtc_modem_time_start_sync_service()*, an event is triggered in the following cases:

- When a time synchronization is successful (status set to SMTC_MODEM_EVENT_TIME_VALID)
- When a time synchronization is not successful and the local time is not synchronized (status set to SMTC_MODEM_EVENT_TIME_NOT_VALID)
- When a time synchronization request does not get an answer but the local time is still synchronized (status set to SMTC_MODEM_EVENT_TIME_VALID_BUT_NOT_SYNC)

If SMTC_MODEM_TIME_ALC_SYNC is set with *smtc_modem_time_start_sync_service()*, an event is triggered in the following cases:

- When a time synchronization is successful (status set to SMTC_MODEM_EVENT_TIME_VALID)
- When a time synchronization is not successful and the local time is not synchronized (status set to SMTC_MODEM_EVENT_TIME_NOT_VALID)

If *smtc_modem_time_trigger_sync_request()* is called, an event is triggered in the following cases:

- When a time synchronization is successful (status set to SMTC_MODEM_EVENT_TIME_VALID)
- When a time synchronization is not successful and the local time is not synchronized (status set to SMTC_MODEM_EVENT_TIME_NOT_VALID)
- When a time synchronization request does not get an answer but the local time is still synchronized (status set to SMTC_MODEM_EVENT_TIME_VALID_BUT_NOT_SYNC)

The user can fetch the current time by calling *smtc_modem_get_time()*. Before a time synchronization is performed, it returns SMTC_MODEM_RC_NO_TIME.

The user can request a time synchronization besides those sent on a periodic basis by calling *smtc_modem_time_trigger_sync_request()*.

## 5.14 Send Information with the Device Management Services

Before initiating a periodic information report, the user has to set several parameters; the order does not matter:

- LoRaWAN FPort, with *smtc_modem_dm_set_fport()*
- Fields to be reported, with *smtc_modem_dm_set_info_fields()*
- (Optional) User data to be reported, with *smtc_modem_dm_set_user_data()* - useful only if SMTC_MODEM_DM_FIELD_APP_STATUS is part of the *user_data* parameter given to *smtc_modem_dm_set_info_fields()*
- Interval between two periodic information reports, with *smtc_modem_dm_set_info_interval()*

If the user modifies the interval between two periodic device management information reports, it cancels the next planned uplink and schedules a new one that will be sent after the newly chosen interval.

The user can request a device management information report besides those sent on periodic basis by calling *smtc_modem_dm_request_single_uplink()* where fields different from the ones set with *smtc_modem_dm_set_info_fields()* can be reported.

## 5.15 Update the Almanac with Device Management Services

The field SMTC_MODEM_DM_FIELD_ALMANAC_STATUS has a specific behaviour in device managemement services. When enabled, either with *smtc_modem_dm_set_info_fields()* or *smtc_modem_dm_request_single_uplink()*, it sends the current state of the almanac to the LoRa Cloud™ Modem & Geolocation Services, which can decide to trigger an incremental update if needed.

The event SMTC_MODEM_EVENT_ALMANAC_UPDATE is triggered when an update is done.

This service is only compatible with LR1110 & LR1120 transceivers.

## 5.16 Send Data with the Stream Service

The user can stream data and let LoRa Basics Modem deal with MTU limits.

Before initiating a stream, the user has to set several parameters by calling *smtc_modem_stream_init()*:

- LoRaWAN FPort
- Encryption mode
- Redundancy ratio

Once configured, the user can add data to the stream buffer by calling *smtc_modem_stream_add_data()*. It is recommended to check the free space in the buffer with *smtc_modem_stream_status()* before adding data.

The event SMTC_MODEM_EVENT_STREAMDONE is triggered when the last byte of the stream buffer is sent. This event is triggered for information purposes; there is no need to wait for it before adding data to the stream buffer.

## 5.17 Send Data with the File Upload Service

The user can send up to 8180 bytes (called a file) and let LoRa Basics Modem deal with MTU limits.

Before initiating a file upload, the user has to set several parameters by calling *smtc_modem_file_upload_init()*:

- Application index
- Encryption mode
- File to be sent and its size
- Delay between two fragment uploads

Once the configuration is done, the user can start the transfer by calling *smtc_modem_file_upload_start()*. The user can abort the transfer by calling *smtc_modem_file_upload_reset()*.

The event SMTC_MODEM_EVENT_UPLOADDONE is triggered when:

- The LoRa Cloud Modem & Geolocation Services acknowledges the reception with a dedicated downlink message (status set to SMTC_MODEM_EVENT_UPLOADDONE_SUCCESSFUL)
- No acknowledgment is received after the last upload (status set to SMTC_MODEM_EVENT_UPLOADDONE_ABORTED)

## 5.18 Configure a Timer

LoRa Basics Modem offers the possibility to configure a timer that can trigger SMTC_MODEM_EVENT_ALARM event.

To start a timer, the user has to call *smtc_modem_alarm_start_timer()*. It can be stopped by calling *smtc_modem_alarm_clear_timer()*. A call to *smtc_modem_alarm_get_remaining_time()* returns the remaining time, in seconds, before the event is triggered.

## 5.19 Interleave Direct Radio Access

It is possible to prevent LoRa Basics Modem from accessing the transceiver and let the application have direct access to it without being interrupted.

This can be done through two functions:

- *smtc_modem_suspend_before_user_radio_access()* - to be called before any direct access to the transceiver
- *smtc_modem_resume_after_user_radio_access()* - to be called after the last direct access to the transceiver

---

**Note:** As soon as *smtc_modem_suspend_before_user_radio_access()* is called, LoRa Basics Modem will not perform any radio-related operation until *smtc_modem_resume_after_user_radio_access()* is called. For instance, class B or C operations - if enabled - cannot be performed.

---

# 6. Known Limitations

## 6.1 File Upload Service

### 6.1.1 Known Limitation #1

(present since v2.1.0)

In case LoRa Basics™ Modem is operating in US915 region with datarate DR0, files smaller than 13 bytes are not properly sent and cannot be reconstructed on LoRa Cloud side.

## 6.2 Charge Computation

### 6.2.1 Known Limitation #1

(present since v2.1.0)

Values returned by *smtc_modem_get_charge()* for regions CN470 and CN470_RP1 are not accurate.

### 6.2.2 Known Limitation #2

Values returned by *smtc_modem_get_charge()* for the LR-FHSS based datarate are not accurate.

## 6.3 Time Service

### 6.3.1 Known Limitation #1

In case the ALC_SYNC time service is used, when a valid time is received, the generated SMTC_MODEM_EVENT_TIME event will show a ghost missed event.

# 7. LoRa Basics™ Modem API Reference

## 7.1 LoRa Basics™ Modem Event Codes Definitions

When the LoRa Basics™ Modem (LBM) is initialized, a callback function is passed to the *smtc_modem_init()* function. This callback is used to signal that a new asynchronous event has occurred. The following event types are used in LBM.

Table 7.1: LoRa Basics™ Modem Events

| Events | Description |
|---|---|
| SMTC_MODEM_EVENT_RESET | Modem has been reset |
| SMTC_MODEM_EVENT_ALARM | Alarm timer expired |
| SMTC_MODEM_EVENT_JOINED | Network successfully joined |
| SMTC_MODEM_EVENT_TXDONE | Frame transmitted |
| SMTC_MODEM_EVENT_DOWNDATA | Downlink data received |
| SMTC_MODEM_EVENT_UPLOADDONE | File upload completed |
| SMTC_MODEM_EVENT_SETCONF | Configuration was changed by Device Management |
| SMTC_MODEM_EVENT_MUTE | Modem muted / un-muted by Device Management |
| SMTC_MODEM_EVENT_STREAMDONE | Stream upload completed (stream data buffer depleted) |
| SMTC_MODEM_EVENT_JOINFAIL | Attempt to join network failed |
| SMTC_MODEM_EVENT_TIME | Update on time happened (synced or invalid) |
| SMTC_MODEM_EVENT_TIMEOUT_ADR_CHANGED | ADR profile was switched to network controlled |
| SMTC_MODEM_EVENT_NEW_LINK_ADR | New link ADR requested by network |
| SMTC_MODEM_EVENT_LINK_CHECK | Link Check answered by network |
| SMTC_MODEM_EVENT_ALMANAC_UPDATE | An almanac update has been received |
| SMTC_MODEM_EVENT_USER_RADIO_ACCESS | Radio callback when user uses the radio by itself |
| SMTC_MODEM_EVENT_CLASS_B_PING_SLOT_INFO | Ping Slot Info answered by network. |
| SMTC_MODEM_EVENT_CLASS_B_STATUS | Downlink class B is ready or not. |
| SMTC_MODEM_EVENT_MIDDLEWARE_1 | Reserved for Middleware. |
| SMTC_MODEM_EVENT_MIDDLEWARE_2 | Reserved for Middleware. |
| SMTC_MODEM_EVENT_MIDDLEWARE_3 | Reserved for Middleware. |
| SMTC_MODEM_EVENT_NONE | No event available |

**Note:** Events can be retrieved through the function *smtc_modem_get_event()*.

## 7.2 LoRa Basics™ Modem Device Management Fields

LBM periodically sends status messages that update the device record in the LoRa Cloud™ Modem & Geolocation Services. The fields that are to be included in these status messages can be read and adjusted through the *smtc_modem_dm_get_info_fields()* and *smtc_modem_dm_set_info_fields()* functions. The following table lists the names of the Device Management fields that can be reported to the LoRa Cloud Modem and Geolocation Services.

Table 7.2: LoRa Basics™ Modem Device Management Fields

| Field | Description |
|---|---|
| SMTC_MODEM_DM_FIELD_STATUS | Modem status |
| SMTC_MODEM_DM_FIELD_CHARGE | Charge counter [mAh] |
| SMTC_MODEM_DM_FIELD_VOLTAGE | Supply voltage [1/50 V] |
| SMTC_MODEM_DM_FIELD_TEMPERATURE | Junction temperature [deg Celsius] |
| SMTC_MODEM_DM_FIELD_SIGNAL | RSSI and SNR of the last downlink |
| SMTC_MODEM_DM_FIELD_UP_TIME | Duration since last reset [h] |
| SMTC_MODEM_DM_FIELD_RX_TIME | Duration since last downlink [h] |
| SMTC_MODEM_DM_FIELD_ADR_MODE | ADR profile (0-3) |
| SMTC_MODEM_DM_FIELD_JOIN_EUI | Join EUI |
| SMTC_MODEM_DM_FIELD_INTERVAL | Reporting interval [values 0-63, units s/m/h/d] |
| SMTC_MODEM_DM_FIELD_REGION | Regulatory region |
| SMTC_MODEM_DM_FIELD_RST_COUNT | Modem reset count |
| SMTC_MODEM_DM_FIELD_DEV_EUI | Device EUI |
| SMTC_MODEM_DM_FIELD_SESSION | Session id / join nonce |
| SMTC_MODEM_DM_FIELD_CHIP_EUI | Chip EUI |
| SMTC_MODEM_DM_FIELD_APP_STATUS | Application-specific status |
| SMTC_MODEM_DM_FIELD_ALMANAC_STATUS | Almanac status |

**Note:** Additionally to the periodic status reporting, the LoRa Cloud Modem & Geolocation Services can explicitly request a status update by means of a *GetInfo* downlink. However, these requests are handled internally by the LBM and are therefore not exposed by the API to the application.

## 7.3 LoRa Basics™ Modem API Return Codes

The LBM API functions return a value of type *smtc_modem_return_code_t* called the *Return Code*. The *Return Code* signals the result of the functions execution. The following table lists the *Return Codes* used in the LBM API.

Table 7.3: LoRa Basics™ Modem API Return Codes

| Return Code | Description |
|---|---|
| SMTC_MODEM_RC_OK | Command executed without errors |
| SMTC_MODEM_RC_NOT_INIT | Command not initialized |
| SMTC_MODEM_RC_INVALID | Command parameters invalid |
| SMTC_MODEM_RC_BUSY | Command cannot be executed now |
| SMTC_MODEM_RC_FAIL | Command execution failed |
| SMTC_MODEM_RC_BAD_SIZE | Size check failed |
| SMTC_MODEM_RC_NO_TIME | No time available |
| SMTC_MODEM_RC_INVALID_STACK_ID | Invalid *stack_id* parameter |

**Note:** Command output values must not be read if the *Return Code* differs from *SMTC_MODEM_RC_OK*.

## 7.4 LoRa Basics™ Modem Datarate Profiles

The following table lists the available datarate profiles.

Table 7.4: LoRa Basics™ Modem Datarate Profiles

| Return Code | Description |
|---|---|
| SMTC_MODEM_ADR_PROFILE_NETWORK_CONTROLLED | Network Server controlled for static devices |
| SMTC_MODEM_ADR_PROFILE_MOBILE_LONG_RANGE | Long range distribution for mobile devices |
| SMTC_MODEM_ADR_PROFILE_MOBILE_LOW_POWER | Low power distribution for mobile devices |
| SMTC_MODEM_ADR_PROFILE_CUSTOM | User defined distribution |

## 7.5 LoRa Basics™ Modem API Functions

LBM provides a set of functions that implement the low-level LoRaWAN network communications. This allows developers to work at a high level, using an API composed of single-command functions that perform all of the interactions with the LoRaWAN network. This chapter details the functions that make up the LBM API.

### 7.5.1 smtc_modem_abort_extended_uplink()

```
smtc_modem_return_code_t smtc_modem_abort_extended_uplink (
                                    uint8_t stack_id,
                                    uint8_t extended_uplink_id
                                    )
```

**Brief**

This feature is introduced for future libraries and functions, it is NOT recommended for the user to call this function. This feature requires a special compilation option to be activated.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *extended_uplink_id* | ID of the queue for extended uplink should be equal to 1 or 2 |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | Extended_uplink_id not equal to 1 or 2 |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | Modem is not available (suspended, muted or not joined) |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.2 smtc_modem_adr_get_profile()

```
smtc_modem_return_code_t smtc_modem_adr_get_profile (
                                    uint8_t stack_id,
                                    smtc_modem_adr_profile_t* adr_profile
                                    )
```

**Brief**

Get the current adaptative data rate (ADR) profile.

**Remarks**

Valid datarate profiles are listed in table 7.4.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *adr_profile* | Current ADR profile |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | Parameter *adr_profile* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.3 smtc_modem_adr_set_profile()

```
smtc_modem_return_code_t smtc_modem_adr_set_profile (
                                   uint8_t stack_id,
                                   smtc_modem_adr_profile_t* adr_profile
                                   const uint8_t  adr_custom_data[SMTC_MODEM_CUSTOM_ADR_DATA_LENGTH]
                                   )
```

**Brief**

Set the adaptative data rate (ADR) profile.

**Remarks**

If *SMTC_MODEM_ADR_PROFILE_CUSTOM* is selected, custom data are taken into account. Valid datarate profiles are listed in table 7.4.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *adr_profile* | ADR profile to be configured |
| [in] | *adr_custom_data* | Definition of the custom ADR profile |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | One or more invalid parameter |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.4 smtc_modem_alarm_clear_timer()

```
smtc_modem_return_code_t smtc_modem_alarm_clear_timer (void)
```

**Brief**

Stop and clear the alarm timer.

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_NOT_INIT* | No alarm timer currently running |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.5 smtc_modem_alarm_get_remaining_time()

```
smtc_modem_return_code_t smtc_modem_alarm_get_remaining_time  (
                                uint32_t*  remaining_time_in_s
                                )
```

**Brief**

Get the number of seconds remaining before the alarm triggers an event.

**Parameters**

| | | |
|---|---|---|
| [out] | *remaining_time_in_s* | Number of seconds remaining before the alarm triggers an event |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_NOT_INIT* | No alarm timer currently running |
| *SMTC_MODEM_RC_INVALID* | Parameter *remaining_time_in_s* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.6 smtc_modem_alarm_start_timer()

```
smtc_modem_return_code_t smtc_modem_alarm_start_timer  (
                                uint32_t alarm_timer_in_s
                                )
```

**Brief**

Set and start the alarm timer (up to 864000s, i.e. 10 days).

**Parameters**

| | | |
|---|---|---|
| [in] | *alarm_timer_in_s* | The alarm timer in seconds |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *alarm_timer_in_s* exceeds max value of 864000s (10 days) |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.7 smtc_modem_class_b_get_ping_slot_periodicity()

```
smtc_modem_return_code_t smtc_modem_class_b_get_ping_slot_periodicity  (
                                uint8_t stack_id,
                                smtc_modem_class_b_ping_slot_periodicity_t* ping_slot_periodicity
                                )
```

**Brief**

Get Class B Ping Slot Periodicity.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *ping_slot_periodicity* | Ping slot periodicity |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *ping_slot_periodicity* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.8 smtc_modem_class_b_set_ping_slot_periodicity()

```
smtc_modem_return_code_t smtc_modem_class_b_set_ping_slot_periodicity  (
                                uint8_t stack_id,
                                smtc_modem_class_b_ping_slot_periodicity_t ping_slot_periodicity
                                )
```

**Brief**

Set Class B Ping Slot Periodicity.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *ping_slot_periodicity* | Ping slot periodicity |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.9 smtc_modem_connection_timeout_get_current_values()

```
smtc_modem_return_code_t smtc_modem_connection_timeout_get_current_values (
                                    uint8_t     stack_id,
                                    uint16_t*   nb_of_uplinks_before_network_controlled,
                                    uint16_t*   nb_of_uplinks_before_reset
                                    )
```

**Brief**

Get the current status of the connection timeouts.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *nb_of_uplinks_before_network_controlled* | Number of remaining uplinks without downlink before the ADR profile switches to network-controlled |
| [out] | *nb_of_uplinks_before_reset* | Number of remaining uplinks without downlink before reset |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | Parameter *nb_of_uplinks_before_network_controlled* and/or *nb_of_uplinks_before_reset* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.10 smtc_modem_connection_timeout_get_thresholds()

```
smtc_modem_return_code_t smtc_modem_connection_timeout_get_thresholds  (
                                    uint8_t stack_id,
                                    uint16_t*   nb_of_uplinks_before_network_controlled,
                                    uint16_t*   nb_of_uplinks_before_reset
                                    )
```

**Brief**

Get the configured connection timeout thresholds.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *nb_of_uplinks_before_network_controlled* | Number of uplinks without downlinks before the ADR profile switches to network-controlled |
| [out] | *nb_of_uplinks_before_reset* | Number of uplinks without downlinks before reset |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | Parameter *nb_of_uplinks_before_network_controlled* and/or *nb_of_uplinks_before_reset* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.11 smtc_modem_connection_timeout_set_thresholds()

```
smtc_modem_return_code_t smtc_modem_connection_timeout_set_thresholds  (
                                    uint8_t stack_id,
                                    uint16_t  nb_of_uplinks_before_network_controlled,
                                    uint16_t  nb_of_uplinks_before_reset
                                    )
```

**Brief**

Set the connection timeout thresholds.

**Remarks**

The value 0 deactivates the command. It is recommended to have *nb_of_uplinks_before_network_controlled* smaller than *nb_of_uplink_before_reset*.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *nb_of_uplinks_before_network_controlled* | Number of uplinks without downlink before the ADR profile switches to network-controlled |
| [out] | *nb_of_uplinks_before_reset* | Number of uplinks without downlink before reset |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.12 smtc_modem_d2d_class_b_get_tx_max_payload()

```
smtc_modem_return_code_t smtc_modem_d2d_class_b_get_tx_max_payload  (
                                    uint8_t stack_id,
                                    smtc_modem_mc_grp_id_t mc_grp_id,
                                    uint8_t* tx_max_payload_size
                                    )
```

**Brief**

Get the maximum payload size that can be used for a device to device uplink on chosen multicast group.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *mc_grp_id* | The multicast group identifier |
| [out] | *tx_max_payload_size* | The maximum payload size in byte |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *tx_max_payload_size* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | Modem is not available (suspended, muted or not joined) or no multicast session is running on this group id |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.13 smtc_modem_d2d_class_b_request_uplink()

```
smtc_modem_return_code_t smtc_modem_d2d_class_b_request_uplink  (
                                    uint8_t stack_id,
                                    smtc_modem_mc_grp_id_t mc_grp_id,
                                    smtc_modem_d2d_class_b_uplink_config_t * d2d_config,
                                    uint8_t fport,
                                    const uint8_t* payload,
                                    uint8_t payload_length
                                    )
```

**Brief**

Request a device-to-device uplink.

**Remarks**

The uplink will be sent as soon as possible in the first available ping slot according to chosen *ping_slots_mask*. It will be repeated *nb_rep* times in following acceptable slots.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *mc_grp_id* | The multicast group identifier |
| [in] | *d2d_config* | The device to device specific uplink configuration structure |
| [in] | *fport* | The LoRaWAN FPort on which the uplink is done |
| [in] | *payload* | The data to be sent |
| [in] | *payload_length* | The number of bytes from payload to be sent |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *fport* is out of the [1:223] range |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | Modem is not available (suspended, muted or not joined) or no multicast session is running on this group id |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.14 smtc_modem_dm_get_fport()

```
smtc_modem_return_code_t smtc_modem_dm_get_fport  (
                                  uint8_t*  dm_fport
                                  )
```

**Brief**

Get the Device Management (DM) LoRaWAN® FPort.

**Parameters**

| | | |
|---|---|---|
| [out] | *dm_fport* | LoRaWAN® FPort on which the DM info is sent |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | Parameter *dm_fport* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.15 smtc_modem_dm_get_info_fields()

```
smtc_modem_return_code_t smtc_modem_dm_get_info_fields  (
                                  uint8_t*  dm_fields_payload,
                                  uint8_t*  dm_field_length
                                  )
```

**Brief**

Get the Device Management (DM) info fields.

**Parameters**

| | | |
|---|---|---|
| [in] | *dm_fields_payload* | DM info fields (see Table 7.2 - Device Management Fields) |
| [in] | *dm_field_length* | DM info field length |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | Parameter *dm_fields_payload* and/or *dm_field_length* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.16 smtc_modem_dm_get_info_interval()

```
smtc_modem_return_code_t smtc_modem_dm_get_info_interval (
                                  smtc_modem_dm_info_interval_format_t*  format,
                                  uint8_t*    interval
                                   )
```

**Brief**

Get the interval between two Device Management (DM) info field messages.

**Parameters**

| | | |
|---|---|---|
| [out] | *format* | Reporting interval format |
| [out] | *interval* | Interval in unit defined in format |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | Parameter *format* and/or *interval* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.17 smtc_modem_dm_get_user_data()

```
smtc_modem_return_code_t smtc_modem_dm_get_user_data  (
                                    uint8_t user_data[SMTC_MODEM_DM_USER_DATA_LENGTH]
                                     )
```

**Brief**

Get user-specific data to be reported by Device Management (DM) frames.

**Parameters**

| | | |
|---|---|---|
| [out] | *user_data* | User-specific data |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | Parameter *user_data* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.18 smtc_modem_dm_request_single_uplink()

```
smtc_modem_return_code_t smtc_modem_dm_request_single_uplink (
                                    const uint8_t*  dm_fields_payload,
                                    uint8_t  dm_field_length
                                     )
```

**Brief**

Request an immediate Device Management (DM) status.

**Remarks**

The content is independent from the configuration set with *smtc_modem_dm_set_info_fields()*.

**Parameters**

| | | |
|---|---|---|
| [in] | *dm_fields_payload* | DM info fields (see *DM info fields codes*) |
| [in] | *dm_field_length* | DM info field length |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | Invalid or duplicated field code or parameter *dm_fields_payload* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | Modem is not available (suspended, muted or not joined) |

### 7.5.19 smtc_modem_dm_set_fport()

```
smtc_modem_return_code_t smtc_modem_dm_set_fport  (
                                    uint8_t dm_fport
                                    )
```

**Brief**

Set the Device Management (DM) LoRaWAN FPort.

**Parameters**

| | | |
|---|---|---|
| [in] | *dm_fport* | LoRaWAN FPort on which the DM info is sent. This value must be in the range [1:223] |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | Parameter *dm_fport* is out of the [1:223] range |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.20 smtc_modem_dm_set_info_fields()

```
smtc_modem_return_code_t smtc_modem_dm_set_info_fields (
                                    const uint8_t*  dm_fields_payload,
                                    uint8_t dm_field_length
                                     )
```

**Brief**

Set the Device Management (DM) info fields to be sent on a regular basis.

**Remarks**

The interval between two DM info field messages is defined with *smtc_modem_dm_set_info_interval()*.

**Parameters**

| | | |
|---|---|---|
| [in] | *dm_fields_payload* | DM info fields (see *DM info fields codes*) |
| [in] | *dm_field_length* | DM info field length |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | Invalid or duplicated DM info fields |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.21 smtc_modem_dm_set_info_interval()

```
smtc_modem_return_code_t smtc_modem_dm_set_info_interval (
                                    smtc_modem_dm_info_interval_format_t format,
                                    uint8_t  interval
                                     )
```

**Brief**

Set the interval between two Device Management (DM) info field messages.

**Remarks**

An interval value set to 0 disables the feature - no matter the format.

**Parameters**

| | | |
|---|---|---|
| [in] | *format* | Reporting interval format |
| [in] | *interval* | Interval in unit defined in format, from 0 to 63 |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | Parameter *interval* is not in the [0:63] range. |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.22 smtc_modem_dm_set_user_data()

```
smtc_modem_return_code_t smtc_modem_dm_set_user_data (
                                    const uint8_t user_data[SMTC_MODEM_DM_USER_DATA_LENGTH]
                                     )
```

**Brief**

Set user-specific data to be reported by Device Management (DM) frames.

**Remarks**

This field will be sent only if it is selected in *smtc_modem_dm_set_info_fields()* or *smtc_modem_dm_request_single_uplink()*.

**Parameters**

| | | |
|---|---|---|
| [in] | *user_data* | User-specific data |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.23 smtc_modem_factory_reset()

```
smtc_modem_return_code_t smtc_modem_factory_reset (void)
```

**Brief**

Reset the modem to its original state.

**Remarks**

Resets all modem-related non-volatile settings to their default values, then resets the MCU. Only LoRaWAN DevNonce is kept.

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.24 smtc_modem_file_upload_init()

```
smtc_modem_return_code_t smtc_modem_file_upload_init  (
                                    uint8_t stack_id,
                                    uint8_t  index,
                                    smtc_modem_file_upload_cipher_mode_t  cipher_mode,
                                    const uint8_t*   file,
                                    uint16_t  file_length,
                                    uint32_t  average_delay_s
                                    )
```

**Brief**

Create and initialize a file upload session.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *index* | Index on which the upload is done |
| [in] | *cipher_mode* | Cipher mode |
| [in] | *file* | File buffer |
| [in] | *file_length* | File size in bytes |
| [in] | *average_delay_s* | The minimal delay between two file upload fragments in seconds (from the end of an uplink to the start of the next one) |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *file_length* is equal to 0 or greater than 8192 bytes, or *file* pointer is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode or a file upload is already ongoing |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.25 smtc_modem_file_upload_reset()

```
smtc_modem_return_code_t smtc_modem_file_upload_reset  (
                                    uint8_t stack_id
                                    )
```

**Brief**

Reset the file upload session.

**Remarks**

This function will stop any ongoing file upload session.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_NOT_INIT* | No file upload session currently running |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.26 smtc_modem_file_upload_start()

```
smtc_modem_return_code_t smtc_modem_file_upload_start  (
                                    uint8_t stack_id
                                    )
```

**Brief**

Start the file upload session.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode or a file upload is already ongoing |
| *SMTC_MODEM_RC_FAIL* | Modem is not available (suspended, muted or not joined) |
| *SMTC_MODEM_RC_BAD_SIZE* | Total data sent does not match the declared *file_length* value in *smtc_modem_file_upload_init()* |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.27 smtc_modem_get_adr_ack_limit_delay()

```
smtc_modem_return_code_t smtc_modem_get_adr_ack_limit_delay  (
                                    uint8_t stack_id
                                    uint8_t * adr_ack_limit,
                                    uint8_t * adr_ack_delay
                                    )
```

**Brief**

Get the configured LoRaWan stack ADR ACK limit and ADR ACK delay in regards to ADR fallback if no downlink are received.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *adr_ack_limit* | ADR ACK limit |
| [out] | *adr_ack_delay* | ADR ACK delay |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *adr_ack_limit* or *adr_ack_delay* are NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.28 smtc_modem_get_available_datarates()

```
smtc_modem_return_code_t smtc_modem_get_available_datarates  (
                                    uint8_t stack_id,
                                    uint16_t*   available_datarates_mask
                                    )
```

**Brief**

Get the current available Datarate in regards to Uplink ChMash and DwellTime.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *available_datarates_mask* | The available data rates, described in a bit field |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *available_datarates_mask* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.29 smtc_modem_get_certification_mode()

```
smtc_modem_return_code_t smtc_modem_get_certification_mode  (
                                    uint8_t stack_id,
                                    bool*  enable
                                    )
```

**Brief**

Get the current state of the certification mode.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *enable* | Certification mode state |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | Parameter *enable* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.30 smtc_modem_get_charge()

```
smtc_modem_return_code_t smtc_modem_get_charge  (
                                    uint32_t*  charge_mah
                                    )
```

**Brief**

Get the total charge counter of the modem in mAh.

**Parameters**

| | | |
|---|---|---|
| [out] | *charge_mah* | Accumulated charge in mAh |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | Parameter *charge_mah* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.31 smtc_modem_get_class()

```
smtc_modem_return_code_t smtc_modem_get_class  (
                                    uint8_t stack_id,
                                    smtc_modem_class_t*  lorawan_class
                                    )
```

**Brief**

Get the current LoRaWAN network class.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *lorawan_class* | Current LoRaWAN class |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.32 smtc_modem_get_crystal_error_ppm()

```
smtc_modem_return_code_t smtc_modem_get_crystal_error (
                                    uint32_t*  crystal_error_ppm
                                    )
```

**Brief**

Get the modem crystal error.

**Parameters**

| | | |
|---|---|---|
| [out] | *crystal_error_ppm* | Crystal error in ppm |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *crystal_error_ppm* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.33 smtc_modem_get_deveui()

```
smtc_modem_return_code_t smtc_modem_get_deveui (
                                    uint8_t stack_id,
                                    uint8_t deveui[SMTC_MODEM_EUI_LENGTH]
                                    )
```

**Brief**

Get the DevEUI.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *deveui* | Current DevEUI |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.34 smtc_modem_get_duty_cycle_status()

```
smtc_modem_return_code_t smtc_modem_get_duty_cycle_status (
                                    int32_t*   duty_cycle_status_ms
                                    )
```

**Brief**

Get the current status of the duty cycle.

**Remarks**

If the returned value is positive, it is the time still available. A negative value indicates the time to wait until band availability.

**Parameters**

| | | |
|---|---|---|
| [out] | *duty_cycle_status_ms* | Status of the duty cycle in milliseconds |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | Parameter *duty_cycle_status_ms* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.35 smtc_modem_get_event()

```
smtc_modem_return_code_t smtc_modem_get_event (
                                    smtc_modem_event_t*  event,
                                    uint8_t*   event_pending_count
                                    )
```

**Brief**

Get the modem event.

**Remarks**

This command can be used to retrieve pending events from the modem.

**Parameters**

| | | |
|---|---|---|
| [out] | *event* | Structure holding event-related information |
| [out] | *event_pending_count* | Number of pending event(s) |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *event* or *event_pending_count* are NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.36 smtc_modem_get_joineui()

```
smtc_modem_return_code_t smtc_modem_get_joineui  (
                                    uint8_t  stack_id,
                                    uint8_t  joineui[SMTC_MODEM_EUI_LENGTH]
                                    )
```

**Brief**

Get the JoinEUI.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *joineui* | Current JoinEUI |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.37 smtc_modem_get_lorawan_version()

```
smtc_modem_return_code_t smtc_modem_get_lorawan_version (
                                    smtc_modem_lorawan_version_t*  lorawan_version
                                    )
```

**Brief**

Get the LoRaWAN stack network.

**Parameters**

| | | |
|---|---|---|
| [out] | *lorawan_version* | The LoRaWAN version |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *lorawan_version* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.38 smtc_modem_get_modem_version()

```
smtc_modem_return_code_t smtc_modem_get_modem_version (
                                    smtc_modem_version_t*  firmware_version
                                    )
```

**Brief**

Get the modem firmware version.

**Parameters**

| | | |
|---|---|---|
| [out] | *firmware_version* | Firmware version |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *firmware_version* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.39 smtc_modem_get_nb_trans()

```
smtc_modem_return_code_t smtc_modem_get_nb_trans  (
                                    uint8_t stack_id,
                                    uint8_t*   nb_trans
                                    )
```

**Brief**

Get the configured number of transmissions in case of unconfirmed uplink.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *nb_trans* | Number of transmissions |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *nb_trans* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.40 smtc_modem_get_network_frame_pending_status()

```
smtc_modem_return_code_t smtc_modem_get_network_frame_pending_status  (
                                    uint8_t stack_id,
                                    smtc_modem_frame_pending_bit_status_t* frame_pending_bit_status
                                    )
```

**Brief**

Get network frame pending status.

**Remarks**

This bit is set by the network when data is available and a downlink opportunity is required.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *frame_pending_bit_status* | Frame pending bit status |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *frame_pending_bit_status* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.41 smtc_modem_get_network_type()

```
smtc_modem_return_code_t smtc_modem_get_network_type  (
                                    uint8_t stack_id,
                                    bool*   network_type
                                    )
```

**Brief**

Get the configured network type.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *network_type* | Current configuration (true: public network / false: private network) |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *network_type* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.42 smtc_modem_get_next_tx_max_payload()

```
smtc_modem_return_code_t smtc_modem_get_next_tx_max_payload  (
                                    uint8_t stack_id,
                                    uint8_t*   tx_max_payload_size
                                    )
```

**Brief**

Get the maximum payload size that can be used for the next uplink.

**Remarks**

This value depends on the LoRaWAN regional parameters for the next transmission using the current data rate.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *tx_max_payload_size* | Maximum payload size in byte |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | Parameter *tx_max_payload_size* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | Modem has not joined a network |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.43 smtc_modem_get_region()

```
smtc_modem_return_code_t smtc_modem_get_region  (
                                    uint8_t stack_id,
                                    smtc_modem_region_t*   region
                                    )
```

**Brief**

Get the current LoRaWAN region.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *region* | Current LoRaWAN region |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *region* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.44 smtc_modem_get_regional_params_version()

```
smtc_modem_return_code_t smtc_modem_get_regional_params_version (
                                    smtc_modem_lorawan_version_t*  regional_params_version
                                    )
```

**Brief**

Get the stack Regional Parameters version.

**Parameters**

| | | |
|---|---|---|
| [out] | *regional_params_version* | The stack regional parameters version |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *regional_params_version* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.45 smtc_modem_get_stack_state()

```
smtc_modem_return_code_t smtc_modem_get_stack_state  (
                                    uint8_t stack_id,
                                    smtc_modem_stack_state_t*   stack_state
                                    )
```

**Brief**

Get the current state of the stack.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *stack_state* | Stack current state |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *stack_state* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.46 smtc_modem_get_status()

```
smtc_modem_return_code_t smtc_modem_get_status  (
                                    uint8_t stack_id,
                                    smtc_modem_status_mask_t*   status_mask
                                    )
```

**Brief**

Get the modem status.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *status_mask* | Modem status (see *smtc_modem_status_mask_e*) |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *status_mask* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.47 smtc_modem_get_time()

```
smtc_modem_return_code_t smtc_modem_get_time  (
                                    uint32_t*   gps_time_s,
                                    uint32_t*   gps_fractional_s
                                    )
```

**Brief**

Get GPS epoch time - number of seconds elapsed since GPS epoch (00:00:00, Sunday 6th of January 1980).

**Parameters**

| | | |
|---|---|---|
| [out] | *gps_time_s* | GPS time in seconds |
| [out] | *gps_fractional_s* | GPS fractional seconds |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *gps_time_s* or *gps_fractional_s* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_NO_TIME* | No time available |

### 7.5.48 smtc_modem_get_tx_power_offset_db()

```
smtc_modem_return_code_t smtc_modem_get_tx_power_offset_db  (
                                    uint8_t stack_id,
                                    int8_t*   tx_pwr_offset_db
                                    )
```

**Brief**

Get the Tx power offset in dB.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *tx_pwr_offset_db* | Tx power offset in dB |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *tx_pwr_offset_db* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.49 smtc_modem_increment_event_middleware()

```
smtc_modem_return_code_t smtc_modem_increment_event_middleware       (
                                                uint8_t         event_type,
                                                uint8_t         status
                                                )
```

**Brief**

Increment a middleware asynchronous event.

**Parameters**

| | | |
|---|---|---|
| [in] | *event_type* | Type of asynchronous message |
| [in] | *status* | Status of asynchronous message |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *event_type* isn't a middleware event type |

### 7.5.50 smtc_modem_init()

```
void smtc_modem_init (
                                  const ralk_t* radio,
                                  void(*)(void) event_callback
                                  )
```

**Brief**

Init the soft modem and set the modem event chosen callback.

**Remarks**

The callback will be called each time a modem event is raised internally.

**Parameters**

| | | |
|---|---|---|
| [in] | event_callback | User event callback prototype |

### 7.5.51 smtc_modem_join_network()

```
smtc_modem_return_code_t smtc_modem_join_network  (
                                   uint8_t stack_id
                                   )
```

**Brief**

Join the network.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors or modem has already joined the network |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode or in joining/joined state |
| *SMTC_MODEM_RC_FAIL* | Modem is not available (suspended or muted) |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.52 smtc_modem_lbt_get_parameters()

```
smtc_modem_return_code_t smtc_modem_lbt_get_parameters  (
                                   uint8_t     stack_id,
                                   uint32_t*   listening_duration_ms,
                                   int16_t*    threshold_dbm,
                                   uint32_t*   bw_hz
                                   )
```

**Brief**

Get the parameters of the Listen Before Talk (LBT) feature.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *listening_duration_ms* | Current listening duration in ms |
| [out] | *threshold_dbm* | Current LBT threshold in dbm |
| [out] | *bw_hz* | Current LBT bandwidth in Hertz |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | At least one parameter is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.53 smtc_modem_lbt_get_state()

```
smtc_modem_return_code_t smtc_modem_lbt_get_state  (
                                    uint8_t  stack_id,
                                    bool*    enabled
                                    )
```

**Brief**

Get the state of the Listen Before Talk (LBT) feature.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *enabled* | Current status of the LBT feature (true: enabled, false: disabled) |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *enabled* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.54 smtc_modem_lbt_set_parameters()

```
smtc_modem_return_code_t smtc_modem_lbt_set_parameters  (
                                    uint8_t stack_id,
                                    uint32_t  listening_duration_ms,
                                    int16_t  threshold_dbm,
                                    uint32_t  bw_hz
                                    )
```

**Brief**

Set the parameters of the Listen Before Talk (LBT) feature.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *listening_duration_ms* | Listening duration in ms to be configured |
| [in] | *threshold_dbm* | LBT threshold in dbm to be configured |
| [in] | *bw_hz* | LBT bandwidth in Hertz to be configured |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.55 smtc_modem_lbt_set_state()

```
smtc_modem_return_code_t smtc_modem_lbt_set_state  (
                                uint8_t  stack_id,
                                bool  enable
                                )
```

**Brief**

Enable or disable the Listen Before Talk (LBT) feature.

**Remarks**

The configuration function *smtc_modem_lbt_set_parameters()* must be called before enabling the LBT feature. LBT is silently enabled if the feature is mandatory in a region selected with *smtc_modem_set_region()*.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *enable* | Status of the LBT feature to set (true: enable, false: disable) |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.56 smtc_modem_leave_network()

```
smtc_modem_return_code_t smtc_modem_leave_network  (
                                uint8_t stack_id
                                )
```

**Brief**

Leave an already joined network or cancels on ongoing join process.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |

**Returns**
Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.57 smtc_modem_lorawan_class_b_request_ping_slot_info()

```
smtc_modem_return_code_t smtc_modem_lorawan_class_b_request_ping_slot_info   (
                                                        uint8_t     stack_id
                                                        )
```

**Brief**
Request a Ping Slot Info MAC command to the network.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |

**Returns**
Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | Modem is not available (suspended, muted or not joined) |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.58 smtc_modem_lorawan_get_lost_connection_counter()

```
smtc_modem_return_code_t smtc_modem_lorawan_get_lost_connection_counter      (
                                                    uint8_t  stack_id,
                                                    uint16_t *       lost_connection_cnt
                                                    )
```

**Brief**
Get the current value of the lost connection counter.

**Remarks**
The counter is incremented after any uplink and is only reset when a valid downlink is received from Network Server.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *lost_connection_cnt* | Lost connection counter current value |

**Returns**
Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *lost_connection_cnt* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.59 smtc_modem_lorawan_request_link_check()

```
smtc_modem_return_code_t smtc_modem_lorawan_request_link_check  (
                                    uint8_t stack_id
                                    )
```

**Brief**

Request a Link Check Req MAC command to the network.

**Remarks**

The request will be sent in a new uplink frame.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | Modem is not available (suspended, muted or not joined) |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.60 smtc_modem_multicast_class_b_get_session_status()

```
smtc_modem_return_code_t smtc_modem_multicast_class_b_get_session_status (
                                                uint8_t   stack_id,
                                                smtc_modem_mc_grp_id_t    mc_grp_id,
                                                bool *    is_session_started,
                                                bool *    is_session_waiting_for_beacon,
                                                uint32_t *        freq,
                                                uint8_t *         dr,
                                                smtc_modem_class_b_ping_slot_periodicity_t *     ␣
↪ping_slot_periodicity
                                                )
```

**Brief**

Get class B multicast session status for a chosen group.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *mc_grp_id* | Multicast group identifier |
| [out] | *is_session_started* | Session status |
| [out] | *is_session_waiting_for_beacon* | Session beacon waiting status |
| [out] | *dr* | Session downlink datarate |
| [out] | *freq* | Session downlink frequency |
| [out] | *ping_slot_periodicity* | Session ping slot periodicity |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *mc_grp_id* is not in the range [0:3] or a parameter is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.61 smtc_modem_multicast_class_b_start_session()

```
smtc_modem_return_code_t smtc_modem_multicast_class_b_start_session(
                         uint8_t      stack_id,
                         smtc_modem_mc_grp_id_t       mc_grp_id,
                         uint32_t freq,
                         uint8_t dr,
                         smtc_modem_class_b_ping_slot_periodicity_t   ping_slot_periodicity
                         )
```

**Brief**

Start class B multicast session for a specific group.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *mc_grp_id* | Multicast group identifier |
| [in] | *freq* | Downlink frequency for this session |
| [in] | *dr* | Downlink datarate for this session |
| [in] | *ping_slot_periodicity* | Ping slot periodicity for this session |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *mc_grp_id* is not in the range [0:3] *freq* or *dr* are not in acceptable range (according to current regional params) |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | This session is already started or modem is not in class B |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.62 smtc_modem_multicast_class_b_stop_all_sessions()

```
smtc_modem_return_code_t smtc_modem_multicast_class_b_stop_all_sessions(
                                                    uint8_t stack_id
                                                    )
```

**Brief**

Stop all started class B multicast sessions.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.63 smtc_modem_multicast_class_b_stop_session()

```
smtc_modem_return_code_t smtc_modem_multicast_class_b_stop_session(
                                    uint8_t stack_id,
                                    smtc_modem_mc_grp_id_t        mc_grp_id
                                    )
```

**Brief**

Stop class B multicast session for a chosen group.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *mc_grp_id* | Multicast group identifier |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *mc_grp_id* is not in the range [0:3] |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.64 smtc_modem_multicast_class_c_get_session_status()

```
smtc_modem_return_code_t smtc_modem_multicast_class_c_get_session_status(
                                            uint8_t stack_id,
                                            smtc_modem_mc_grp_id_t    mc_grp_id,
                                            bool *    is_session_started,
                                            uint32_t *        freq,
                                            uint8_t *         dr
                                            )
```

**Brief**

Get class C multicast session status for a chosen group.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *mc_grp_id* | Multicast group identifier |
| [out] | *is_session_started* | Session status |
| [out] | *freq* | Downlink frequency in Hz for this session |
| [out] | *dr* | Downlink datarate for this session |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *mc_grp_id* is not in the range [0:3] or a parameter is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.65 smtc_modem_multicast_class_c_start_session()

```
smtc_modem_return_code_t smtc_modem_multicast_class_c_start_session(
                                                uint8_t stack_id,
                                                smtc_modem_mc_grp_id_t    mc_grp_id,
                                                uint32_t *        freq,
                                                uint8_t *         dr
                                                )
```

**Brief**

Start class C multicast session for a specific group.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *mc_grp_id* | Multicast group identifier |
| [in] | *freq* | Downlink frequency in Hz for this session |
| [in] | *dr* | Downlink datarate for this session |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| SMTC_MODEM_RC_OK | Command executed without errors |
| SMTC_MODEM_RC_INVALID | *mc_grp_id* is not in the range [0:3]. *freq* or *dr* are not in acceptable range (according to current regional params). *freq* or *dr* are not compatible with an already running multicast session. |
| SMTC_MODEM_RC_BUSY | Modem is currently in test mode |
| SMTC_MODEM_RC_FAIL | This session is already started or modem is not in class C |
| SMTC_MODEM_RC_INVALID_STACK_ID | Invalid *stack_id* |

### 7.5.66 smtc_modem_multicast_class_c_stop_all_sessions()

```
smtc_modem_return_code_t smtc_modem_multicast_class_c_stop_all_sessions(
                                                    uint8_t stack_id
                                                    )
```

**Brief**

Stop all started class C multicast sessions.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.67 smtc_modem_multicast_class_c_stop_session()

```
smtc_modem_return_code_t smtc_modem_multicast_class_c_stop_session(
                                                    uint8_t stack_id,
                                                    smtc_modem_mc_grp_id_t          mc_grp_id
                                                    )
```

**Brief**

Stop class C multicast session for a chosen group.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *mc_grp_id* | Multicast group identifier |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *mc_grp_id* is not in the range [0:3] or a parameter is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.68 smtc_modem_multicast_get_grp_config()

```
smtc_modem_return_code_t smtc_modem_multicast_get_grp_config  (
                                    uint8_t stack_id,
                                    smtc_modem_mc_grp_id_t  mc_grp_id,
                                    uint32_t*   mc_grp_addr
                                    )
```

**Brief**

Get the configuration of the chosen multicast group.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *mc_grp_id* | Multicast group identifier |
| [out] | *mc_grp_addr* | Multicast group address |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *mc_grp_id* is not in the range [0:3] or parameter *mc_grp_addr* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.69 smtc_modem_multicast_set_grp_config()

```
smtc_modem_return_code_t smtc_modem_multicast_set_grp_config  (
                                    uint8_t  stack_id,
                                    smtc_modem_mc_grp_id_t  mc_grp_id,
                                    uint32_t  mc_grp_addr,
                                    const uint8_t  mc_nwk_skey[SMTC_MODEM_KEY_LENGTH],
                                    const uint8_t  mc_app_skey[SMTC_MODEM_KEY_LENGTH]
                                    )
```

**Brief**

Configure a multicast group.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *mc_grp_id* | Multicast group identifier |
| [in] | *mc_grp_addr* | Multicast group address |
| [in] | *mc_nwk_skey* | Multicast network session key for the group |
| [in] | *mc_app_skey* | Multicast application session key for the group |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *mc_grp_id* is not in the range [0:3] |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | Error during crypto process or a running session already exists on this id |

### 7.5.70 smtc_modem_request_emergency_uplink()

```
smtc_modem_return_code_t smtc_modem_request_emergency_uplink  (
                                    uint8_t stack_id,
                                    uint8_t  fport,
                                    bool  confirmed,
                                    const uint8_t*  payload,
                                    uint8_t  payload_length
                                    )
```

**Brief**

Request an immediate LoRaWAN uplink.

**Remarks**

It has higher priority than all other services and is not subject to duty cycle restrictions, if any LoRaWAN *NbTrans* parameter can be set in mobiles and custom ADR modes with *smtc_modem_set_nb_trans()*

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *fport* | LoRaWAN FPort on which the uplink is done |
| [in] | *confirmed* | Message type (true: confirmed, false: unconfirmed) |
| [in] | *payload* | Data to be sent |
| [in] | *payload_length* | Number of bytes from payload to be sent |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *fport* is out of the [1:223] range or equal to the DM LoRaWAN *FPort* |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | Modem is not available (suspended, muted or not joined) |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.71 smtc_modem_request_empty_uplink()

```
smtc_modem_return_code_t smtc_modem_request_empty_uplink (
                                    uint8_t  stack_id,
                                    bool  send_fport,
                                    uint8_t  fport,
                                    bool  confirmed
                                    )
```

**Brief**

Request a LoRaWAN uplink without payload, and an optional FPort.

**Remarks**

It can be used to create downlink opportunities / heartbeat without routing messages to an application server

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *send_fport* | Add the FPort to the payload (true: add the FPort, false: send without FPort) |
| [in] | *fport* | The LoRaWAN FPort on which the uplink is done, if used |
| [in] | *confirmed* | Message type (true: confirmed, false: unconfirmed) |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *fport* is out of the [1:223] range or equal to the DM LoRaWAN FPort |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | Modem is not available (suspended, muted or not joined) |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.72 smtc_modem_request_extended_uplink()

```
smtc_modem_return_code_t smtc_modem_request_extended_uplink(
                                                uint8_t stack_id,
                                                uint8_t   f_port,
                                                bool      confirmed,
                                                const uint8_t *   payload,
                                                uint8_t   payload_length,
                                                uint8_t   extended_uplink_id,
                                                void(*)(void)     lbm_notification_callback
                                                )
```

**Brief**

Request a LoRaWAN extended uplink.

**Remarks**

This feature is introduced for future libraries and functions, it is NOT recommended for the user to call this function. This feature requires a special compilation option to be activated.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *fport* | LoRaWAN FPort on which the uplink is done |
| [in] | *confirmed* | Message type (true: confirmed, false: unconfirmed) |
| [in] | *payload* | Data to be sent |
| [in] | *payload_length* | Number of bytes from payload to be sent |
| [in] | *extended_uplink_id* | ID of the queue for extended uplink should be equal to 1 or 2 |
| [in] | *lbm_notification_callback* | Notification callback (to notify middleware when tx is finished) |

**Returns**

Modem return code as defined in smtc_modem_return_code_t

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | Parameter *fport* is out of the [1:223] range or equal to *dm_fport*, or *extended_uplink_id* not equal to 1 or 2 |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | Modem is not available (suspended, muted or not joined) |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.73 smtc_modem_request_uplink()

```
smtc_modem_return_code_t smtc_modem_request_uplink  (
                                    uint8_t  stack_id,
                                    uint8_t  fport,
                                    bool  confirmed,
                                    const uint8_t*   payload,
                                    uint8_t  payload_length
                                    )
```

**Brief**

Request a LoRaWAN uplink.

**Remarks**

LoRaWAN *NbTrans* parameter can be set in mobiles and custom ADR modes with *smtc_modem_set_nb_trans()*.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *fport* | The LoRaWAN FPort on which the uplink is done |
| [in] | *confirmed* | Message type (true: confirmed, false: unconfirmed) |
| [in] | *payload* | Data to be sent |
| [in] | *payload_length* | Number of bytes from payload to be sent |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *fport* is out of the [1:223] range or equal to the DM LoRaWAN FPort |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | Modem is not available (suspended, muted or not joined) |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.74 smtc_modem_reset()

```
smtc_modem_return_code_t smtc_modem_reset (void)
```

**Brief**

Reset the modem.

**Remarks**

Resets modem transient state (including session information) by resetting the MCU. Device Management Port, Modem Region and LoRaWAN Devnonce are kept.

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.75 smtc_modem_reset_charge()

```
smtc_modem_return_code_t smtc_modem_reset_charge (void)
```

**Brief**

Reset the total charge counter of the modem.

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.76 smtc_modem_resume_after_user_radio_access()

```
smtc_modem_return_code_t smtc_modem_resume_after_user_radio_access (void)
```

**Brief**

Release user radio access and resume modem features (scheduler and radio access).

**Remarks**

The user must call this function after performing operations requiring a direct access to the radio (e.g. test modes). Otherwise, all modem-related tasks remain pending.

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.77 smtc_modem_rp_abort_user_radio_access_task()

```
smtc_modem_return_code_t smtc_modem_rp_abort_user_radio_access_task( uint8_t user_task_id )
```

**Brief**

Abort a user task in radio planner.

**Parameters**

| | | |
|---|---|---|
| [in] | *user_task_id* | ID of the user task to abort |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.78 smtc_modem_rp_add_user_radio_access_task()

```
smtc_modem_return_code_t smtc_modem_rp_add_user_radio_access_task    (
                                    smtc_modem_rp_task_t *        rp_task
                                    )
```

**Brief**

Add a user task in radio planner.

**Parameters**

| | | |
|---|---|---|
| [in] | *rp_task* | Structure holding radio planner task information |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | A user task is already running in radio planner |

### 7.5.79 smtc_modem_run_engine()

```
uint32_t smtc_modem_run_engine (void)
```

**Brief**

Run the modem engine.

**Remarks**

This function must be called in main loop.

**Returns**

The time in ms after which the function must at least be called again.

### 7.5.80 smtc_modem_set_adr_ack_limit_delay()

```
smtc_modem_return_code_t smtc_modem_set_adr_ack_limit_delay(
                                                uint8_t stack_id,
                                                uint8_t adr_ack_limit,
                                                uint8_t adr_ack_delay
                                                )
```

**Brief**

Set the LoRaWAN stack ADR ACK limit and ADR ACK delay regarding the ADR fallback if no downlink is received.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *adr_ack_limit* | ADR ACK limit. Accepted value: ( *adr_ack_limit* > 1 ) && ( *adr_ack_limit* < 128 ) |
| [in] | *adr_ack_delay* | ADR ACK delay. Accepted value: ( *adr_ack_delay* > 1 ) && ( *adr_ack_delay* < 128 ) |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *adr_ack_limit and adr_ack_delay* are not in the range [2:127] |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.81 smtc_modem_set_certification_mode()

```
smtc_modem_return_code_t smtc_modem_set_certification_mode  (
                                uint8_t  stack_id,
                                bool  enable
                                )
```

**Brief**

Enable / disable the certification mode.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *enable* | Certification mode state (default: disabled) |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.82 smtc_modem_set_class()

```
smtc_modem_return_code_t smtc_modem_set_class (
                                    uint8_t  stack_id,
                                    smtc_modem_class_t  lorawan_class
                                    )
```

**Brief**

Set the class used by the LoRaWAN network.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *lorawan_class* | The LoRaWAN class to be configured |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | LoRaWAN class is not in an acceptable range |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | For Class B only: no time is available or modem is not joined |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid "stack_id" |

### 7.5.83 smtc_modem_set_crystal_error_ppm()

```
smtc_modem_return_code_t smtc_modem_set_crystal_error_ppm  (
                                    uint32_t  crystal_error_ppm
                                    )
```

**Brief**

Set modem crystal error.

**Parameters**

| | | |
|---|---|---|
| [in] | *crystal_error_ppm* | Crystal error in ppm |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.84 smtc_modem_set_deveui()

```
smtc_modem_return_code_t smtc_modem_set_deveui  (
                                    uint8_t  stack_id,
                                    const uint8_t  deveui[SMTC_MODEM_EUI_LENGTH]
                                    )
```

**Brief**

Set the DevEUI.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *deveui* | DevEUI to be configured |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode or in joining/joined state |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.85 smtc_modem_set_joineui()

```
smtc_modem_return_code_t smtc_modem_set_joineui  (
                                    uint8_t  stack_id,
                                    const uint8_t  joineui[SMTC_MODEM_EUI_LENGTH]
                                    )
```

**Brief**

Set the JoinEUI.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *joineui* | JoinEUI to be configured |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode or in joining/joined state |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.86 smtc_modem_set_nb_trans()

```
smtc_modem_return_code_t smtc_modem_set_nb_trans  (
                                    uint8_t  stack_id,
                                    uint8_t  nb_trans
                                    )
```

**Brief**

Set the number of transmissions allowed in case of unconfirmed uplink.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *nb_trans* | Number of transmissions ( 0 < value < 16 ) |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *nb_trans* is not in [1:15] range or ADR profile is "Network controlled" |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.87 smtc_modem_set_network_type()

```
smtc_modem_return_code_t smtc_modem_set_network_type  (
                                    uint8_t  stack_id,
                                    bool  network_type
                                    )
```

**Brief**

Configure LoRaWAN network type to private or public.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *network_type* | Configuration to be applied (true: public network / false: private network) Default: public |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.88 smtc_modem_set_nwkkey()

```
smtc_modem_return_code_t smtc_modem_set_nwkkey  (
                                    uint8_t  stack_id,
                                    const uint8_t  nwkkey[SMTC_MODEM_KEY_LENGTH]
                                    )
```

**Brief**

Set the LoRaWAN v1.1.x Network Key (aka Application Key in LoRaWAN v1.0.x).

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *nwkkey* | Key to be configured |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode or in joining/joined state |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.89 smtc_modem_set_region()

```
smtc_modem_return_code_t smtc_modem_set_region  (
                                    uint8_t  stack_id,
                                    smtc_modem_region_t  region
                                    )
```

**Brief**

Set the LoRaWAN region.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *region* | LoRaWAN region to be configured |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *region* is not supported |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode or in joining/joined state |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.90 smtc_modem_set_tx_power_offset_db()

```
smtc_modem_return_code_t smtc_modem_set_tx_power_offset_db  (
                                    uint8_t  stack_id,
                                    int8_t  tx_pwr_offset_db
                                    )
```

**Brief**

Set the Tx power offset in dB.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *tx_pwr_offset_db* | Tx power offset in dB to be configured |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *tx_pwr_offset_db* is out of [-30:30] range |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.91 smtc_modem_stream_add_data()

```
smtc_modem_return_code_t smtc_modem_stream_add_data  (
                                    uint8_t  stack_id,
                                    const uint8_t*   data,
                                    uint8_t  len
                                    )
```

**Brief**

Add data to the stream.

**Remarks**

If *smtc_modem_stream_init* is not called beforehand, the stream uses the DM FPort with a redundancy ratio set to 110%.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *data* | Data to be added to the stream |
| [in] | *len* | Number of bytes from data to be added to the stream |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *len* is not in range [1-254] or *data* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode or the streaming buffer is full |
| *SMTC_MODEM_RC_FAIL* | Modem is not available (suspended, muted or not joined) |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.92 smtc_modem_stream_init()

```
smtc_modem_return_code_t smtc_modem_stream_init  (
                                    uint8_t  stack_id,
                                    uint8_t  fport,
                                    smtc_modem_stream_cipher_mode_t  cipher_mode,
                                    uint8_t  redundancy_ratio_percent
                                    )
```

**Brief**

Create and initialize a data stream.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *fport* | LoRaWAN FPort on which the stream is sent (0 forces the DM LoRaWAN FPort) |
| [in] | *redundancy_ratio_percent* | The stream redundancy ratio. |
| [in] | *cipher_mode* | The cipher mode |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *fport* is out of the [0:223] range |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode or the streaming buffer is not empty |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.93 smtc_modem_stream_status()

```
smtc_modem_return_code_t smtc_modem_stream_status  (
                                    uint8_t  stack_id,
                                    uint16_t*   pending,
                                    uint16_t*   free
                                    )
```

**Brief**

Return the current stream status.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *pending* | Length of pending data for transmission |
| [out] | *free* | Length of free space in the buffer |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_NOT_INIT* | No stream session is running |
| *SMTC_MODEM_RC_INVALID* | *pending* or *free* are NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.94 smtc_modem_suspend_before_user_radio_access()

```
smtc_modem_return_code_t smtc_modem_suspend_before_user_radio_access (void)
```

**Brief**

Grant user radio access by suspending the modem and killing all current modem radio tasks.

**Remarks**

The user must call this command before performing operations requiring a direct access to the radio (e.g. test modes). Otherwise, undefined behavior may occur.

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.95 smtc_modem_suspend_radio_communications()

```
smtc_modem_return_code_t smtc_modem_suspend_radio_communications (
                                                    bool  suspend
                                                    )
```

**Brief**

Suspend the radio communications initiated by the modem.

**Parameters**

| | | |
|---|---|---|
| [in] | *suspend* | The configuration to be applied (true: suspend communications / false: resume communications) |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.96 smtc_modem_time_get_alcsync_fport()

```
smtc_modem_return_code_t smtc_modem_time_get_alcsync_fport  (
                                    uint8_t*   alcsync_fport
                                    )
```

**Brief**

Get ALCSync service LoRaWAN FPort.

**Parameters**

| | | |
|---|---|---|
| [out] | *alcsync_fport* | FPort for ALCsync messages |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *alcsync_fport* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.97 smtc_modem_time_get_sync_interval_s()

```
smtc_modem_return_code_t smtc_modem_time_get_sync_interval_s  (
                                          uint32_t*   sync_interval_s
                                          )
```

**Brief**

Get the interval between time synchronizations.

**Parameters**

| | | |
|---|---|---|
| [out] | *sync_interval_s* | Interval in seconds |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *sync_interval_s* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.98 smtc_modem_time_get_sync_invalid_delay_s()

```
smtc_modem_return_code_t smtc_modem_time_get_sync_invalid_delay_s  (
                                            uint32_t*   sync_invalid_delay_s
                                            )
```

**Brief**

Get the configured delay beyond which the time synchronization is no longer valid.

**Parameters**

| | | |
|---|---|---|
| [out] | *sync_invalid_delay_s* | Invalid delay in seconds |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *sync_invalid_delay_s* is NULL |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.99 smtc_modem_time_set_alcsync_fport()

```
smtc_modem_return_code_t smtc_modem_time_set_alcsync_fport  (
                                                uint8_t  alcsync_fport
                                                )
```

**Brief**

Set ALCSync service LoRaWAN port.

**Remarks**

When using Device Management (DM) port for *alcsync_fport*, ALCSync messages are encapsulated into DM frames.

**Parameters**

| | | |
|---|---|---|
| [in] | *alcsync_fport* | LoRaWAN FPort for ALCSync messages |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *alcsync_fport* is invalid: out of [0:223] range |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.100 smtc_modem_time_set_sync_interval_s()

```
smtc_modem_return_code_t smtc_modem_time_set_sync_interval_s (
                                                uint32_t  sync_interval_s
                                                )
```

**Brief**

Set the interval between two time synchronization messages.

**Remarks**

*sync_interval_s* has to be lower than the value set with *smtc_modem_time_set_sync_invalid_delay_s()*. The default value is set to 36 hours (129600 seconds).

**Parameters**

| | | |
|---|---|---|
| [in] | *sync_interval_s* | Interval in seconds |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *sync_interval_s* is invalid |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.101 smtc_modem_time_set_sync_invalid_delay_s()

```
smtc_modem_return_code_t smtc_modem_time_set_sync_invalid_delay_s  (
                                            uint32_t  sync_invalid_delay_s
                                            )
```

**Brief**

Set the delay beyond which the time synchronization is no longer considered valid by the modem.

**Remarks**

*sync_invalid_delay_s* has to be higher than the value set with *smtc_modem_time_set_sync_interval_s()*. The default value is set to 49 days (4233600 seconds). Modem will generate a *SMTC_MODEM_EVENT_TIME* event if there are no time synchronizations for more time than this "invalid delay".

**Parameters**

| | | |
|---|---|---|
| [in] | *sync_invalid_delay_s* | Invalid delay in seconds |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_INVALID* | *sync_invalid_delay_s* is > 49 days |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |

### 7.5.102 smtc_modem_time_start_sync_service()

```
smtc_modem_return_code_t smtc_modem_time_start_sync_service  (
                                    uint8_t                         stack_id,
                                    smtc_modem_time_sync_service_t  sync_service
                                    )
```

**Brief**

Start a chosen time synchronization service.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [in] | *sync_service* | Time synchronization service to use |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | A time synchronization service is already running |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.103 smtc_modem_time_stop_sync_service()

```
smtc_modem_return_code_t smtc_modem_time_stop_sync_service (
                                          uint8_t stack_id
                                          )
```

**Brief**

Stop current time synchronization service.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | No time synchronization service is running |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.5.104 smtc_modem_time_trigger_sync_request()

```
smtc_modem_return_code_t smtc_modem_time_trigger_sync_request  (
                                            uint8_t stack_id
                                            )
```

**Brief**

Trigger a single uplink requesting time using current enabled time synchronization service.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_FAIL* | Modem is not available (suspended, muted or not joined) or no time synchronization service is running |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

## 7.6 LoRa Basics™ Modem API LR11xx Extension

The following functions constitute the LBM API LR11xx Extension.

### 7.6.1 smtc_modem_derive_keys()

```
smtc_modem_return_code_t smtc_modem_derive_keys()  (
                                        uint8_t stack_id
                                        )
```

**Brief**

Derive keys.

**Remarks**

This command can only be used with LR11xx radio. Derives the application key using the stored *dev_eui* (default set to *chip_eui*) and stored *join_eui*.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.6.2 smtc_modem_get_chip_eui()

```
smtc_modem_return_code_t smtc_modem_get_chip_eui  (
                                    uint8_t stack_id,
                                    uint8_t chip_eui[SMTC_MODEM_EUI_LENGTH]
                                    )
```

**Brief**

Get the modem chip EUI.

**Remarks**

This command can only be used with LR11xx radio.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *chip_eui* | 8-byte chip EUI |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

### 7.6.3 smtc_modem_get_pin()

```
smtc_modem_return_code_t smtc_modem_get_pin() (
                                    uint8_t  stack_id,
                                    uint8_t  chip_pin[SMTC_MODEM_PIN_LENGTH]
                                    )
```

**Brief**

Get the modem PIN code.

**Remarks**

This command can only be used on a Basic Modem with LR11xx radio.

**Parameters**

| | | |
|---|---|---|
| [in] | *stack_id* | Stack identifier |
| [out] | *chip_pin* | 4-byte PIN code |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

**Return values**

| | |
|---|---|
| *SMTC_MODEM_RC_OK* | Command executed without errors |
| *SMTC_MODEM_RC_BUSY* | Modem is currently in test mode |
| *SMTC_MODEM_RC_INVALID_STACK_ID* | Invalid *stack_id* |

## 7.7 LoRa Basics™ Modem API Test Functions

The following functions are useful when developing tests for regulatory conformance, certification, and functional testing.

### 7.7.1 smtc_modem_test_direct_radio_read()

```
smtc_modem_return_code_t smtc_modem_test_direct_radio_read (
                                            uint8_t *    command,
                                            uint16_t     command_length,
                                            uint8_t *    data,
                                            uint16_t     data_length
                                            )
```

**Brief**

Direct access to radio command read.

**Parameters**

| | | |
|---|---|---|
| [in] | *command* | Pointer to the buffer to be transmitted |
| [in] | *command_length* | Buffer size to be transmitted |
| [out] | *data* | Pointer to the buffer to be received |
| [in] | *data_length* | Buffer size to be received |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

### 7.7.2 smtc_modem_test_direct_radio_write()

```
smtc_modem_return_code_t smtc_modem_test_direct_radio_write (
                                            uint8_t *    command,
                                            uint16_t     command_length,
                                            uint8_t *    data,
                                            uint16_t     data_length
                                            )
```

**Brief**

Direct access to radio command write.

**Parameters**

| | | |
|---|---|---|
| [in] | *command* | Pointer to the buffer to be transmitted |
| [in] | *command_length* | Buffer size to be transmitted |
| [in] | *data* | Pointer to the buffer to be received |
| [in] | *data_length* | Buffer size to be received |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

### 7.7.3 smtc_modem_test_duty_cycle_app_activate()

```
smtc_modem_return_code_t smtc_modem_test_duty_cycle_app_activate (
                                                    bool status
                                                    )
```

**Brief**

Enable / disable the applicative dutycycle.

This function can be called regardless of test mode state.

**Parameters**

| | | |
|---|---|---|
| [in] | *status* | 0: disable, 1: enable |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

### 7.7.4 smtc_modem_test_get_nb_rx_packets()

```
smtc_modem_return_code_t smtc_modem_test_get_nb_rx_packets   ( uint32_t * nb_rx_packets )
```

**Brief**

Read number of received packets during test RX continue.

**Parameters**

| | | |
|---|---|---|
| [out] | *nb_rx_packets* | Number of received packets in RX Continue |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

### 7.7.5 smtc_modem_test_get_rssi()

```
smtc_modem_return_code_t smtc_modem_test_get_rssi (
                                        int8_t * rssi
                                        )
```

**Brief**

Get RSSI result (to be called when test rssi is finished).

**Remarks**

Returns the computed RSSI.

**Parameters**

| | | |
|---|---|---|
| [out] | *rssi* | rssi + 64 |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

### 7.7.6 smtc_modem_test_nop()

```
smtc_modem_return_code_t smtc_modem_test_nop ( void )
```

**Brief**

Perform no operation. This function can be used to terminate an ongoing continuous operation.

**Remarks**

Abort the radio planner task.

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

### 7.7.7 smtc_modem_test_radio_reset()

```
smtc_modem_return_code_t smtc_modem_test_radio_reset ( void )
```

**Brief**

Reset the Radio for test purpose.

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

### 7.7.8 smtc_modem_test_rssi()

```
smtc_modem_return_code_t smtc_modem_test_rssi (
                  uint32_t                frequency_hz,
                  smtc_modem_test_bw_t    bw,
                  uint16_t                time_ms
                  )
```

**Brief**

Test mode RSSI.

**Remarks**

Measure continuously the RSSI during a chosen time and give an average value.

**Parameters**

| | | |
|---|---|---|
| [in] | *frequency_hz* | Frequency in Hz |
| [in] | *bw* | Bandwidth following *smtc_modem_test_bw_t* definition |
| [in] | *time_ms* | Test duration in ms (1 rssi every 10 ms) |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

### 7.7.9 smtc_modem_test_rx_continuous()

```
smtc_modem_return_code_t smtc_modem_test_rx_continuous (
                         uint32_t               frequency_hz,
                         smtc_modem_test_sf_t   sf,
                         smtc_modem_test_bw_t   bw,
                         smtc_modem_test_cr_t   cr
                         )
```

**Brief**

Put modem in Test RX continuous mode.

**Remarks**

Continuously receive packets.

**Parameters**

| | | |
|---|---|---|
| [in] | *frequency_hz* | Frequency in Hz |
| [in] | *sf* | Spreading factor following *smtc_modem_test_sf_t* definition |
| [in] | *bw* | Bandwidth following *smtc_modem_test_bw_t* definition |
| [in] | *cr* | Coding rate following *smtc_modem_test_cr_t* definition |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

### 7.7.10 smtc_modem_test_start()

```
smtc_modem_return_code_t smtc_modem_test_start ( void )
```

**Brief**

Put modem in test mode.

**Remarks**

No other modem commands can be handled during modem test mode

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

### 7.7.11 smtc_modem_test_stop()

```
smtc_modem_return_code_t smtc_modem_test_stop ( void )
```

**Brief**

Exit modem test mode.

**Remarks**

Exit test mode and perform a reset of modem.

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

### 7.7.12 smtc_modem_test_tx()

```
smtc_modem_return_code_t smtc_modem_test_tx (
                                    uint8_t *               payload,
                                    uint8_t                 payload_length,
                                    uint32_t                frequency_hz,
                                    int8_t                  tx_power_dbm,
                                    smtc_modem_test_sf_t    sf,
                                    smtc_modem_test_bw_t    bw,
                                    smtc_modem_test_cr_t    cr,
                                    uint32_t                preamble_size,
                                    bool                    continuous_tx
                                    )
```

**Brief**

Test mode TX single or continuous.

**Remarks**

Transmit a single packet or continuously transmit packets as fast as possible.

**Parameters**

| | | |
|---|---|---|
| [in] | *payload* | Payload that will be sent. If NULL a randomly generated payload_length msg will be sent |
| [in] | *payload_length* | Length of the payload |
| [in] | *frequency_hz* | Frequency in Hz |
| [in] | *tx_power_dbm* | Power in dBm |
| [in] | *sf* | Spreading factor following *smtc_modem_test_sf_t* definition |
| [in] | *bw* | Bandwidth following *smtc_modem_test_bw_t* definition |
| [in] | *cr* | Coding rate following *smtc_modem_test_cr_t* definition |
| [in] | *preamble_size* | Size of the preamble |
| [in] | *continuous_tx* | false: single transmission / true: continuous transmission |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

### 7.7.13 smtc_modem_test_tx_cw()

```
smtc_modem_return_code_t smtc_modem_test_tx_cw (
                                        uint32_t frequency_hz,
                                        int8_t   tx_power_dbm
                                        )
```

**Brief**

Test mode transmit a continuous wave.

**Parameters**

| | | |
|---|---|---|
| [in] | *frequency_hz* | Frequency in Hz |
| [in] | *tx_power_dbm* | Power in dbm |

**Returns**

Modem return code as defined in *smtc_modem_return_code_t*.

# 8. Revision History

| User Manual Version | ECO | Date | Applicable to | Changes |
|---|---|---|---|---|
| 1.0 | 059175 | Oct-2021 | LBM Version: 2.1.0 | First Release |
| 2.0 | 060217 | Jan-2022 | LBM Version: 2.1.0 | Extracted the Porting Guide into a new document |
| 3.0 | 061836 | May-2022 | LBM Version: 3.1.7 | Updated for LoRa Basics™ Modem v3.1.7 |
| 3.2 | 063508 | Sep-2022 | LBM Version: 3.2.4 | Updated for LoRa Basics™ Modem v3.2.4 |
| 3.3 | 064883 | Jan-2023 | LBM Version: 3.2.4 | Added LR1121 support, Added known limitations |
| 3.4 | 067305 | Jun-2023 | LBM Version: 3.3.0 | Updated for LoRa Basics™ Modem v3.3.0 |

**Important Notice**

Information relating to this product and the application or design described herein is believed to be reliable, however such information is provided as a guide only and Semtech assumes no liability for any errors in this document, or for the application or design described herein. Semtech reserves the right to make changes to the product or this document at any time without notice. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. Semtech warrants performance of its products to the specifications applicable at the time of sale, and all sales are made in accordance with Semtech's standard terms and conditions of sale.

SEMTECH PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS, OR IN NUCLEAR APPLICATIONS IN WHICH THE FAILURE COULD BE REASONABLY EXPECTED TO RESULT IN PERSONAL INJURY, LOSS OF LIFE OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. INCLUSION OF SEMTECH PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE UNDERTAKEN SOLELY AT THE CUSTOMER'S OWN RISK. Should a customer purchase or use Semtech products for any such unauthorized application, the customer shall indemnify and hold Semtech and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs damages and attorney fees which could arise.



## Contact Information

Semtech Corporation
Wireless & Sensing Products
200 Flynn Road, Camarillo, CA 93012
Phone: (805) 498-2111, Fax: (805) 498-3804