ProLink LoRaWAN EndNode Modem HCI Specification

Version 2.3

Document ID: 4000/40140/0173

IMST GmbH
Carl-Friedrich-Gauß-Str. 2-4
47475 KAMP-LINTFORT
GERMANY

Document Information

File name	ProLink_LoRaWAN_EndNode_Modem_HCI_Spec.docx
Created	2019-01-21
Total pages	149

Revision History

Version	Note
0.1	Created, initial version
0.2	Reviewed
0.3	Updated with minor changes
0.4	Updated Functional Description
0.5	General update
1.0	Update for AU915/US915
1.1	Chapter 2 updated for SF7BW500 and clarifification related to RF Gain Chapter 4.1.9 updated for ERP clarification Update for "Customer Mode" clarification
1.2	Chapter 2 updated for supported data rates
1.3	General update and document renamed
2.0	Valid from firmware V3.0, Build Count 194 WiMOD LoRaWAN EndNoce HCI specification integrated
2.1	Valid from firmware V3.0, Build Count 201 Chapter 4.2.17 added for RXC configuration (Multicast in Class C)
2.2	Valid from firmware V3.0, Build Count 203 Chapter 4.2.6 modified for clarifications (subband mask)
2.3	Valid from firmware V3.0, Build Count 207 4.2.16.2 modified for clarification (error indication)

Aim of this Document

This document contains the System Specification for the ProLink LoRaWAN® EndNode Modem firmware, which contains an implementation of a proprietary LoRa® communication additional to the LoRaWAN® radio stacks.

Moreover, it describes its ProLink LoRaWAN® EndNode Modem Host Controller Interface (HCI) protocol. This firmware can be used in combination with the WiMOD LoRa® radio module family.

Table of Contents

1. Introduction

1.1 Scope

The aim of the ProLink LoRaWAN® EndNode Modem firmware is to add a proprietary LoRa®[1] device to device communication to the standard LoRaWAN® stack. In this case, the proprietary LoRa® link provides the wireless transport of the data. Which data and how this data is transmitted is in the responsibility of the application and the host.

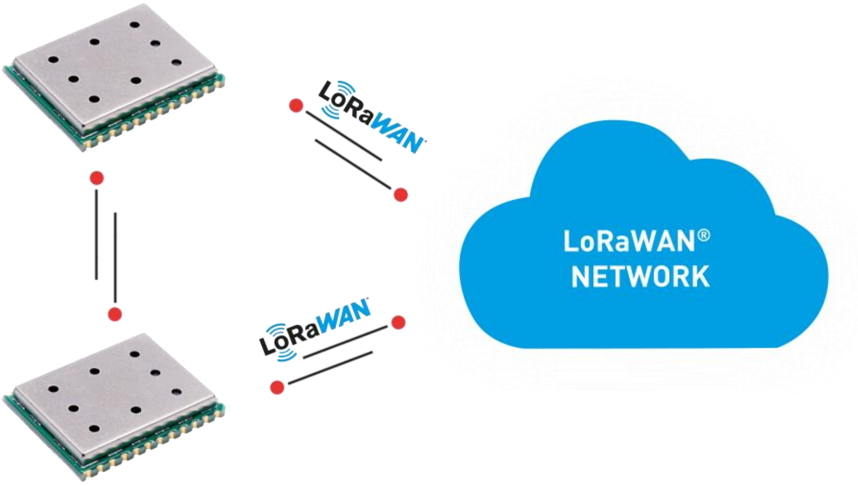

Fig. 1-1: Overview - ProLink LoRaWAN® System

[1] LoRa® is a Trademark of Semtech Corporation / LoRaWAN® is a Trademark of LoRa Alliance®

1.2 Firmware Overview

The ProLink LoRaWAN® EndNode Modem firmware provides the following features:

- Compliant with LoRaWAN® Specification V1.0.4 (see [1]) and RP002-1.0.1 LoRaWAN® Regional Parameters document (see [2])
- LoRaWAN® EU868, IN865, US915, AU915, AS923 and RU868 channel plans supported
- LoRaWAN® Class A and Class C (unicast/multicast messages supported)
- Over The Air Activation (OTAA) and Activation By Personalization (ABP)[1]
- Proprietary LoRa® device to device communication
- Multitasking Operating System WiMOD-OS with Automatic Power Saving (APS)
- Host Controller Interface (HCI) for access to radio functions & parameters
- EndNode Test Application required for the certification process

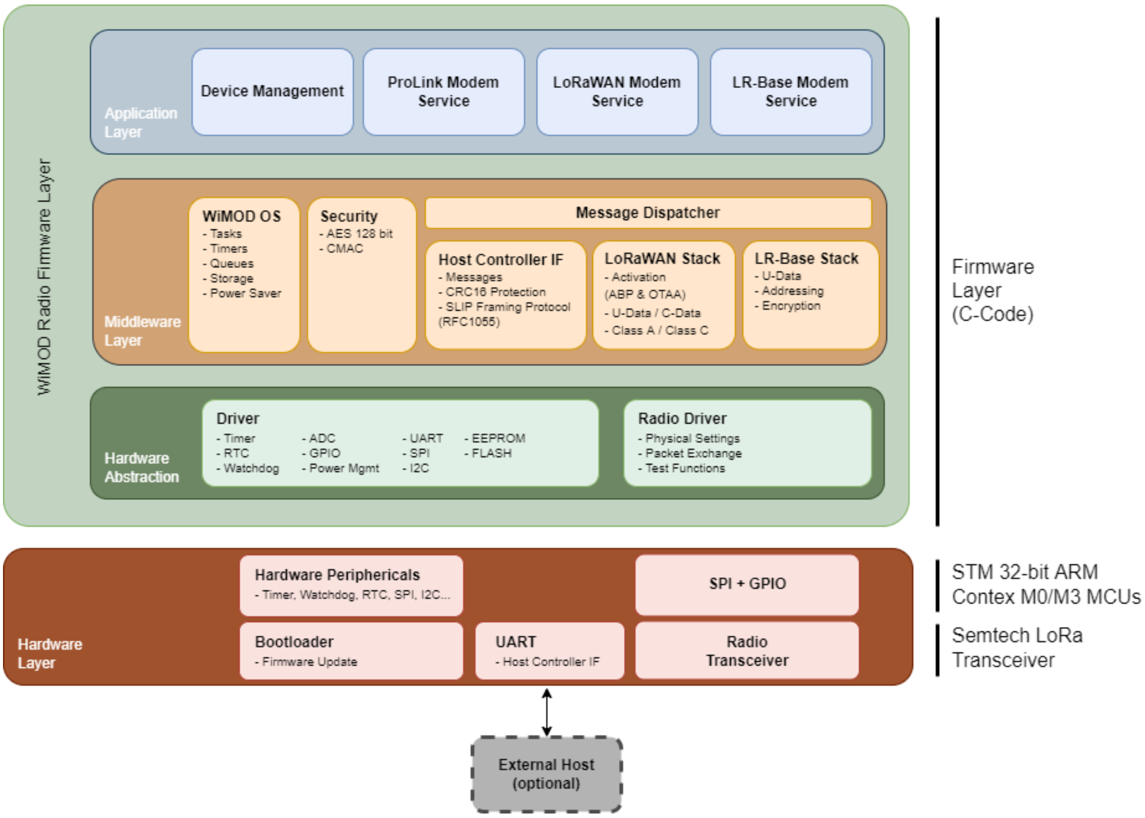

Fig. 1-2: ProLink LoRaWAN® EndNode Modem Firmware Architecture

[1] Activation By Personalization is available only for testing purposes.

1.3 HCI Protocol Overview

The ProLink LoRaWAN® EndNode Modem HCI protocol is designed to expose the radio firmware services to an external host controller.

The communication between host and the radio (WiMOD) is based on so called HCI messages which can be sent through a UART interface (see figure below). The ProLink LoRaWAN® EndNode Modem firmware provides several services for configuration, control and radio link access.

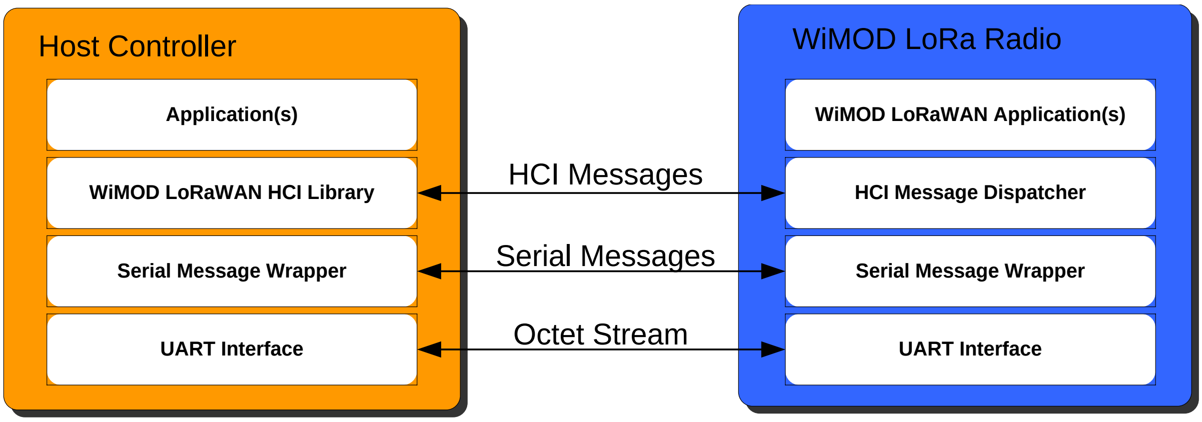

Fig. 1-3: Host Controller Communication

2. Functional Description

This chapter explains several points to exhibit the functionality of the ProLink LoRaWAN® EndNode Modem firmware.

2.1 General Services

The Device Management component provides general services for module configuration, module identification, and everything which is not related to the radio data exchange.

The main features are:

- Information elements for identification purposes (e.g. module type, device ID)
- Identification of the firmware version (FW version, build count, build date, FW name)
- Real Time Clock handling
- System Operation Modes (e.g. application mode, customer mode)
- Firmware Update
- Protocol stack selection (e.g. LoRaWAN® or proprietary LoRa® communication).

  Note: the LoRaWAN® stack will be automatically selected after a power-up reset.
- Configuration of common parameters for both stacks, such as the automatic power saving feature

2.1.1 Firmware Update

The end-device offers a fully automatic activation of the bootloader via the HCI interface, which could be used for future firmware updates.

2.1.2 Automatic Power Saving

In case the Automatic Power Saving is enabled, the end-device will enter low power mode whenever possible and the current consumption will be reduced to a typical low power current depending on the given hardware module, where the RTC remains running (for more information refer to the corresponding hardware datasheet, e.g. see [3]).

Note that if the LoRa® transceiver is configured in continuously reception mode (e.g class C support is enabled in LoRaWAN® stack) the current consumption will increase to the value which corresponds to the continuously listening mode.

If the LoRaWAN® stack is selected, the end-device does not enter low power mode direct after a transmission and this is not enabled before it either has received a downlink message or the second receive window is expired (no Rx indication).

The following picture shows an example of a voltage graph (multiplied by 10) measured at a 10 Ohm resistor on an iM880B-L module, including a LoRaWAN® transmission and both reception windows.

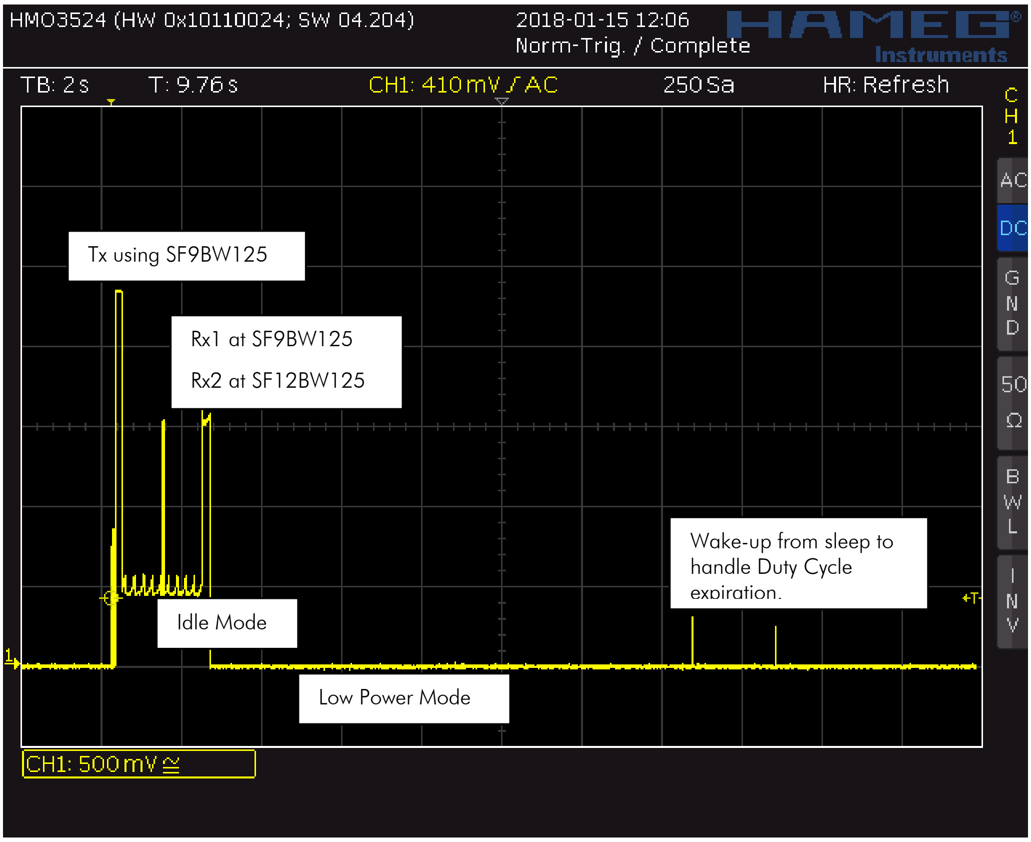

Fig. 2-1: Exemplary current consumption diagram - iM880B-L

2.2 Customization Services

This feature offers the configuration of some customization parameters for the end-device. For this, the Customer Mode should be selected. It is expected that those parameters are personalized before any LoRa® activity is started.

2.2.1 Device EUI

The end-device provides the services for read-out and configuration of the 64-bit unique Device EUI required by the LoRaWAN® specification (see [1]).

2.2.2 Band Selection

This parameter allows to configure the radio band to be used. Please refer to 4.2.6 for the configuration of this parameter.

2.2.3 Duty Cycle Control

The duty cycle limitation may be disabled for testing purposes. Please refer to 4.2.6 for the configuration of this parameter.

2.2.4 RF Sub-bands Configuration

In some regions, the ISM band is divided in several frequency sub-bands with different regulatory limitations. The end-device allows to modify the default settings in order to configure different values if required.

The parameters related to this feature are:

- Tx Power Limit

  configuration of the maximum allowed transmit power for each frequency sub-band (see corresponding regional HCI specification, [4]).

2.2.5 RF Gain

The RF gain defines an offset used to compensate possible transmission losses/gains in the final product (including circuit, matching, antennas...). This value should be rated in units of dBd (decibels relative to a half-wavelength dipole antenna, where *0dBd=2.15dBi*).

It is recommended to set this constant before the radio stack parameters to ensure a correct configuration of the device.

For more details refer to the appendix (chapter 6.2), which contains some examples for possible configurations. The most important parameters related to this feature are:

- Max. RF Power

  maximum RF output power corresponding to the module to be used (for more information refer to the corresponding hardware datasheet, e.g. see [3]).

- Max. allowed EIRP

  maximum allowed EIRP for the selected band, e.g. EU868, US915... (see [2] for more details).

- RF Gain

  configured RF gain related to the final product.

- Max. EIRP

  maximum EIRP available for the final product. This value is calculated as following:

  Max. EIRP = MIN (Max. allowed EIRP, Max. RF Power + RF Gain + 2.15dB)

- Configured EIRP

  EIRP configured for the next uplink radio message.

- Configured TRX power

  transmitted power to be configured in the transceiver to achieve the configured EIRP.

  The firmware considers that: *EIRP = TRX Power + RF Gain + ~2dB*

2.3 LoRaWAN® Services

2.3.1 Device Configuration

The end-device provides several features and parameters which can be configured under the radio stack configuration (see 4.2.6). The main parameters are:

- Band Selection: e.g. EU868.
- Uplink Data Rate
- Tx Power Level (EIRP)
- Adaptive Data Rate
- Duty Cycle Control
- Class A & Class C Selection
- Number of Retransmissions
- Header MAC Cmd capacity
- Private/Public LoRaWAN® network configuration

Some of these parameters, like the uplink data rate and the transmitted power level, are only used in certain situations (described in 2.3.1.1) and may change automatically during runtime or via LoRaWAN® MAC commands from network server side.

2.3.1.1 Adaptive Data Rate (ADR)

This feature allows an automatic data rate adaption from server side. Therefore the behaviour of the end-device depends on its configuration as defined in the following sections.

2.3.1.1.1 ADR enabled

According to the LoRaWAN® specification, if the end-device is configured with Activation By Personalization, the minimum data rate is used until the LoRaWAN® network requests a higher data rate through the LinkADRReq MAC command. In this case, the stored settings for data rate and the transmitted power level under the radio stack configuration will not be applied.

On the other hand, if the Over The Air Activation is selected and no special behaviour is established in the Regional Parameter document (see [2]), the first transmission of the Join Request happens with the already stored data rate under the radio stack configuration. Each data rate will be used twice and will be lowered after that (see 6.3.1). After a successful activation of the end-device, it will send an empty LoRaWAN® frame. For this, the data rate of the last Join Request will be used.

2.3.1.1.2 ADR disabled

If the ADR feature is disabled the data rate used by the end-device to send the application data will always remain unchanged and therefore the stored values for the data rate and tx power under the radio stack configuration will be applied. Note that this is valid as long as no special behaviour is established in the Regional Parameter document for the Join Procedure.

Using this configuration, if a LinkADRReq MAC Command from the LoRaWAN® network server is received, the end-device will accept it, where only the channel mask and tx power will be interpreted and accordingly modified. The data rate and redundancy parameters do not change.

Furthermore, the end-device will check for connectivity loss and therefore it sets the ADRAckReq bit after 64 successive uplinks without any Class A downlink response. In case no downlink is received in the following 32 uplinks, a link disconnect indication is sent to the host application via HCI, the ADRAckReq bit is reset to 0 and the data rate remains unchanged.

2.3.1.2 Class C Implementation

The end-device follows the Class C implementation as defined by the LoRaWAN® specification.

Additionally, following interpretations are considered by the current implementation:

- If any downlink which requires an uplink from the end-device (e.g. confirmed downlink) is received, the end-device will not listen in continuous mode until the pending uplink is sent.
- The indication with the information that no data has been received (including the corresponding error code if required) will only be forwarded to the application if the received downlink was addressed to the selected end-device. This is valid for the downlinks received during continuous listening mode.
- The firmware supports the configuration of up to three different multicast configurations. The end-device will use the multicast configuration once it is successful activated (by ABP or OTAA) and the class C support is enabled. The end-device will use the sequence counter included in the first received multicast downlink to synchronize its internal downlink sequence counter.

2.3.1.3 Battery Level Status

The firmware offers the possibility to update the status of the battery level of the end-device. This will be sent to the LoRaWAN® server in the reply to the DevStatusReq MAC command (see 4.2.9).

2.3.1.4 Non-Volatile Memory Handling

After a reset of the radio module the firmware will automatically restore the configuration available in the non-volatile memory. If this information is not available or corrupted, the initial firmware settings stored during production time will be used. In case the factory settings are missing or corrupted, the firmware will use its default settings (see 4.2.6.3).

In the same way, the DevNonce is stored in the non-volatile memory to ensure that this is incremented, which is required for a successful Over The Air Activation. If this information is not available or corrupted, a corresponding indication will be sent to the host application via HCI. Therefore it is recommended that the host application implements a restoring mechanism for this situation.

2.3.2 Device Activation

An end-device must be activated before it can communicate with a LoRaWAN® server. Two activation options are supported: Activation By Personalization (ABP) and Over The Air Activation (OTAA).

After a successful activation (ABP or OTAA), the end-device will send an empty unconfirmed uplink message ("alive" message) over the air. In case Class C is selected, the end-device will send an empty confirmed uplink message, as defined in the LoRaWAN® specification.

Note that after a reset of the radio module the firmware will automatically restore the last activation status stored in the non-volatile memory. Therefore, in case the end-device was previously ABP activated an empty uplink message will be sent. On the other side, the join procedure will be started if the end-device was OTAA activated or the join procedure was already initiated.

2.3.2.1 Activation By Personalization (ABP)

The activation parameters must be known on both sides - the end-device and the LoRaWAN® network. The following parameters are required:

- Device Address
- Network Session Key: used for MIC calculation and verification
- Application Session Key: used to encrypt and decrypt the payload field of application specific messages

Note that this activation method is available only for testing purposes, as the frame counters and other parameters are not stored persistently.

2.3.2.2 Over The Air Activation (OTAA)

The end-device can be configured and triggered to execute the so called join procedure defined in the LoRaWAN® specification. The result of a successful join procedure is a new device address, a new network session key and a new application session key.

The following parameters are required:

- Device EUI
- Join EUI
- Application Key

The end-device uses the frequencies defined by the corresponding radio band (see corresponding regional HCI specification, [4], for radio band configuration) to broadcast the Join Request message. Note that these transmissions follow the retransmissions backoff defined in the LoRaWAN® specification and the duty-cycle requirements, even if this is deactivated.

The join request will be retransmitted on a new randomly selected frequency channel if no join accept message is received. For this, the maximum number of retries for a join request is fixed to 12.

2.3.2.3 Activation Parameters

The parameters required for Over The Air Activation and Activation By Personalization are configurable via HCI interface. These parameters are not readable and they are stored in encrypted form in a non-volatile memory to resist a power cycle.

2.3.3 Data Exchange

2.3.3.1 Uplink Services

2.3.3.1.1 Uplink Unreliable Data Transmission

The end-device could send data in an unreliable way to the network server. This requires no acknowledgement from the network server.

If the end-device is configured by the network server to retransmit the unconfirmed/confirmed data frames (NbTrans) and an unconfirmed data frame is sent, a new transmission is not allowed before it either has received a downlink message or the second receive window of the last retransmission is expired.

The data frame will be retransmitted on a new frequency but using the same data rate (see 6.3.2).

2.3.3.1.2 Reliable Data Transmission

The end-device could send data in a reliable way to the network server. The server will acknowledge the received packet within the defined downlink timeslots. Note that if a downlink with the acknowledge bit unset is received, the end-device will ignore it.

If the end-device is configured by the network server to retransmit the unconfirmed/confirmed data frames (NbTrans) and a confirmed uplink has been sent, a new transmission is not allowed before it either has received an acknowledge or the second receive window of the last retransmission is expired.

In the absence of the acknowledgement the end-device will try to retransmit the same application payload, with a maximum number of retries (stored in the radio stack configuration). The same application payload will be retransmitted on a new uplink frame using a new randomly selected frequency channel. In case the ADR feature is enabled, each data rate will be used twice and will be lowered after that till the minimum data rate is achieved (see 6.3.3). The maximum number of retransmissions to be sent can be changed in the end-device configuration (see 4.2.6). The maximum value allowed is 254.

If the retransmission procedure finishes without success (e.g. maximum number of retransmission achieved or maximum payload size exceeded for the selected data rate), the corresponding error code will be sent (see 4.2.5.2).

2.3.3.1.3 Duty Cycle

A new transmission is not allowed if all channels are blocked by duty cycle. The application should try to send the data again (see 6.3.4).

2.3.3.1.4 Payload Size

The maximum length of the LoRaWAN® message is limited according to the maximum payload size defined in the Regional Parameters document (see [2]). In case the application data exceeds these limits the corresponding error code will be returned (see 0, 4.2.4).

2.3.3.2 Downlink Service

The end-device is able to receive packets within dedicated Rx timeslots scheduled as defined in the LoRaWAN® specification.

Depending on the type of received or not received data, the corresponding messages will be sent to the Host.

2.3.3.2.1 Message Acknowledge Bit

The end-device will automatically transmit an acknowledgement using an empty data message after the reception of a data message requiring a confirmation. This uplink is delayed 60 seconds, allowing the user to send piggybacked application payload if desired (see 6.3.5).

In case Class C is selected, in difference to the Class A implementation, the end-device will transmit an acknowledgement using an empty data message immediately after the reception of a data message requiring a confirmation.

2.3.3.2.2 Frame Pending Bit

The frame pending bit functionality is implemented according to the LoRaWAN® specification. An empty frame will be sent after the reception of a data message with the frame pending bit set to 1. This uplink is delayed 60 seconds, allowing the user to send piggybacked application payload if desired (see 6.3.6).

2.3.3.3 Frame Counter

The end-device implements a 32 bit frame counter.

2.3.4 MAC Commands

The end-device supports the MAC commands defined in the LoRaWAN® specification.

2.3.4.1 MAC Commands Request

The end-device allows the transmission of a MAC command request, either piggybacked in the header or in the Payload field with the Port field being set to 0.

2.3.4.1.1 Device Time Request

The firmware supports the DevTimeReq MAC command with following limitations:

- The end-device will automatically synchronize its RTC if a DeviceTimeAns is received in any unicast class A downlink.
- The GPS epoch time is used for the time synchronization on the end-device.
- Note that there is a small time different between the GPS epoch and the UTC time (leap seconds).
- Moreover, the configuration of the time zone is not available on the end-device and therefore no correction is performed.

Note that the LoRaWAN® network server must support the DeviceTimeReq/DeviceTimeAns MAC command to manage a successful time synchronization on the end-device.

2.3.4.2 MAC Commands Response

The end-device will send the answer to the MAC commands piggybacked within the next uplink. If this is not possible because they exceed the maximum length set under the radio stack configuration (max. 15 bytes), they will be sent immediately using the port 0 (see 6.3.7).

Note that the answers to the MAC commands that need to be retransmitted by the end-device until a Class A downlink is received, will be sent piggybacked in the header of the following uplinks.

2.4 Proprietary LoRa® Communication Services

The proprietary LoRa® communication is based on the LR-Base firmware available for the WiMOD LoRa® radio modules. The following features are accessible via the HCI interface:

- Configuration of the physical radio parameters to be used (for more information about the available values see corresponding regional HCI specification, [4]):
    - Frequency
    - Data rate
    - Transmission power

      Note: the configured value for RF Gain used to compensate possible transmission losses/gains in the final product will be applied in a similar way to this setting (see 2.2.5).
    - Coding rate

  Note that the Duty Cycle handling is within the responsibility of the application.
- Configuration of the communication parameters to be used:
    - Individual device address and/or group address
    - Encryption activation/deactivation
    - Encryption keys: the radio packets can be encrypted if needed. In this case, the used encryption scheme is based on the generic algorithm described in

IEEE 24 802.15.4/2006 Annex B [IEEE802154] using AES with a key length of 128 bits.

A message integrity code (MIC) field is attached to the radio message.

- Radio message exchange: communication based on unconfirmed radio messages.
    - Note that it is under the responsibility of the user application to ensure that the message reception is successful and request any retransmission if needed.
    - Maximum allowed radio message size: the proprietary LoRa® stack does not include any check for the maximum allowed radio message size. Its handling is within the responsibility of the application in order to ensure a successful radio communication. It is recommended not to exceed the following limitations depending on the effective modulation rate (please refer to [2] for some reference values).

3. HCI Communication

The communication between the WiMOD LoRa®[1] radio module and a host controller is based on messages. The following chapters describe the general message flow and message format.

3.1 Message Flow

The HCI protocol defines three different types of messages which are exchanged between the host controller and the radio module:

1. Command Messages: always sent from the host controller to the WiMOD LoRa® module to trigger a function.

2. Response Messages: sent from the radio module to the host controller to answer a preceding HCI request message.

3. Event Messages: can be sent from the radio module to the host controller at any time to indicate an event or to pass data which was received over the radio link from a peer device.

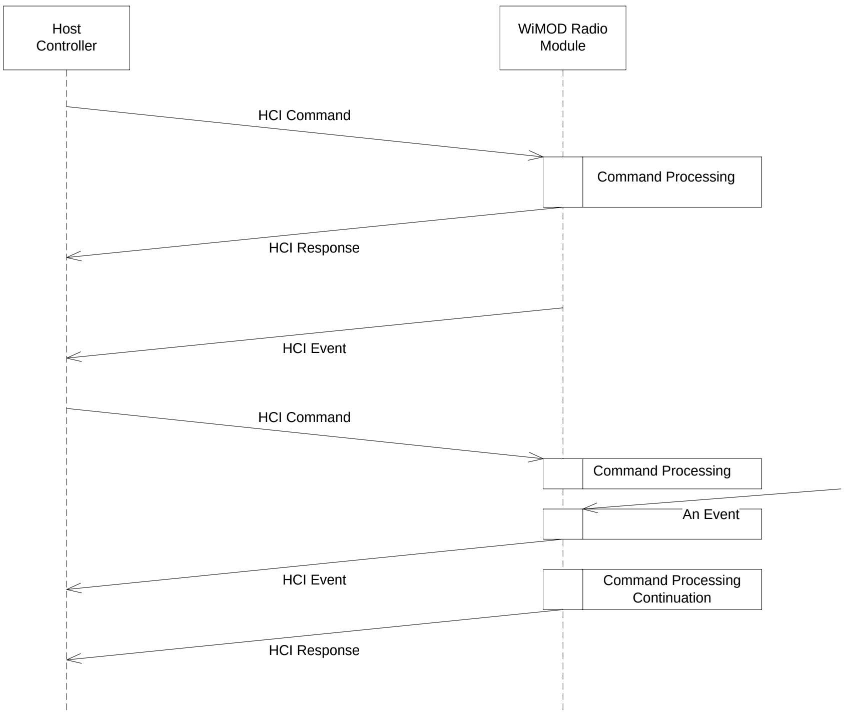

Fig. 3-1: HCI Message Flow

[1] LoRa is a registered trademark of Semtech Corporation. LoRaWAN is a registered trademark of the LoRa Alliance.

3.2 HCI Message Format

The following figure outlines the message format which is used for communication purposes.

HCI Message

Dst ID	Msg ID	Payload Field
8 Bit	8 Bit	n * 8 Bit

Fig. 3-2: HCI Message Format

3.2.1 Destination Endpoint Identifier (DstID)

This field identifies a logical service access point (endpoint) within a device. A service access point can be considered as a large firmware component which implements multiple services which can be called by corresponding HCI messages. This modular approach allows to support up to 256 independent components per device.

3.2.2 Message Identifier (MsgID)

This field identifies a specific type of message and is used to trigger a corresponding service function or to indicate a service response or event when sent to the host controller.

3.2.3 Payload Field

The Payload Field has variable length and transports message dependent parameters. The maximum size of this field is 300 Bytes.

3.2.4 Byte Ordering

The Payload Field usually carries data of type integer. Multi-octet integer values (2-Byte, 3-byte and 4-Byte integers) are transmitted in little endian order with least significant byte (LSB) first, unless otherwise specified in the corresponding HCI message information.

3.2.5 Frame Check Sequence Field (FCS)

Following the Payload Field a 16-Bit Frame Check Sequence (FCS) is added to support a reliable packet transmission. The FCS contains a 16-Bit CRC-CCITT cyclic redundancy check which enables the receiver to check a received packet for bit errors. The CRC computation starts from the Destination Endpoint Identifier Field and ends with the last byte of the Payload Field. The CRC ones complement is added before SLIP encoding (see chapter 6.6.5 for CRC16 example).

3.2.6 Communication over UART

The standard host controller communication interface is a UART interface. The ProLink LoRaWAN® HCI Protocol uses the SLIP (RFC1055) framing protocol when transmitted over asynchronous serial interfaces (UART).

3.2.6.1 SLIP Wrapper

The SLIP layer provides a mean to transmit and receive complete data packets over a serial communication interface. The SLIP coding is according to RFC 1055 [http://www.faqs.org/rfcs/rfc1055.html]

The next diagram explains how a HCI message is embedded in a SLIP packet.

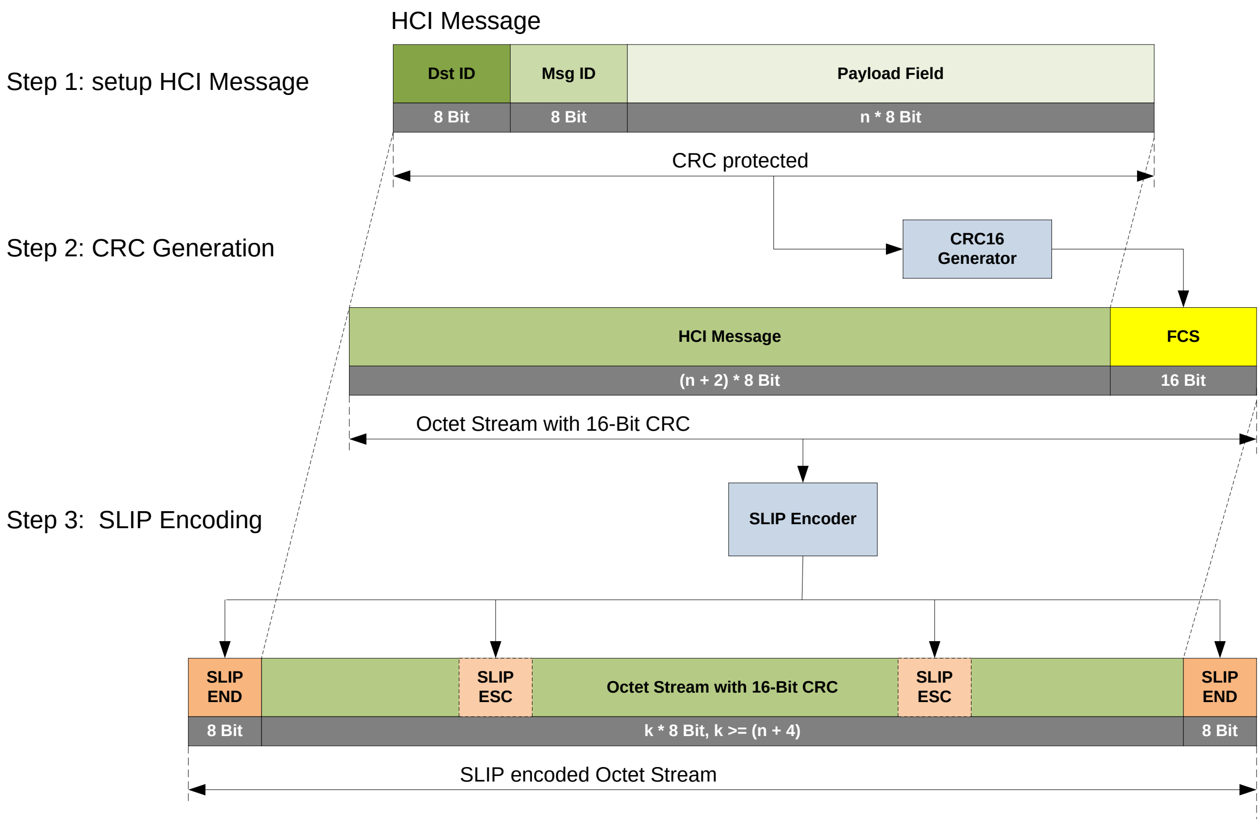

Fig. 3-3: Communication over UART

Note: The variable payload length is not explicitly transmitted over the UART communication link. Indeed it can be derived from the SLIP wrappers receiver unit.

3.2.6.2 Physical Parameters

The default UART settings are:

115200 bps, 8 Data bits, No Parity Bit, 1 Stop Bit

4. Firmware Services

This chapter describes the message format for the firmware services in detail. The services are ordered according to their corresponding endpoint.

4.1 Device Management Services

The Device Management endpoint provides general services for module configuration, module identification, and everything which is not related to the data exchange via radio link. The following services are available:

The main features are:

- Ping
- Reset
- Get Device Information
- Get Firmware Information
- RTC Configuration and RTC Alarm support
- System Operation Modes Handling
- Radio Stack Selection
- Device Configuration

4.1.1 Ping

This command is used to check if the serial connection is ok and if the connected radio module is alive. The host should expect a Ping Response within a very short time interval.

Message Flow

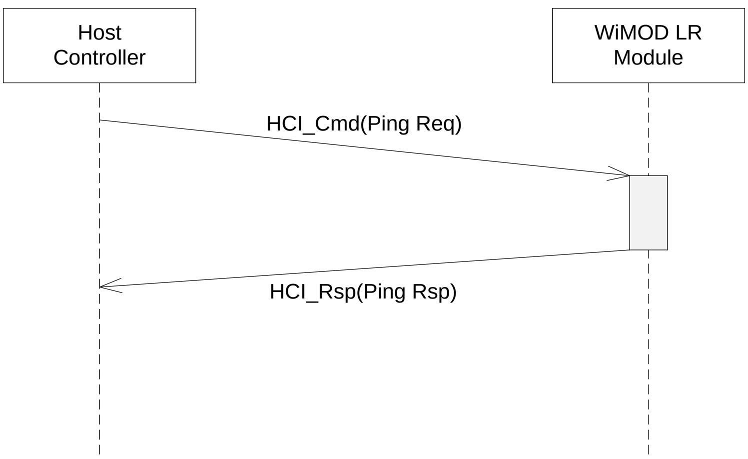

Fig. 4-1: Ping Request

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_PING_REQ	Ping Request
Length	0	no payload

Response Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_PING_RSP	Ping Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.2.2)

4.1.2 Reset

This message can be used to reset the radio module. The reset will be performed after approx. 200ms.

Message Flow

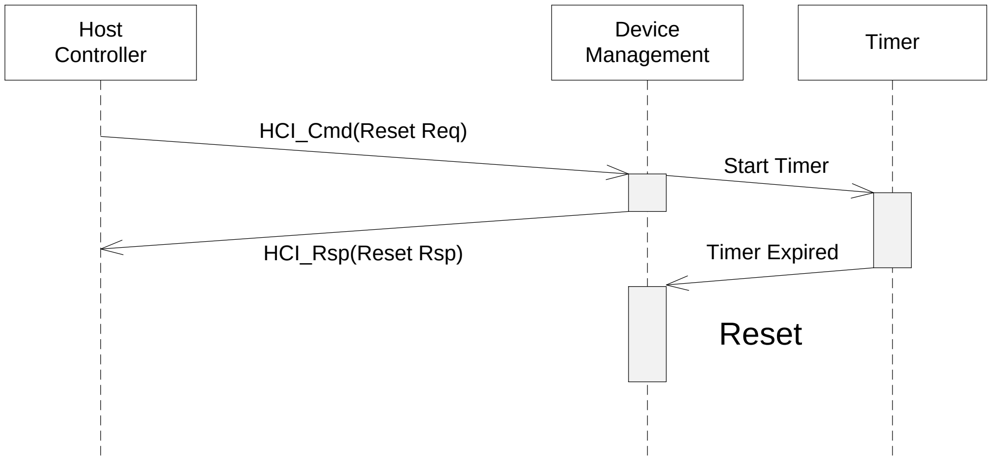

Fig. 4-2: Reset Request

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_RESET_REQ	Reset Request
Length	0	no payload

Response Message

This message acknowledges the Reset Request message.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_RESET_RSP	Reset Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.2.2)

4.1.3 Device Information

The radio firmware provides a service to readout some information elements for identification purposes.

4.1.3.1 Get Device Information

This message can be used to identify the local connected device. As a result the device sends a response message which contains a Device Information Field.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_DEVICE_INFO_REQ	Get Device Info Request
Length	0	no payload

Response Message

The response message contains the requested Device Information Field.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_DEVICE_INFO_RSP	Get Device Info Response
Length	10	10 octets
Payload[0]	Status Byte	see appendix (6.5.2.2)
Payload[1..9]	Device Information Field	see below

4.1.3.2 Device Information Field

The Device Information Field contains the following elements:

Offset	Size	Name	Description
0	1	ModuleType	Radio Module Identifier 0x90 = iM880A (obsolete) 0x92 = iM880A-L (128k) 0x93 = iU880A (128k) 0x98 = iM880B-L 0x99 = iU880B 0x9A = iM980A 0x9B = iU980A 0x9C = iM980B 0xA0 = iM881A
1	4	Device Address	32-Bit Device Address for radio communication
5	4	Device ID	32-Bit Device ID for identification purpose

4.1.4 Firmware Information

The radio firmware provides some further information to identify the firmware version itself.

4.1.4.1 Get Firmware Information

The following message can be used to identify the radio firmware.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_FW_INFO_REQ	Get FW Information
Length	0	no payload

Response Message

This message contains the requested information field.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_FW_INFO_RSP	Get FW Info Response
Length	1+n	1+n octets
Payload[0]	Status Byte	see appendix (6.5.2.2)
Payload[1..n]	Firmware Information Field	see below

4.1.4.2 Firmware Information Field

The Firmware Information Field contains the following elements:

Offset	Size	Name	Description
0	1	FW Version	Minor FW Version number
1	1	FW Version	Major FW Version number
2	2	Build Count	Firmware Build Counter, 16 Bit
4	10	Build Date	Firmware Build Date, e.g. : «16.04.2015»
14	m	Firmware Image	Name of Firmware Image and integrated LoRaWAN® radio stack, separated by semicolon

4.1.5 Device Status

The radio firmware provides some status information elements which can be read at any time.

4.1.5.1 Get Device Status

This message can be used to read the current device status.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_DEVICE_STATUS_REQ	Get Device Status Request
Length	0	no payload

Response Message

This response message contains the requested information elements.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_DEVICE_STATUS_RSP	Get Device Status Response
Length	84	84 octets
Payload[0]	Status Byte	see appendix (6.5.2.2)
Payload[1..15]	Device Status Field - Common	see below
Payload[16..59]	Device Status Field - LoRaWAN® Stack	see below
Payload[60..83]	Device Status Field - Proprietary Stack	see below

4.1.5.2 Device Status Field - Common

The Device Status Field common for both stacks includes the following information elements:

Offset	Size	Name	Description
0	1	System Tick Resolution	System Tick Resolution in milliseconds (e.g.: 1 = 1ms)
1	4	System Ticks	System Ticks since last start-up/reset
5	4	Target Time	RTC Time (see RTC Time Format)
9	2	NVM Status	Bit field for non-volatile memory blocks: Bit 0 = System Configuration Block, contains Operation Mode, Device ID Bit 1 = Radio Configuration Block, contains Radio Parameter and AES Key Bit Values : 0 = OK, block ok 1 = ERROR, block corrupt
11	2	Battery Level	Measured Supply Voltage in mV
13	2	Extra Status	Reserved Bit Field

4.1.5.3 Device Status Field - LoRaWAN® Stack

The Device Status Field related to the LoRaWAN® stack includes the following information elements:

Offset	Size	Name	Description
0	4	Tx U-Data	Number of unreliable radio packets transmitted
4	4	Tx C-Data	Number of reliable radio packets transmitted
8	4	Tx Error	Number of radio packets not transmitted due to an error
12	4	Rx1 U-Data	Number of unreliable radio packets received in 1st window
16	4	Rx1 C-Data	Number of reliable radio packets received in 1st window
20	4	Rx1 MIC-Error	Number of radio packets received in 1st window with MIC error
24	4	Rx2 U-Data	Number of unreliable radio packets received in 2nd window
28	4	Rx2 C-Data	Number of reliable radio packets received in 2nd window
32	4	Rx2 MIC-Error	Number of radio packets received in 2nd window with MIC error
36	4	Tx Join	Number of join request radio packets transmitted
40	4	Rx Accept	Number of join accept radio packets received

4.1.5.4 Device Status Field - Proprietary Stack

The Device Status Field related to the proprietary stack includes the following information elements:

Offset	Size	Name	Description
0	4	Rx Packets	Number of received radio packets with CRC OK
4	4	Rx Address Match	Number of received radio packets with CRC and Address OK
8	4	Rx CRC Error	Number of received radio packets with CRC Error
12	4	Tx Packets	Number of transmitted radio packets
16	4	Tx Error	Number of not transmitted radio packets
20	4	Tx Media Busy Events	Number of not transmitted packets due to LBT result "media busy"

4.1.6 Real Time Clock Support (RTC)

The radio module provides an embedded Real Time Clock which can be used to determine the module operating hours.

4.1.6.1 Get RTC Time

This message can be used to read the current RTC time value.
Note: the return value is zero when the RTC is disabled.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_RTC_REQ	Get RTC Value Request
Length	0	no payload

Response Message

This message contains the requested RTC value.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_RTC_RSP	Get RTC Value Response
Length	5	5 octets
Payload[0]	Status Byte	see appendix (6.5.2.2)
Payload[1-4]	32 Bit time	see RTC Time Format

4.1.6.2 Set RTC Time

This message can be used to set the RTC time to a given value.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_RTC_REQ	Set RTC Request
Length	4	4 octets
Payload[0-3]	32 Bit time value	see RTC Time Format

Response Message

This message acknowledges the Set RTC Request.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_RTC_RSP	Set RTC Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.2.2)

4.1.6.3 RTC Time Format

The RTC time is transmitted as a 32-Bit integer value.

Field	Content
Payload [n]	Bits 0 – 7
Payload [n+1]	Bits 8 – 15
Payload [n+2]	Bits 16 – 23
Payload [n+3]	Bits 24 – 31

The time value is coded as follows:

Value	Size	Position	Value Range
Seconds	6 Bits	Bit 0 – 5	0 – 59
Minutes	6 Bits	Bit 6 - 11	0 – 59
Months	4 Bits	Bit 12 – 15	1 – 12
Hours	5 Bits	Bit 16 – 20	0 – 23
Days	5 Bit	Bit 21 – 25	1 – 31
Years	6 Bit	Bit 26 – 31	0 – 63 -> 2000 - 2063

4.1.6.4 Set RTC Alarm

This message can be used to set a single or daily RTC alarm.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_RTC_ALARM_REQ	Set RTC Alarm Request
Length	4	4 octets
Payload[0]	Options	0x00 : single alarm 0x01 : daily repeated alarm
Payload[1]	Hour	Hour (range from 0 to 23)
Payload[2]	Minutes	Minutes (range from 0 to 59)
Payload[3]	Seconds	Seconds (range from 0 to 59)

Response Message

This message acknowledges the Set RTC Alarm Request.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_RTC_ALARM_RSP	Set RTC Alarm Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.2.2)

4.1.6.5 RTC Alarm Indication

This message indicates an RTC Alarm event.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_RTC_ALARM_IND	RTC Alarm Event Indication
Length	1	1 octets
Payload[0]	Status Byte	see appendix (6.5.2.2)

4.1.6.6 Get RTC Alarm

This message can be used to get a single or daily RTC alarm configuration.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_RTC_ALARM_REQ	Get RTC Alarm Request
Length	0	no payload

Response Message

This message acknowledges the Get RTC Alarm Request.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_RTC_ALARM_RSP	Get RTC Alarm Response
Length	6	6 octet
Payload[0]	Status Byte	see appendix (6.5.2.2)
Payload[1]	Alarm Status	0x00 : no alarm set 0x01 : alarm set
Payload[2]	Options	0x00 : single alarm 0x01 : daily repeated alarm
Payload[3]	Hour	Hour (range from 0 to 23)
Payload[4]	Minutes	Minutes (range from 0 to 59)
Payload[5]	Seconds	Seconds (range from 0 to 59)

4.1.6.7 Clear RTC Alarm

This message can be used to clear a pending RTC alarm.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_CLEAR_RTC_ALARM_REQ	Clear RTC Alarm Request
Length	0	no payload

Response Message

This message acknowledges the Clear RTC Alarm Request.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_CLEAR_RTC_ALARM_RSP	Clear RTC Alarm Response
Length	1	1 octets
Payload[0]	Status Byte	see appendix (6.5.2.2)

4.1.7 System Operation Mode Handling

The radio firmware can operate in different System Operation Modes to enable / disable specific features. The System Operation Mode is stored in the non-volatile memory and determined during firmware start-up.

The following System Operation Modes are supported:

- Standard / Application Mode
- Customer Mode - enables the customization services (see 2.2)

4.1.7.1 Get System Operation Mode

This message is used to read the current System Operation Mode.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_OPMODE_REQ	Get Operation Mode Request
Length	0	no payload

Response Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_OPMODE_RSP	Get Operation Mode Response
Length	2	2 octets
Payload[0]	Status Byte	see appendix (6.5.2.2)
Payload[1]	Current System Operation Mode	see appendix (6.1)

4.1.7.2 Set System Operation Mode

This message can be used to activate the next System Operation Mode. The mode value is stored in the non-volatile memory and a firmware reset is performed after approx. 200ms.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_OPMODE_REQ	Set Operation Mode Request
Length	1	1 octet
Payload[0]	Next Operation Mode	see appendix (6.1)

Response Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_OPMODE_RSP	Set Operation Mode Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.2.2)

4.1.8 Radio Stack Selection

The ProLink LoRaWAN EndNode Modem firmware supports two different radio modes, a LoRaWAN® operation mode and a proprietary mode based on the WiMOD LR-Base firmware. This feature allows to switch between both radio stacks.

Note that the LoRaWAN® stack will be automatically selected after a power-up reset. A switch to the proprietary stack is only allowed if no LoRaWAN® tasks are pending (e.g. receive windows expired and no uplink to server pending). The corresponding response via HCI will indicate if the operation succeeded.

4.1.8.1 Set Radio Stack

This service can be used to select the radio stack.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_RADIO_STACK_REQ	Set Radio Stack Request
Length	1	1 octets
Payload[0]	Radio Stack	0x00: LoRaWAN® 0x01: proprietary (based on LR Base)

Response Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_RADIO_STACK_RSP	Set Radio Stack Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.2.2)

4.1.8.2 Get Radio Stack

This service can be used to read the current radio stack.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_RADIO_STACK_REQ	Get Radio Stack Request
Length	0	no payload

Response Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_RADIO_STACK_RSP	Get Radio Stack Response
Length	2	2 octets
Payload[0]	Status Byte	see appendix (6.5.2.2)
Payload[1]	Radio Stack	0x00: LoRaWAN® 0x01: proprietary (based on LR Base)

4.1.9 Device Configuration

Both radio stacks provide some common features and parameters which can be configured via HCI:

- Automatic Power Saving

  this feature can be enabled to activate the automatic power saving mode. The module will enter a low power mode whenever possible. Wakeup via HCI message requires a sequence of ~40 additional wakeup characters (at 115200bps UART baud rate) "0xC0" prior to any SLIP encoded message.

- Miscellaneous Options

  this function enables the configuration of a HCI Power-Up indication, which is sent to the host when the module is ready to communicate after a power-up reset.

Note that those parameters are valid for both radio stacks.

4.1.9.1 Set Device Configuration

This service can be used to configure the device configuration. The new parameters will be saved directly in the non-volatile flash memory.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_DEVICE_CONFIG_REQ	Set Device Configuration Request
Length	4	4 octets
Payload[0]	Reserved	Reserved
Payload[1]	Power Saving Mode	0x00 : off 0x01 : automatic
Payload[2]	Reserved	Reserved
Payload[3]	Miscellaneous Options	Bit 3: HCI Power-Up Indication 0 = disabled 1 = enabled

Response Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_DEVICE_CONFIG_RSP	Set Device Configuration Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.2.2)

4.1.9.2 Get Device Configuration

This service can be used to read the current device configuration.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_DEVICE_CONFIG_REQ	Get Device Configuration Request
Length	0	no payload

Response Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_DEVICE_CONFIG_RSP	Get Device Configuration Response
Length	5	5 octets
Payload[0]	Status Byte	see appendix (6.5.2.2)
Payload[1]	Reserved	Reserved
Payload[2]	Power Saving Mode	0x00 : off 0x01 : automatic
Payload[3]	Reserved	Reserved
Payload[4]	Miscellaneous Options	Bit 3: HCI Power-Up Indication 0 = disabled 1 = enabled

4.1.9.3 Reset Device Configuration

This message can be used to restore the default device configuration.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_RESET_DEVICE_CONFIG_REQ	Reset Device Configuration Request
Length	0	no payload

Response Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_RESET_DEVICE_CONFIG_RSP	Reset Device Configuration Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.2.2)

4.1.9.4 HCI Power-Up Indication

Some module variants require a few milliseconds startup-time after power-up reset before the communication over the serial interface is possible. During that startup-phase the clock-system is configured and calibrated which is a prerequisite for accurate baud rate generation. The HCI Power-UP Indication message can be enabled to signal to the host controller when the module is ready to receive the first commands over the HCI interface.

Event Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_POWER_UP_IND	HCI Power-UP Indication
Length	0	No payload

4.1.9.5 Default Device Configuration

The following table lists the default device configuration used if no configuration is stored in the non-volatile memory.

Parameter	Value
Automatic Power Saving	0 = off
Miscellaneous Options	0 = HCI Power-Up Indication disabled

4.1.10 Device HCI Settings

This service can be used to configure HCI Parameters. Configurable HCI Parameters are:

- Baudrate

  baudrate to be used for the serial communication.

- Number of Tx Wakeup Chars

  number of Wakeup characters (SLIP_END = 0xC0) to be sent in the transmitted HCI messages by the end-device. This could be used to wake-up the host controller in case it implements a low power mode mechanism.

- Tx Hold Time

  the Tx Hold Time begins with the last transmitted character of a HCI message. Any new HCI message will be transmitted without additional Wakeup Characters during this time.

- Rx Hold Time

  the Rx Hold Time begins with the last received character of a HCI message. Any new HCI message will be transmitted without additional Wakeup Characters during this time.

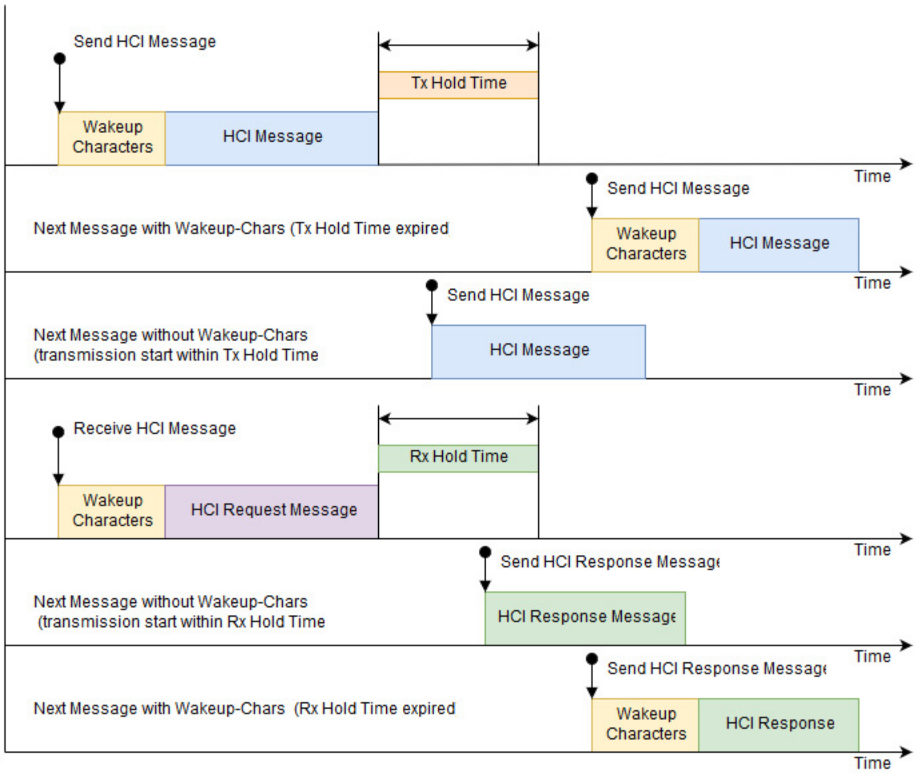

Fig. 4-3: HCI Settings

4.1.10.1 Set HCI Configuration

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_HCI_CFG_REQ	Set Device HCI Settings Request
Length	6	6 octets
Payload[0]	Store in non-volatile memory	0x00 : disabled 0x01 : enabled
Payload[1..5]	HCI Parameter Field	see below

Response Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_HCI_CFG_RSP	Set Device HCI Settings Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.2.2)

4.1.10.2 Get HCI Configuration

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_HCI_CFG_REQ	Get Device HCI Settings Request
Length	0	no payload

Response Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_HCI_CFG_RSP	Get Device HCI Settings Response
Length	6	6 octets
Payload[0]	Status Byte	see appendix (6.5.2.2)
Payload[1..5]	HCI Parameter Field	see below

4.1.10.3 HCI Parameter Field

This field contains all configurable HCI parameters.

Offset	Size	Content	Description
0	1]	Baudrate ID	0x03 : 57600 bps 0x04 :115200 bps
1	2	Number of Wakeup Chars	The number of transmitted Wakeup Characters. The maximal Number of Wakeup Chars corresponds to a maximum time of about 100ms. 57600 bps : range from 0 to 576 115200 bps: range from 0 to 1152
3	1	Tx Hold time	Hold time in ms (range from 0 to 255)
4	1	Rx Hold time	Hold time in ms (range from 0 to 255)

4.1.10.4 Default Configuration

The following table lists the default configuration.

Baudrate ID	0x04 :115200 bps
Number of Wakeup Chars	0
Tx Hold time	0
Rx Hold time	0

By a factory reset the settings are not restored to the default configuration.

4.2 LoRaWAN® Radio Link Services

The LoRaWAN® Service Access Point provides several services for radio communication according to the LoRaWAN® specification:

- End-Device Activation by Personalization (ABP)
- End-Device Activation Over-the-Air (OTAA)
- Unreliable Data Transmission
- Confirmed Data Transmission
- Ack + Data Reception
- Radio Stack Configuration including Automatic Power Saving
- Device EUI Configuration
- Factory Reset
- Network Status
- LoRaWAN® MAC Commands
- Multicast Configuration and Data Reception

4.2.1 End-Device Activation by Personalization (ABP)

This service provides a method for direct device activation via HCI.
Note: a device must be activated prior to any further data exchange with a server. After a successful activation, the device will send an empty uplink message ("alive" message) over the air (see 2.3.2.1).

The end-device activation service includes two HCI messages: a command message for parameter configuration and corresponding response message from the device.

Note: the activation parameters must be known on both sides - the end-device and the LoRaWAN® network.

Note that this activation is only available for testing purposes, as the frame counters and other parameters are not stored persistently.

4.2.1.1 Activate Device

This service can be used to activate the device via HCI. The following parameters will be stored in a non-volatile memory:

- Device Address

  a unique 32-Bit device-address, used for radio communication within a network

- Network Session Key

  a device-specific 128-Bit network session key used for MIC calculation and verification

- Application Session Key

  a device-specific 128-Bit application session key used to encrypt and decrypt the payload field of application specific messages

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_ACTIVATE_DEVICE_REQ	Activate Device Request
Length	36	36 octets
Payload[0..3]	32-Bit Device Address	32-Bit Integer (LSB first)
Payload[4..19]	128-Bit Network Session Key	Octet sequence (MSB first)
Payload[20..35]	128-Bit Application Session Key	Octet sequence (MSB first)

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_ACTIVATE_DEVICE_RSP	Activate Device Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.4.2)

4.2.1.2 Reactivate Device

This service can be used to activate the device via HCI using the parameters previously stored in the non-volatile memory.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_REACTIVATE_DEVICE_REQ	Reactivate Device Request
Length	0	0 octets

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_REACTIVATE_DEVICE_RSP	Reactivate Device Response
Length	5	1 octet
Payload[0]	Status Byte	see appendix (6.5.4.2)
Payload[1..4]	32-Bit Device Address	32-Bit Integer (LSB first)

4.2.2 End-Device Activation Over-the-Air

This service provides end-device activation over the air, i.e. the device can be configured and triggered to execute the so called join procedure defined in the LoRaWAN® specification. The result of a successful join procedure is a new device address, a new network session key and a new application session key (see 2.3.2.2).

The following HCI messages are implemented:

- a command message for parameter configuration and corresponding response message from the device
- a command message to start the join network procedure and corresponding response message from the device
- a join network radio packet transmit indication message
- a final join network indication message notifying the new device address to the host on success

Note: a device must be activated prior to any further data exchange with a server. After a successful activation, the device will send an empty uplink message ("alive" message) over the air.

4.2.2.1 Set Join Parameters

This service can be used to configure the over-the-air activation parameters which are used during the join procedure. These parameters will be stored in a non-volatile memory.

Note: these parameters must be known on the LoRaWAN® network side too.

- Join EUI

  a globally unique 64-Bit application ID

- Application Key

  a device-specific 128-Bit AES application key

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_JOIN_PARAM_REQ	Set Join Parameters Request
Length	24	24 octets
Payload[0..7]	64-Bit Join EUI	Octet sequence (MSB first)
Payload[8..23]	128-Bit Application Key	Octet sequence (MSB first)

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_JOIN_PARAM_RSP	Set Join Parameter Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.4.2)

4.2.2.2 Join Network Request

This service can be used to start the join network procedure. The module sends a join network radio packet and waits for a response from server side.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_JOIN_NETWORK_REQ	Join Network Request
Length	0	no payload

The command message is immediately answered by means of the following corresponding response message:

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_JOIN_NETWORK_RSP	Join Network Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.4.2)

4.2.2.3 Join Network Packet Transmit Indication

This HCI message is sent to the host after the join radio message has been sent to the server.

Event Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_JOIN_NETWORK_TX_IND	Join Network Tx Indication
Length	1 (+8)	1 (+8) octets
Payload[0]	Status & Payload Format	0x00 : radio packet sent 0x01 : radio packet sent, Tx Channel Info attached else : error, packet not sent
Payload[1]	Channel Index	See corresponding regional HCI specification, [4]
Payload[2]	Data Rate Index	See corresponding regional HCI specification, [4]
Payload[3]	NumTxPackets	Number of transmitted radio packets of last request
Payload[4]	TRX Power Level	Transmit power level configured in transceiver in dBm (min. value 0 dBm)[1]
Payload[5..8]	RF Message Airtime	32-Bit Airtime in milliseconds of transmitted radio message

[1] The minimum TRX power level depends on the radio module and it could slightly vary from the given power level value for the low power levels.

4.2.2.4 Join Network Indication

This message is sent to the host either after successful reception of a server join response packet or after the expiration of a complete join process without success.

Note: the maximum number of retries for a join request is fixed to 12.

Event Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_JOIN_NETWORK_IND	Join Network Indication
Length	1 (+4 or +9)	1 (+4 or +9) octets
Payload[0]	Status & Payload Format	0x00 : device successfully activated 0x01 : device successfully activated, Rx Channel Info attached else : error, device not activated
Payload[1..4]	New Device Address	32-Bit Integer (LSB first) Only sent if successfully activated
Payload[5]	Channel Index	See corresponding regional HCI specification, [4]
Payload[6]	Data Rate Index	See corresponding regional HCI specification, [4]
Payload[7]	RSSI	RSSI value in dBm (signed integer)
Payload[8]	SNR	SNR value in dB (signed integer)
Payload[9]	Rx Slot	Rx Slot value: 1: first window 2: second window

4.2.2.5 Set DevNonce Parameter

This service can be used to configure the DevNonce value to be used within the next Join Request message. This parameter will be stored in a non-volatile memory.

Note that according to the LoRaWAN® specification this value must be always incremented to ensure a successful Over The Air Activation. If this information is not available or corrupted, the corresponding indication will be sent to the host application via HCI. Therefore it is recommended that the host application implements a restoring mechanism for this situation.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_DEVNONCE_REQ	Set DevNonce Request
Length	2	2 octets
Payload[0..1]	DevNonce	16-Bit Integer (LSB first)

The command message is immediately answered by means of the following corresponding response message:

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_DEVNONCE_RSP	Set DevNonce Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.4.2)

4.2.2.6 Get DevNonce Parameter

This service can be used to read the DevNonce value to be used within the next Join Request message.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_DEVNONCE_REQ	Get DevNonce Request
Length	0	No payload

The command message is immediately answered by means of the following corresponding response message:

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_DEVNONCE_RSP	Get DevNonce Response
Length	3	3 octets
Payload[0]	Status Byte	see appendix (6.5.4.2)
Payload[1..2]	DevNonce	16-Bit Integer (LSB first)

4.2.2.7 Reset DevNonce Indication

If a valid DevNonce is not available or corrupted, the following indication will be sent to the host.
It is recommended that the host application implements a restoring mechanism for this situation.

Event Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_DEVNONCE_RESET_IND	Reset DevNonce Indication
Length	0	No payload

4.2.2.8 Set JoinNonce Parameter

This service can be used to configure a new JoinNonce value or reset the current one. This parameter will be stored in a non-volatile memory.

Note that according to the LoRaWAN® specification the JoinNonce is a non-repeating value provided by the Join Server and therefore the end-device will ignore a Join Accept if its JoinNonce is equal to the last received value. This value is initially set to 0, which allows the reception of any JoinNonce by the end-device.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_JOINNONCE_REQ	Set JoinNonce Request
Length	2	2 octets
Payload[0..1]	JoinNonce	16-Bit Integer (LSB first)

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_JOINNONCE_RSP	Set JoinNonce Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.4.2)

4.2.2.9 Get JoinNonce Parameter

This service can be used to read the last JoinNonce received by the LoRaWAN® network server.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_JOINNONCE_REQ	Get JoinNonce Request
Length	0	No payload

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_JOINNONCE_RSP	Get JoinNonce Response
Length	3	3 octets
Payload[0]	Status Byte	see appendix (6.5.4.2)
Payload[1..2]	JoinNonce	16-Bit Integer (LSB first)

4.2.3 Unreliable Data Transmission

This service can be used to send data in an unreliable way to the network server. No acknowledgement will be sent from the network server side and no retransmission method is available on the end-device side (see 2.3.3.1.1).

The following four HCI messages are implemented:

- a command message to initiate the unreliable packet transmission and corresponding response message from the device
- a final radio packet transmit indication message, notifying the end of transmission and optional radio channel information
- a link disconnect indication message, notifying a possible connectivity loss to the server

4.2.3.1 Send Unreliable Data Request

This command can be used to initiate an unreliable data transmission.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SEND_UDATA_REQ	Send Unreliable Data Request
Length	1+n	1+n octets
Payload[0]	LoRaWAN® Port	LoRaWAN® Port number (1..223)
Payload[1..n]	Application Payload	Application Layer Payload

Response Message

Field	Content	Description
Endpoint ID	LORAWAN _ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SEND_UDATA_RSP	Send Unreliable Data Response
Length	1 (+4)	1 (+4) octet
Payload[0]	Status Byte	see appendix (6.5.4.2)
Payload[1..4]	32-Bit time	32-Bit Integer (LSB first) Time [ms] remaining till channel available (sent if channel blocked by Duty Cycle, see appendix 6.5.4.2)

4.2.3.2 Unreliable Data Transmit Indication

This HCI message is sent to the host after the radio packet has been sent.

Event Message

Field	Content	Description
Endpoint ID	LORAWAN _ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SEND_UDATA_TX_IND	Send Unreliable Data Tx Indication
Length	1 (+8)	1 (+8) octets
Payload[0]	Status & Payload Format	0x00 : radio packet sent 0x01 : radio packet sent, Tx Channel Info attached else : error, radio packet not sent
Payload[1]	Channel Index	See corresponding regional HCI specification, [4]
Payload[2]	Data Rate Index	See corresponding regional HCI specification, [4]
Payload[3]	NumTxPackets	Number of transmitted radio packets of last request
Payload[4]	TRX Power Level	Transmit power level configured in transceiver in dBm (min. value 0 dBm)[1]
Payload[5..8]	RF Message Airtime	32-Bit Airtime in milliseconds of transmitted radio message

4.2.3.3 Link Disconnect Indication

This HCI message is sent to the host to notify a possible connectivity loss to the server in case the Adaptive Data Rate feature is disabled (see 2.3.1.1.2).

Event Message

Field	Content	Description
Endpoint ID	LORAWAN _ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_LINK_DISCONNECT_IND	Send Link Disconnect Indication
Length	0	No payload

[1] The minimum TRX power level depends on the radio module and it could slightly vary from the given power level value for the low power levels.

4.2.4 Reliable Data Transmission

This service can be used to send data in a reliable way to the network server. The server will acknowledge the received packet within the defined downlink timeslots (see 2.3.3.1.2).

The following three HCI messages are implemented for this service:

- a command message to initiate the reliable packet transmission and corresponding response message from the device
- a radio packet transmit indication message, notifying the end of transmission and optional radio channel information

Note: the ACK message and potential downlink data is outlined in the next chapter Ack & Data Reception, 4.2.5.

4.2.4.1 Send Reliable Data

This command can be used to initiate a reliable data transmission.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SEND_CDATA_REQ	Send Reliable Data Request
Length	1+n	1+n octets
Payload[0]	LoRaWAN® Port	LoRaWAN® Port number (1-223)
Payload[1..n]	Application Payload	Application Layer Payload

Response Message

Field	Content	Description
Endpoint ID	LORAWAN _ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SEND_CDATA_RSP	Send Reliable Data Response
Length	1 (+4)	1 (+4) octet
Payload[0]	Status Byte	see appendix (6.5.4.2)
Payload[1..4]	32-Bit time	32-Bit Integer (LSB first) Time [ms] remaining till channel available (sent if channel blocked by Duty Cycle, see appendix 6.5.4.2)

4.2.4.2 Reliable Data Transmit Indication

This HCI message is sent to the host after the radio packet has been sent or if the retransmission procedure finishes without success, containing in this case the corresponding error code.

Event Message

Field	Content	Description
Endpoint ID	LORAWAN _ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SEND_CDATA_TX_IND	Send Reliable Data Tx Indication
Length	1 (+8)	1 (+8) octets
Payload[0]	Status & Payload Format	0x00 : radio packet sent 0x01 : radio packet sent, Tx Channel Info attached 0x02 : error, maximum number of retransmissions reached 0x04: error, maximum payload size exceeded for current data rate else : error, radio packet not sent
Payload[1]	Channel Index	See corresponding regional HCI specification, [4]
Payload[2]	Data Rate Index	See corresponding regional HCI specification, [4]
Payload[3]	NumTxPackets	Number of transmitted radio packets of last request
Payload[4]	TRX Power Level	Transmit power level configured in transceiver in dBm (min. value 0 dBm)[1]
Payload[5..8]	RF Message Airtime	32-Bit Airtime in milliseconds of transmitted radio message

[1] The minimum TRX power level depends on the radio module and it could slightly vary from the given power level value for the low power levels.

4.2.5 Ack & Data Reception

The LoRaWAN® Stack is able to receive packets within dedicated Rx timeslots scheduled.

Depending on the type of received or not received data, one of the following three HCI event messages will be sent to the Host:

- Unreliable Data Indication
- Reliable Data Indication
- No-Data Indication

4.2.5.1 Unreliable Data Indication

This HCI message is sent to the host after reception of an unreliable radio packet containing application payload.

Event Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_RECV_UDATA_IND	Unreliable Data Indication
Length	1+n (+5)	1+n (+5) octets
Payload[0]	Status and Format	Bit 0 : 0 = no attachment 1 = Rx Channel Info attached Bit 1 : 0 = no Ack for uplink packet 1 = Ack received for last uplink packet Bit 2 : 0 = no downlink frame pending 1 = downlink frame pending
Payload[1]	LoRaWAN® Port	LoRaWAN® Port number (255 is set if the payload field is empty and no port is included in the downlink)
Payload[2..n]	Application Payload	Application Layer Payload
Payload[n+1]	Channel Index	See corresponding regional HCI specification, [4]
Payload[n+2]	Data Rate Index	See corresponding regional HCI specification, [4]
Payload[n+3]	RSSI	RSSI value in dBm (signed integer)
Payload[n+4]	SNR	SNR value in dB (signed integer)
Payload[n+5]	Rx Slot	Rx Slot value: 1: first window 2: second window 3: second window – scan mode

4.2.5.2 Reliable Data Indication

This HCI message is sent to the host after reception of a reliable radio packet containing application payload. The device will acknowledge the reception with a set Ack-Bit in the next reliable/unreliable uplink radio packet to the network server (see 2.3.3.2.1).

Event Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_RECV_CDATA_IND	Unreliable Data Indication
Length	1+n (+5)	1+n (+5) octets
Payload[0]	Status and Format	Bit 0 : 0 = no attachment 1 = Rx Channel Info attached Bit 1 : 0 = no Ack for uplink packet 1 = Ack received for last uplink packet Bit 2 : 0 = no downlink frame pending 1 = downlink frame pending
Payload[1]	LoRaWAN® Port	LoRaWAN® Port number (255 is set if the payload field is empty and no port is included in the downlink)
Payload[2..n]	Application Payload	Application Layer Payload
Payload[n+1]	Channel Index	See corresponding regional HCI specification, [4]
Payload[n+2]	Data Rate Index	See corresponding regional HCI specification, [4]
Payload[n+3]	RSSI	RSSI value in dBm (signed integer)
Payload[n+4]	SNR	SNR value in dB (signed integer)
Payload[n+5]	Rx Slot	Rx Slot value: 1: first window 2: second window 3: second window – scan mode

4.2.5.3 No-Data Indication

This HCI message is sent to the host in case no expected confirmation or data has been received as a result of prior reliable uplink radio packet.

Event Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_RECV_NO_DATA_IND	Ack Indication
Length	1(+1)	1(+1) octets
Payload[0]	Status and Format	0 = not further attachment Bit 1 : Wrong LoRaWAN® frame received (error code attached)
Payload[1]	Error Code	Bit 0 : Wrong MType received Bit 1 : Wrong Device Address received Bit 2 : Wrong MIC received Bit 3 : Unexpected FCnt received Bit 4 : Wrong MAC commands received (e.g. MAC commands simultaneously present in the payload field and the frame options field) Bit 5 : Wrong downlink received Bit 6 : Expected ACK missing

4.2.6 Radio Stack Configuration

The radio stack provides several features and parameters which can be configured via HCI:

- Data Rate

  this is used in certain situations and depends on the Adaptive Data Rate setting (see 2.3.1.1), otherwise it will be ignored.

- TX Power Level (EIRP)[1]

  this value is used in certain situations and depends on the Adaptive Data Rate setting (see 2.3.1.1), otherwise it will be ignored.

- Adaptive Date Rate

  this feature can be enabled to allow an automatic data rate adaption from server side (see 2.3.1.1).

- Duty Cycle Control

  this function can be disabled for test purpose.

  Note: this parameter can only be written in “Customer Mode” (see “System Operation Modes"), otherwise it will be ignored.

- Class A & C Support

  the radio can operate in one of these two modes.

- MAC Events Support

  this feature enables an event to forward the received MAC Commands to the corresponding host (for test purpose).

  If this feature is enabled, an additional HCI message will be sent to indicate the reception of MAC commands piggybacked in the header (see 4.2.13.2) and the MAC commands will be available via the standard UDATA or CDATA HCI messages (see 4.2.5.1 and 4.2.5.2) if these are received in port 0.

  Otherwise, if this feature is disabled, the MAC commands will not be visible to the corresponding host.

- Extended HCI Output Support

  this feature enables extended RF packet output format, where the Tx/Rx channel info is attached.

- Private LoRaWAN® Network Configuration

  this feature enables the configuration of a private LoRaWAN® network, which implies a change on the sync word.

[1] The RF Output Power may be limited by the radio module. For more information refer to the corresponding hardware datasheet (e.g. see [3]).

- Number of Retransmissions

  this value sets the maximum number of retries for a reliable radio packet where an acknowledgment is not received. Note that these retransmissions are additional to the frame retransmissions configured by the LoRaWAN® network server via the LinkADRReq MAC Command (NbTrans)

- Band Index

  used to configure the radio band to be used. In case a change in the radio band is requested, the end-device will be automatically deactivated.

  Note: this parameter can only be modified in "Customer Mode" (see "System Operation Modes"), otherwise it will be rejected.

- Header MAC Cmd Capacity

  used to configure the maximum length of the MAC commands to be piggybacked in the header within the next uplink. If the length of the reply exceeds this value, they will be sent immediately using the port 0.

- Sub-Band Mask1 (only for US915/AU915)

  used to select the 125 kHz bandwidth channels to be used for the transmission of the radio messages.

  Note: this parameter applies if the device is deactivated, otherwise it will be ignored.

- Sub-Band Mask2 (only for US915/AU915)

  used to select the 500 kHz bandwidth channels to be used for the transmission of the radio messages.

  Note: this parameter applies if the device is deactivated, otherwise it will be ignored.

4.2.6.1 Set Radio Stack Configuration

This service can be used to configure the integrated radio stack. This configuration will be stored in a non-volatile memory.

If the configured parameters are not allowed an error code will indicate that there is a wrong parameter. In this case, it is recommended to check that the uplink data rate and the transmitted power level are compatible with the selected band.

It is recommended to set again the desired radio stack configuration after a firmware update, especially if the band index is modified.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_RSTACK_CONFIG_REQ	Set Radio Stack Configuration Request
Length	7 / 9 (US915/AU915)	7 / 9 (US915/AU915) octets
Payload[0]	Data Rate Index	See corresponding regional HCI specification, [4]
Payload[1]	TX Power Level (EIRP)	Tx Power value in dBm (parameter range: 0 dBm to max. EIRP allowed by the device in 1 dB steps)
Payload[2]	Options	Bit 0: 0 = Adaptive Data Rate disabled 1 = Adaptive Data Rate enabled Bit 1: 0 = Duty Cycle Control disabled 1 = Duty Cycle Control enabled *(Customer Mode required)* Bit 2: 0 = Class A selected 1 = Class C selected Bit 5: 0 = public LoRaWAN® network 1 = private LoRaWAN® network Bit 6: 0 = standard RF packet output format 1 = extended RF packet output format: Tx/Rx channel info attached Bit 7: 0 = Rx MAC Command Forwarding disabled 1 = Rx MAC Command Forwarding enabled
Payload [3]	Reserved	Reserved
Payload [4]	Number of Retransmissions	Maximum number of retries for a reliable radio packet (parameter range: 0 to 254)
Payload [5]	Band Index	Radio Band Selection (see corresponding regional HCI specification, [4]) *(Customer Mode required)*

Payload [6]	Header MAC Cmd Capacity	Maximum length of the MAC commands to be piggybacked in the header (parameter range: 0 to 15)
Payload [7]	Sub-band Mask1	Sub-band Selection for 125 kHz bandwidth channels (see [2]) *Only if US915/AU915*
Payload [8]	Sub-band Mask2	Sub-band Selection for 500 kHz bandwidth channels (see [2]) *Only if US915/AU915*

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_RSTACK_CONFIG_RSP	Set Radio Stack Configuration Response
Length	1 (+1)	1 (+1) octet
Payload[0]	Status Byte	see appendix (6.5.4.2)
Payload[1]	Wrong Parameter Error Code	Bit 0: 0 = Correct Data Rate 1 = Wrong Data Rate Bit 1: 0 = Correct TX Power Level 1 = Wrong TX Power Level Bit 2-4: not used Bit 5: 0 = Correct Band Index 1 = Wrong Band Index Bit 6-7: not used Only sent if status byte contains LORAWAN_STATUS_WRONG_PARAMETER

4.2.6.2 Get Radio Stack Configuration

This service can be used to read the current radio stack configuration.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_RSTACK_CONFIG_REQ	Get Radio Stack Configuration Request
Length	0	no payload

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_RSTACK_CONFIG_RSP	Get Radio Stack Configuration Response
Length	8 / 10 (US915/AU915)	8 / 10 (US915/AU915) octets
Payload[0]	Status Byte	see appendix (6.5.4.2)
Payload[1]	Data Rate Index	See corresponding regional HCI specification, [4]
Payload[2]	TX Power Level (EIRP)	Tx Power value in dBm (parameter range: 0 dBm to max. EIRP allowed by the device in 1 dB steps)
Payload[3]	Options	Bit 0: 0 = Adaptive Data Rate disabled 1 = Adaptive Data Rate enabled Bit 1: 0 = Duty Cycle Control disabled 1 = Duty Cycle Control enabled Bit 2 : 0 = Class A selected 1 = Class C selected Bit 5: 0 = public LoRaWAN® network 1 = private LoRaWAN® network Bit 6: 0 = standard RF packet output format 1 = extended RF packet output format: Tx/Rx channel info attached Bit 7: 0 = Rx MAC Command Forwarding disabled 1 = Rx MAC Command Forwarding enabled
Payload [4]	Reserved	Reserved
Payload [5]	Number of Retransmissions	Maximum number of retries for a reliable radio packet (parameter range: 0 to 254)
Payload [6]	Band Index	Radio Band Selection (see corresponding regional HCI specification, [4])
Payload [7]	Header MAC Cmd Capacity	Maximum length of the MAC commands to be piggybacked in the header (parameter range: 0 to 15)
Payload [8]	Sub-band Mask1	Sub-band Selection for 125 kHz bandwidth channels (see [2]) *Only if US915/AU915*
Payload [9]	Sub-band Mask2	Sub-band Selection for 500 kHz bandwidth channels (see [2]) *Only if US915/AU915*

4.2.6.3 Default Radio Stack Configuration

The following table lists the default radio stack configuration used if no factory settings are stored in the non-volatile memory.

Parameter	Value
Band Index	See corresponding regional HCI specification, [4]
Data Rate Index	See corresponding regional HCI specification, [4]
TX Power Level (EIRP)	Max. EIRP allowed and supported
Adaptive Data Rate	Enabled
Duty Cycle Control	Enabled
Class C Support	Disabled (Class A selected)
Private LoRaWAN® Network Configuration	Disabled (Public selected)
MAC Events Support	Disabled
Extended HCI Output Support	Disabled
Number of Retransmissions	0
Header MAC Cmd Capacity	15

4.2.6.4 Get Supported Bands Information

This service can be used to get information related to the supported bands by the firmware. Moreover, the maximum supported EIRP for the each band is provided.

Note that the maximum supported EIRP depends on the radio module and the configured RF Gain value (for more information refer to [3]).

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_SUPPORTED_BANDS_REQ	Get Supported Bands Request
Length	0	no payload

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_SUPPORTED_BANDS_RSP	Get Supported Bands Response
Length	1+2+2*n	1+2+2*n octets
Payload[0]	Status Byte	see appendix (6.5.4.2)
Payload[1]	Band Index 0	Band Index 0 (see corresponding regional HCI specification, [4])
Payload[2]	Max. EIRP for Band Index 0	Maximum supported EIRP for Band Index 0 in dBm
Payload[1+2*n]	Band Index n	Band Index n (see corresponding regional HCI specification, [4])
Payload[2+2*n]	Max. EIRP for Band Index n	Maximum supported EIRP for Band Index n in dBm

4.2.7 Device EUI Configuration

The LoRaWAN® specification requires a 64-bit unique Device EUI. The firmware provides the following services for read-out and configuration.

Note: the 64-bit Device EUI is independent from the 32-bit Device ID which can be considered as an IMST product serial number.

4.2.7.1 Get Device EUI

This message can be used to read the 64-bit Device EUI.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_DEVICE_EUI_REQ	Get Device EUI
Length	0	no payload

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_DEVICE_EUI_RSP	Get Device EUI Response
Length	9	9 octets
Payload[0]	Status Byte	see appendix (6.5.4.2)
Payload[1-8]	64-Bit Device EUI	Octet sequence (MSB first)

4.2.7.2 Set Device EUI

This message can be used to write the 64-bit Device EUI.

Note: this parameter will be stored in a non-volatile memory and can only be written in "Customer Mode" (see "System Operation Modes").

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_DEVICE_EUI_REQ	Set Device EUI Request
Length	8	8 octets
Payload[0-7]	64-Bit Device EUI	Octet sequence (MSB first)

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_DEVICE_EUI_RSP	Set Device EUI Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.4.2)

4.2.8 Custom Configuration

The following custom parameters can be configured via HCI:

- RF Gain

  the RF gain defines an offset used to compensate possible transmission losses/gains in the final product (including circuit, matching, antennas...). This value should be rated in units of dBd (decibels relative to a half-wavelength dipole antenna, where 0dBd=2.15dBi).

The firmware provides the following services for read-out and configuration.

4.2.8.1 Get Custom Configuration

This message can be used to read the custom configuration parameters.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_CUSTOM_CFG_REQ	Get Custom Configuration
Length	0	no payload

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_CUSTOM_CFG_RSP	Get Custom Configuration Response
Length	2	2 octets
Payload[0]	Status Byte	see appendix (6.5.4.2)
Payload[1]	RF Gain	RF Gain value in dBd (parameter range: -128 dBd to 127 dBd in 1 dB steps)

4.2.8.2 Set Custom Configuration

This message can be used to configure the custom parameters.

Note: this parameters will be stored in a non-volatile memory and can only be written in "Customer Mode" (see "System Operation Modes").

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_ CUSTOM_CFG_REQ	Set Custom Configuration Request
Length	1	1 octets
Payload[0]	RF Gain	RF Gain value in dBd (parameter range: -128 dBd to 127 dBd in 1 dB steps)

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_ CUSTOM_CFG_RSP	Set Custom Configuration Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.4.2)

4.2.8.3 Default Custom Configuration

The following table lists the default custom configuration used if no configuration is stored in the non-volatile memory.

Parameter	Value
RF Gain	0 dBd

4.2.9 Battery Level Status

The firmware offers the possibility to update the status of the battery level of the end-device. The last configured value will be sent to the LoRaWAN® server in the reply to the DevStatusReq MAC command.

Following services are available for its configuration.

4.2.9.1 Set Battery Level Status

This message can be used to configure the battery level status.

Note: this parameter will not be stored in the non-volatile memory and will be restored to its default value after a module reset. Therefore it is recommended to update the battery level status after each reset if this is required by the application.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_BATTERY_LEVEL_REQ	Set Battery Level Status Request
Length	1	1 octet
Payload[0]	Battery Level Status	Battery Level Status: 0: mains/extern powered 1-254: battery status (where 1 means minimum and 254 maximum) 255: undefined

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_BATTERY_LEVEL_RSP	Set Battery Level Status Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.4.2)

4.2.9.2 Default Battery Level Status

The default battery level status is set to 0xFF (undefined). This means that the device will answer to the DevStatusReq MAC command with this value, informing that it was not able to measure the battery level.

4.2.10 Factory Reset

This service allows to restore the initial firmware settings stored during production time.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_FACTORY_RESET_REQ	Factory Reset Request
Length	0	no payload

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_FACTORY_RESET_RSP	Factory Reset Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.4.2)

4.2.11 Device Deactivation

This service allows to deactivate the LoRaWAN® end-device, i.e. further data exchange over the air is disabled.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_DEACTIVATE_DEVICE_REQ	Deactivate Device Request
Length	0	no payload

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_DEACTIVATE_DEVICE_RSP	Deactivate Device Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.4.2)

4.2.12 Network/Activation Status

This service allows to read the current network / activation status.

In case the device has been successfully activated, following parameters are included in the response message:

- Device Address

  unique 32-Bit device-address used for radio communication within a network

- Data Rate Index

  current data rate used for unreliable and confirmed data packets in the next uplink

- Power Level (EIRP)

  current configured transmit power level

- Maximum Payload Size

  maximum number of bytes allowed in the payload field, according to the current data rate and taking into account the possible MAC commands piggybacked in the header

- Number of Transmissions by MAC

  number of transmissions configured by the LoRaWAN® network server via the LinkADReq (NbTrans)

The device is inactive as default activation status if no factory settings are stored in the non-volatile memory.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_NWK_STATUS_REQ	Get Network Status Request
Length	0	no payload

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_NWK_STATUS_RSP	Get Network Status Response
Length	2 (+8)	2 (+8) octets
Payload[0]	Status Byte	see appendix (6.5.4.2)
Payload[1]	Network Status:	1 octet 0x00 : device inactive 0x01 : active (ABP) 0x02 : active (OTAA) 0x03 : joining (OTAA)
Payload[2..5]	Device Address	32-Bit Integer (LSB first)
Payload[6]	Data Rate Index	See corresponding regional HCI specification, [4]
Payload[7]	Power Level (EIRP)	Power level in dBm
Payload[8]	Maximum Payload Size	Maximum number of bytes allowed in the payload field
Payload[9]	Number of Transmissions by MAC	Number of transmissions configured by the LoRaWAN® network server

4.2.13 LoRaWAN® MAC Commands Support

The service allows to send and show several LoRaWAN® stack internal MAC commands.

The following three HCI messages are implemented:

- a command message to initiate the transmission of a MAC command and corresponding response message from the device
- a MAC command receive indication message, which must be enabled via configuration

Note: the standard radio packet transmit indication message will notify the end of the transmission.

4.2.13.1 Send MAC Command

This command can be used to send a single MAC command. If possible, the end-device will send the MAC command piggybacked in the header.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SEND_MAC_CMD_REQ	Send MAC Command Request
Length	1+n	1+n octets
Payload[0]	Data Service Type	0 : Unreliable Data Service 1: Reliable Data Service
Payload[1]	Command ID	MAC command according to LoRaWAN® spec. (see [1])
Payload[2..n]	Options	MAC Command Parameters

Response Message

Field	Content	Description
Endpoint ID	LORAWAN _ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SEND_MAC_CMD_RSP	Send MAC Command Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.4.2)

4.2.13.2 Receive MAC Command

This HCI message is sent to the host after reception of a radio packet including MAC command(s) piggybacked in the header. The application payload will be forwarded via the standard UDATA or CDATA HCI messages.

Note: this HCI event message must be enabled via configuration.

Event Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_RECV_MAC_CMD_IND	MAC Command Indication
Length	1+n (+5)	1+n (+5) octets
Payload[0]	Status and Format	Bit 0 : 0 = no attachment 1 = Rx Channel Info attached
Payload[1..n]	MAC Command List	See [1]
Payload[n+1]	Channel Index	See corresponding regional HCI specification, [4]
Payload[n+2]	Data Rate Index	See corresponding regional HCI specification, [4]
Payload[n+3]	RSSI	RSSI value in dBm (signed integer)
Payload[n+4]	SNR	SNR value in dB (signed integer)
Payload[n+5]	Rx Slot	Rx Slot value: 1: first window 2: second window 3: second window – scan mode

4.2.14 Network Time Request

The service can be used to initiate the transmission of the DeviceTimeReq MAC command to request the network time.

The following three HCI messages are implemented:

- a command message to initiate the transmission of the DeviceTimeReq MAC command and corresponding response message from the device
- an indication message, notifying the reception or lack of reception of the expected MAC response

Note that the end-device will automatically synchronize its RTC if a DeviceTimeAns is received in any unicast class A downlink (see 2.3.4.1.1 for more details).

4.2.14.1 Send Network Time Request

This command can be used to initiate the transmission of the DeviceTimeReq MAC command. The end-device will send this MAC command using a reliable data transmission.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SEND_DEVICETIMEREQ_REQ	Send DeviceTimeReq Request
Length	0	No payload

Response Message

Field	Content	Description
Endpoint ID	LORAWAN _ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SEND_DEVICETIMEREQ _RSP	Send MAC DeviceTimeReq Response
Length	1 (+4)	1 (+4) octet
Payload[0]	Status Byte	see appendix (6.5.4.2)
Payload[1..4]	32-Bit time	32-Bit Integer (LSB first) Time [ms] remaining till channel available (sent if channel blocked by Duty Cycle, see appendix 6.5.4.2)

4.2.14.2 Network Time Indication

This message indicates if the requested time from network server has been received.

Event Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_DEVICETIMEANS_IND	DeviceTimeAns Indication
Length	1(+4)	1(+4) octets
Payload[0]	Status Byte	see appendix (6.5.4.2)
Payload[1..4]	32-Bit time	32-Bit Integer (LSB first) GPS epoch time [s] received by the LoRaWAN® network server (sent if successful response received)

4.2.15 Multicast Configuration

This service provides a method for configuration of the multicast parameters via HCI.

The multicast parameters must be known on both sides - the end-device and the LoRaWAN® network.

The end-device will use the multicast configuration once it is successful activated (by ABP or OTAA) and the class C support is enabled. The end-device will use the sequence counter included in the first received multicast downlink to synchronize its internal downlink sequence counter.

Note: the multicast parameters are not stored in a non-volatile memory and therefore they do not persist after a reset of the device.

The radio stack allows the configuration of several parameters via HCI:

- Multicast Index
  used to identify the multicast parameters (up to three different multicast configurations are allowed).
- Multicast Device Address
  a 32-Bit device-address, used for multicast communication on the network.
- Multicast Network Session Key
  a 128-Bit network session key used for MIC calculation and verification associated to the multicast device address.
- Multicast Application Session Key
  a 128-Bit application session key used to encrypt and decrypt the payload field of application specific messages associated to the multicast device address.

4.2.15.1 Set Multicast Configuration

This service can be used to configure the multicast parameters.

Note: the selected multicast device address will be automatically enabled.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_MCAST_CONFIG_REQ	Set Multicast Configuration Request
Length	37	37 octets
Payload[0]	Multicast Index	Multicast index (range from 0 to 2)
Payload[1..4]	32-Bit Device Address	32-Bit Integer (LSB first)
Payload[5..20]	128-Bit Network Session Key	Octet sequence (MSB first)
Payload[21..36]	128-Bit Application Session Key	Octet sequence (MSB first)

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_MCAST_CONFIG_RSP	Set Multicast Configuration Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.4.2)

4.2.15.2 Get Multicast Configuration

This service can be used to get some information related to a single multicast configuration.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_MCAST_CONFIG_REQ	Get Multicast Configuration Request
Length	1	1 octets
Payload[0]	Multicast Index	Multicast index (range from 0 to 2)

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_MCAST_CONFIG_RSP	Get Multicast Configuration Response
Length	7	7 octets
Payload[0]	Status Byte	see appendix (6.5.4.2)
Payload[1]	Multicast Index	Selected multicast index
Payload[2]	Multicast Status	Current status of the selected index (0: inactive; 1: active)
Payload[3..6]	32-Bit Device Address	32-Bit Integer (LSB first)

4.2.15.3 Remove Multicast Configuration

This service can be used to remove a single multicast configuration.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_DEL_MCAST_CONFIG_REQ	Remove Multicast Configuration Request
Length	1	1 octets
Payload[0]	Multicast Index	Multicast index (range from 0 to 2)

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_DEL_MCAST_CONFIG_RSP	Remove Multicast Configuration Response
Length	1	1 octets
Payload[0]	Status Byte	see appendix (6.5.4.2)

4.2.16 Multicast Data Reception

After a successful LoRaWAN® activation, a device in class C mode can start receiving multicast downlinks in the configured multicast device addresses.

Depending if the received data is valid or invalid one of the following HCI event messages will be sent to the Host:

- Multicast Data Indication
- Invalid Multicast Data Indication

4.2.16.1 Multicast Data Indication

This HCI message is sent to the host after reception of a valid multicast radio packet.

Event Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_RECV_MCAST_DATA_IND	Multicast Data Indication
Length	1+n (+5)	1+n (+5) octets
Payload[0]	Status and Format	Bit 0 : 0 = no attachment 1 = Rx Channel Info attached
Payload[1..4]	32-Bit Device Address	32-Bit Integer (LSB first)
Payload[5]	LoRaWAN® Port	LoRaWAN® Port number
Payload[6..n]	Application Payload	Application Layer Payload
Payload[n+1]	Channel Index	See corresponding regional HCI specification, [4]
Payload[n+2]	Data Rate Index	See corresponding regional HCI specification, [4]
Payload[n+3]	RSSI	RSSI value in dBm
Payload[n+4]	SNR	SNR value in dB
Payload[n+5]	Rx Slot	Rx Slot value

4.2.16.2 Invalid Multicast Data Indication

This HCI message is sent to the host after reception of an invalid multicast radio packet.

Event Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_RECV_MCAST_NO_DATA_IND	Invalid Multicast Data Indication
Length	6	6 octets
Payload[0]	Status and Format	Bit 1 : wrong LoRaWAN® frame received (error code attached)
Payload[1]	Error Code	Bit 0 : Wrong MType received (e.g. MType field must carry the value for Unconfirmed Data Down in a multicast downlink frame) Bit 1 : Wrong Device Address Bit 2 : Wrong MIC received Bit 3 : Unexpected FCnt received Bit 4 : MAC commands Error (e.g. MAC commands are not allowed in a multicast downlink frame) Bit 5 : Wrong downlink received Bit 7 : Multicast downlink error (e.g. the ACK and ADRACKReq bits must be zero in a multicast downlink frame)
Payload[2..5]	32-Bit Device Address	32-Bit Integer (LSB first)

4.2.17 Multicast Rx Configuration – Class C

This service provides a method for configuration of the multicast rx channel parameters to be used in the continuously reception window via HCI.

The end-device will use the new configuration the next time that the continuously reception window is open.

Note: the multicast channel parameters are not stored in a non-volatile memory and therefore they do not persist after a reset of the device.

The radio stack allows the configuration of several parameters via HCI:

- Class C Address
  used to select the parameters to be applied for continuously reception window. If multicast is selected, the configured multicast frequency and data rate (RXC parameters) will be used, otherwise the Rx2 parameters will be set.
- Multicast Data Rate
  used to configure the data rate to be used for the continuously reception window if the multicast reception is selected.
- Multicast Frequency
  used to configure the frequency to be used for the continuously reception window if the multicast reception is selected.

4.2.17.1 Set Multicast RXC Configuration

This service can be used to configure the multicast rx channel parameters.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_MCAST_RXC_CONFIG_REQ	Set Multicast RXC Configuration Request
Length	5	5 octets
Payload[0]	Class C Selection	Selection of Class C parameters: 0: unicast (Rx2 parameters) 1: multicast (RXC parameters)
Payload[1]	RXC Data Rate	Data rate to be used if multicast selected (RXC data rate). *Note: refer to the corresponding regional HCI specification,* [4] *for the allowed configuration.*
Payload[2..4]	RXC Frequency	Frequency to be used if multicast selected (RXC frequency). This field is a 24-bit unsigned integer, where the actual channel frequency (in Hz) is 100 × Frequency,

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_SET_MCAST_RXC_CONFIG_RSP	Set Multicast RXC Configuration Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.4.2)

4.2.17.2 Get Multicast RXC Configuration

This service can be used to get the multicast rx channel configuration.

Command Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_MCAST_RXC_CONFIG_REQ	Get Multicast RXC Configuration Request
Length	0	0 octets

Response Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_GET_MCAST__RXC_CONFIG_RSP	Get Multicast RXC Configuration Response
Length	6	5 octets
Payload[0]	Status Byte	see appendix (6.5.4.2)
Payload[1]	Class C Selection	Selection of Class C parameters: 0: unicast (Rx2 parameters) 1: multicast (RXC parameters)
Payload[2]	RXC Data Rate	Data rate to be used if multicast selected (RXC data rate). *Note: refer to the corresponding regional HCI specification, [4] for the allowed configuration.*
Payload[3..5]	RXC Frequency	Frequency to be used if multicast selected (RXC frequency). This field is a 24-bit unsigned integer, where the actual channel frequency (in Hz) is 100 × Frequency,

4.3 Proprietary LoRa® Link Services

The Proprietary LoRa® Link Service Access Point provides functions for configuration and transmission and reception of radio link messages. These services apply only to the proprietary stack.

The radio firmware part operates in Standard Mode, including support for unreliable radio message exchange with address filtering.

Note that the transmission of radio messages is not allowed in "Customer Mode" (see 4.1.7).

4.3.1 Radio Configuration

The radio firmware supports several configurable parameters which are stored in the non-volatile flash memory. The following items can be configured:

Item	Description
Radio Mode	Determines the radio module operation. Limited to **Standard** mode.
Group Address	Used to separate groups of radio modules. This value is compared against the *TxGroupAddress* field of a received radio message to filter radio packets in **Standard** mode (0xFF = BROADCAST address).
Device Address	Used to address a specific radio device. This value is compared against the *TxDeviceAddress* field of a received radio message to filter radio packets in **Standard** mode (0xFFFF = BROADCAST address).
Modulation	0 = LoRa®, 1 = FSK
RF Carrier Frequency	Defines the used radio frequency. See 6.4 for further details.
LoRa® Signal Bandwidth	Defines the LoRa® signal bandwidth 0 = 125 kHz, 1 = 250 kHz, 2 = 500 kHz
LoRa® Spreading Factor	Defines the LoRa® spreading factor 0 – 7 = SF7, 8 = SF8, 9 = SF9, 10 = SF10, 11 = SF11, 12 = SF12
FSK Datarate	Determines the datarate if FSK modulation is enabled 0 = 50000bps
Error Coding	Defines the radio error coding format 0 = 4/5, 1 = 4/5, 2 = 4/6, 3 = 4/7, 4 = 4/8
Power Level (ERP)	Defines the transmit power level from 5 dBm to 20 dBm: 0 – 5 = 5 dBm, 6 = 6 dBm, ... , 20 = 20 dBm
Rx Control	Receiver Control Option: 0 = Receiver off 1 = Receiver always on (except during packet transmission) 2 = Receiver on for limited time defined by Rx Window parameter
Rx Window Time	Configurable time for radio receive mode after radio packet transmission. Note: Rx Window option must be enabled in the *Rx Control* parameter. A value of zero (0) disables the receive mode.

Misc. Options	Bit field to configure further radio firmware options: Bit 0: 0 = standard RF packet output format 1 = extended RF packet output format: attached RSSI, SNR and Timestamp Bit 2: HCI Tx Indication - this message is sent to the host after an RF message was sent over the air. 0 = disabled 1 = enabled Bit 5: 0 = AES Encryption/Decryption off 1 = AES Encryption/Decryption on

4.3.1.1 Get Radio Configuration

This message can be used to read the configuration parameters.

Command Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_GET_RADIO_CONFIG_REQ	Get Radio Configuration Request
Length	0	no payload

Response Message

The response message contains the current radio configuration. The Radio Configuration Field is described in more detail below.

Field	Content	Description
Endpoint ID	RADIOLINK _ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_GET_RADIO_CONFIG_RSP	Get Radio Configuration Response
Length	26	26 octets
Payload[0]	Status Byte	see appendix (6.5.3.2)
Payload[1..25]	Radio Configuration Field	see Radio Configuration Field

4.3.1.2 Set Radio Configuration

This function can be used to change several radio parameters. The function allows to change parameter directly and to save them optionally in the non-volatile flash memory.

Command Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_SET_RADIO_CONFIG_REQ	Set Radio Configuration Request
Length	26	26 octets
Payload[0]	Store NVM Flag 0x00 : change configuration only temporary (RAM) 0x01 : save configuration also in NVM	non-volatile memory flag
Payload[1..25]	Radio Configuration Field	see Radio Configuration Field

Response Message

This message acknowledges the Set Radio Configuration Request message.

Field	Content	Description
Endpoint ID	RADIOLINK _ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_SET_RADIO_CONFIG_RSP	Get Radio Configuration Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.3.2)

4.3.1.3 Radio Configuration Field

The Radio Configuration Field contains the following configurable radio parameters:

Offset	Size	Name	Description
0	1	Radio Mode	0x00 = Standard mode: Device & Group address used for packet filtering
1	1	Group Address	Own group address (0x01 – 0xFE) for packet filtering (0xFF reserved as BROADCAST address)
2	1	Reserved	Reserved
3	2	Device Address	Own device address (0x0001 – 0xFFFE) for packet filtering (0xFFFF reserved as BROADCAST address)
5	2	Reserved	Reserved
7	1	Modulation	0 = LoRa®, 1 = FSK (50000 bps) *Note: refer to the corresponding regional HCI specification,* [4] *for the allowed configuration.*

8	1	RF Carrier Frequency Least Significant Bits	Defines the used radio frequency. See 6.4 for details. *Note: refer to the corresponding regional HCI specification, [4] for the allowed configuration.*
9	1	RF Carrier Frequency Intermediate Bits	Defines the used radio frequency. See 6.4 for details. *Note: refer to the corresponding regional HCI specification, [4] for the allowed configuration.*
10	1	RF Carrier Frequency Most Significant Bits	Defines the used radio frequency. See 6.4 for details. *Note: refer to the corresponding regional HCI specification, [4] for the allowed configuration.*
11	1	LoRa® Signal Bandwidth	0 = 125 kHz, 1 = 250 kHz, 2 = 500 kHz *Note: refer to the corresponding regional HCI specification, [4] for the allowed configuration.*
12	1	LoRa® Spreading Factor	0 – 7 = SF7 8 = SF8 9 = SF9 10 = SF10 11 = SF11 12 = SF12 *Note: refer to chapter 2 for the allowed configuration.*
13	1	Error Coding	0 = 4/5 1 = 4/5 2 = 4/6 3 = 4/7 4 = 4/8
14	1	Power Level	0 – 5 = 5 dBm 6 = 6 dBm 7 = 7 dBm ... 20 = 20 dBm
15	1	Reserved	Reserved
16	1	Rx Control	Receiver Control Option: 0 = Receiver off 1 = Receiver always on (except during packet transmission) 2 = Receiver on for limited time defined by Rx Window parameter
17	2	Rx Window Time	0 = receiver disabled, no Rx Window 1 – 65535 = 1 - 65535 ms
19	1	Reserved	Reserved

20	1	Misc. Options	Bit 0: 0 = standard RF packet output format 1 = extended RF packet output format: attached RSSI, SNR and Timestamp Bit 2: 0 = HCI Tx Indication disabled 1 = HCI Tx Indication enabled Bit 5: 0 = AES Encryption/Decryption off 1 = AES Encryption/Decryption on
21	1	FSK Datarate	0 = 50000 bps *Note: refer to the corresponding regional HCI specification,* [4] *for the allowed configuration.*
22	1	Reserved	Reserved
23	2	Reserved	Reserved

4.3.1.4 Reset Radio Configuration

This message can be used to restore the default radio settings.

Command Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_RESET_RADIO_CONFIG_REQ	Reset Radio Config Request
Length	0	no payload

Response Message

This message acknowledges the Reset Radio Configuration Request message.

Field	Content	Description
Endpoint ID	RADIOLINK _ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_RESET_RADIO_CONFIG_RSP	Reset Radio Config Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.3.2)

4.3.1.5 Default Configuration

The following table lists the default configuration for some parameters (refer to the corresponding regional HCI specification, [4] for the specific default parameters for each region).

Parameter	Value EU868	Value US915	Value AU915
Radio Mode	0 = Standard Mode		
Group Address	0x10		
Device Address	0x1234		
Rx Control	1 = Rx always on		
Rx Window Time	3 s		
Misc. Options	0x01: - extended RF packet output format enabled - HCI Tx Indication disabled - AES Encryption/Decryption off		

4.3.2 AES Key Configuration

The radio firmware can perform an automatic radio packet encryption and decryption (see chapter 4.3.4 for more details).

The implemented cipher is based on the AES 128 bit Counter Mode. The following commands can be used to set and read the required 128 bit AES key.

4.3.2.1 Set AES Key

This message is used to set a new AES key. The key will be stored in the NVM to resist a power cycle.

Command Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_SET_AES_KEY_REQ	Set AES Key Req.
Length	16	16 octets
Payload[0..15]	128 bit AES Key	octet sequence

Response Message

Field	Content	Description
Endpoint ID	RADIOLINK _ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_SET_AES_KEY_RSP	Set AES Key Rsp.
Length	1	1 octets
Payload[0]	Status Byte	see appendix (6.5.3.2)

4.3.2.2 Get AES Key

This message is used to read the configured AES key.

Command Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_GET_AES_KEY_REQ	Get AES Key Req.
Length		no payload

Response Message

Field	Content	Description
Endpoint ID	RADIOLINK _ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_GET_AES_KEY_RSP	Get AES Key Rsp.
Length	1 + 16	17 octets
Payload[0]	Status Byte	see appendix (6.5.3.2)
Payload[1..16]	128 bit AES Key	octet sequence

4.3.3 Unreliable Data Exchange

This service can be used to exchange radio messages in an unreliable way, i.e. it is not guaranteed that a transmitted message will be received on a peer radio device. There is no automatic acknowledgement or retry mechanism implemented combined with this function.

4.3.3.1 Send Unreliable Message

This command can be used to send a radio message either as broadcast message to all other radios in range or to a certain radio device with given address. Depending on the chosen radio settings, the transmission of a single radio message can take several hundred milliseconds. The firmware supports an HCI Tx Indication message which is sent to the host controller when the radio transmission has finished.

Command Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_SEND_U_DATA_REQ	Send unreliable radio message request
Length	N	n octets
Payload	Tx Radio Message Field	see below

Response Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_SEND_U_DATA_RSP	Send unreliable radio message response
Length	1	1 octet
Payload[0]	Status Byte	see appendix (6.5.3.2)

Event Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_U_DATA_TX_IND	Unreliable radio message transmission finished
Length	7	7 octets
Payload[0]	Status Byte	see appendix (6.5.3.2)
Payload[1..2]	Tx Event Counter	Incremented for every Tx event
Payload[3..6]	RF Message Airtime	32-Bit Airtime in milliseconds of transmitted radio message

4.3.3.2 Tx Radio Message Field

The following figure outlines the relationship between the HCI message, sent from the host controller and the radio message, sent from the radio module.

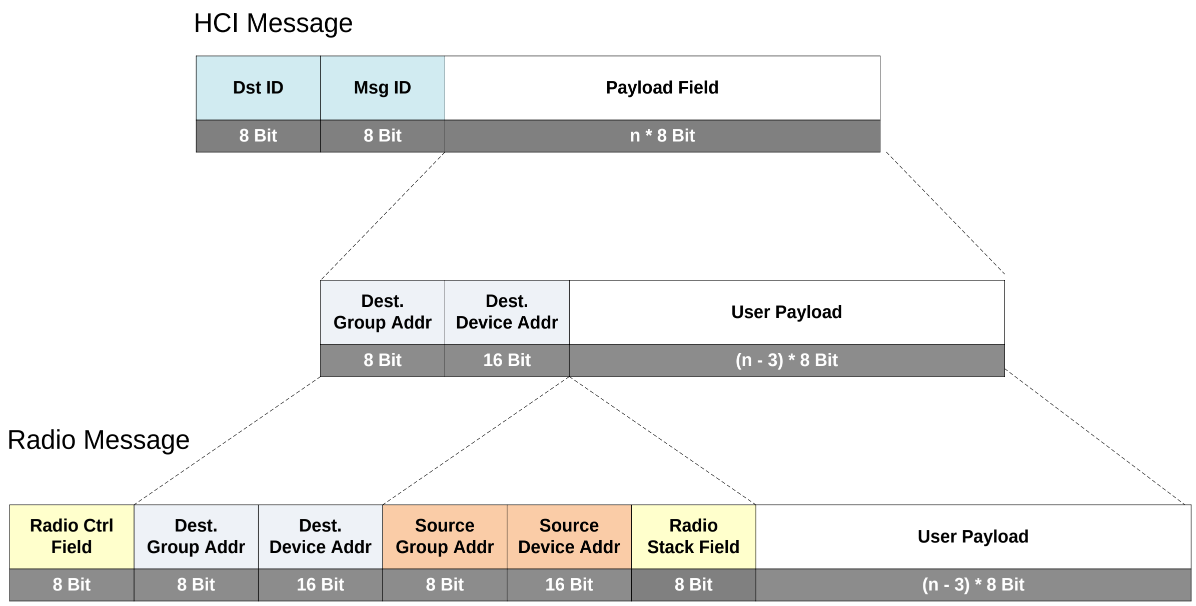

Fig. 4-4: Tx Radio Message and HCI Message

The Radio Ctrl Field (see below), Radio Stack Field and Source Address Fields are automatically added by the firmware itself.

The HCI Payload field content is defined as follows:

Offset	Size	Name	Description
0	1	Dest. Group Address	Destination Group Address (0xFF = BROADCAST) of message receiver
1	2	Dest. Device Address	Destination Device Address (0xFFFF = BROADCAST) of message receiver
3	N	User Payload	N bytes user defined payload with 1 <= N <= N1 N1 = 255 – 8 = 247 bytes (not encrypted data) N1 = 255 – 12 = 243 bytes (encrypted data)

4.3.3.3 Unreliable Radio Message Reception

The radio module is able to receive messages as long as the receiver is enabled. The receive mode is configurable (see Radio Configuration) and can be:

- disabled (off, Rx-Window = 0)
- always on (except during packet transmission)
- enabled for a limited Rx-Window after a transmitted message

While operating in Standard Mode, the received messages are forwarded to the host controller when they contain a BROADCAST address or the specific device address of the receiver.

Event Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_U_DATA_RX_IND	Unreliable message indication
Length	n	n octets
Payload	Rx Radio Message Field	see appendix (6.5.3.2)

4.3.3.4 Rx Radio Message Field

The following figure outlines the relationship between the radio message, received on the radio module and the forwarded HCI message.

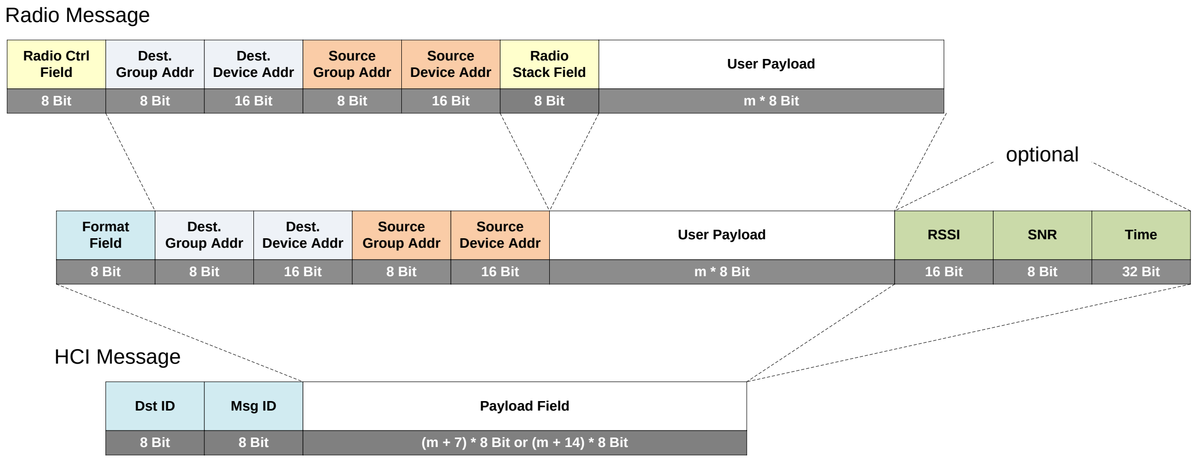

Fig. 4-5: Rx Radio Message and HCI Message

The HCI Payload Field has the following content:

Offset	Size	Name	Description
0	1	Format & Status Field	Defines the packet output format (see chap. HCI Format & Status Field)
1	1	Dest. Group Address	Destination Group Address (0xFF = BROADCAST) of message receiver
2	2	Dest. Device Address	Destination Device Address (0xFFFF = BROADCAST) of message receiver
4	1	Source Group Address	Group Address of message sender
5	2	Source Device Address	Device Address of message sender
7	N	Payload	user defined payload
7+N	2	RSSI (optional)	Received Signal Strength Indicator [dBm], signed integer
9+N	1	SNR (optional)	Signal to Noise Ratio [dB], signed integer
10+N	4	Rx Time (optional)	Timestamp from RTC

4.3.3.5 Radio Control Field

The Radio Control Field in each radio packet has the following meaning:

Bit	Name	Description
0	ACK REQUEST BIT	This bit is set to: "0" : in unconfirmed radio messages
1	Reserved	
2	CIPHER BIT	"1" : Indicates an encrypted radio message
3 - 7	Reserved	

4.3.3.6 HCI Format & Status Field

The HCI Format & Status Field has the following meaning:

Bit	Name	Description
0	EXTENDED_OUTPUT	"0" : standard output format, no attachment "1" : extended output format with attached RSSI, SNR and RTC Timestamp
1 – 4	Reserved	
5	DECRYPTED_DATA	"1" : indicates a decrypted data output
6	DECRYPTION_ERROR	"1" : indicates a decryption error
7	ENCRYPTED_DATA	"1" : indicates encrypted data output

4.3.4 Radio Packet Encryption

The automatic radio packet encryption & decryption can be activated (see chapter 4.3.1) for every unconfirmed radio message.

The implemented cipher is based on the AES Counter Mode algorithm.

The radio packet format for encrypted messages is outlined in the following figure:

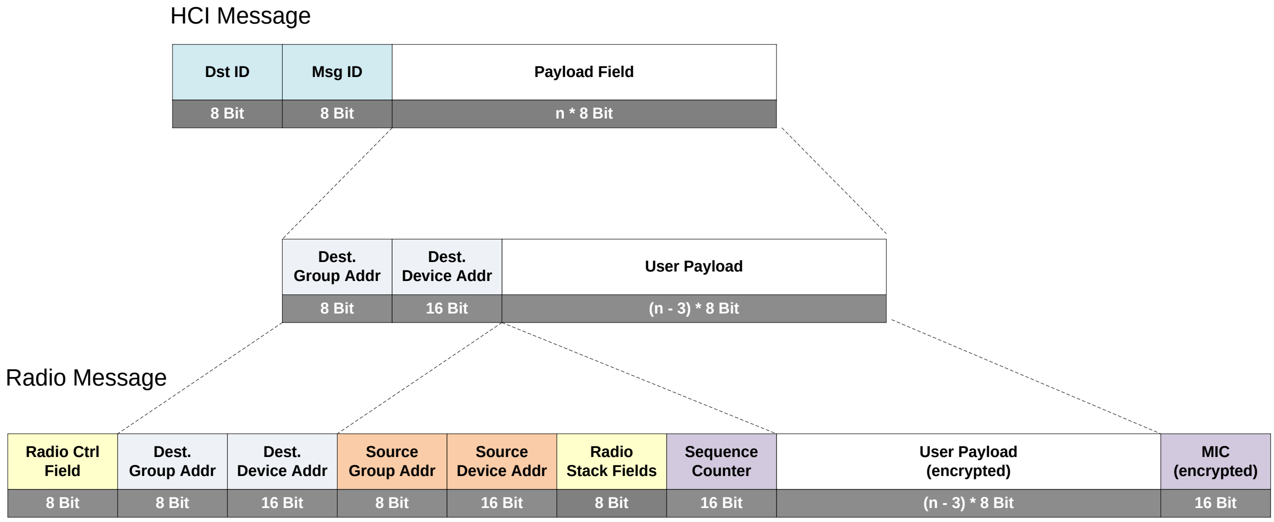

Fig. 4-6: Radio Packet Format for encrypted messages

In addition to the not encrypted message format two new fields are added to the overall packet structure:

- Sequence Counter

  An automatic incrementing 16 Bit counter used as input for the AES 128 bit counter mode encryption

- MIC

  A 16 bit message integrity code, used to verify a successful packet decryption on receiver side

Receiver Side:

On receiver side the following scenarios are possible:

- Received message was successfully decrypted:

  The forwarded HCI message uses the same output format as for not encrypted messages.

- Decryption on receiver side is disabled:

  The forwarded HCI messages includes the sequence counter, encrypted user payload and attached MIC. The HCI Status & format Field indicates that the payload is encrypted.

- Decryption is enabled but a decryption error was detected (MIC error):

The forwarded HCI messages includes the sequence counter, encrypted user payload and attached MIC. The HCI Status & format Field indicates that the payload is encrypted and that a decryption error was detected.

The packet format for those three cases is outlined here:

Successful Decryption

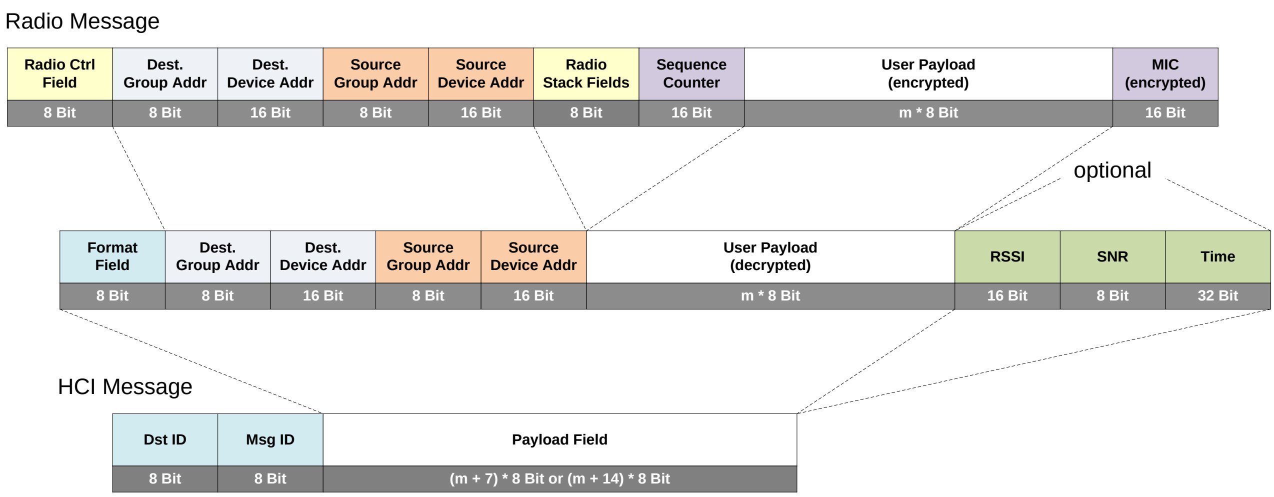

Fig. 4-7: Rx Radio Message and HCI Message (encrypted radio data, decrypted HCI output)

The HCI Payload Field has the following content:

Offset	Size	Name	Description
0	1	Format & Status Field	Defines the packet output format (see chap. HCI Format & Status Field)
1	1	Dest. Group Address	Destination Group Address (0xFF = BROADCAST) of message receiver
2	2	Dest. Device Address	Destination Device Address (0xFFFF = BROADCAST) of message receiver
4	1	Source Group Address	Group Address of message sender
5	2	Source Device Address	Device Address of message sender
7	N	Payload	User defined decrypted payload
7+N	2	RSSI (optional)	Received Signal Strength Indicator [dBm], signed integer
9+N	1	SNR (optional)	Signal to Noise Ratio [dB], signed integer
10+N	4	Rx Time (optional)	Timestamp from RTC

Not Decrypted Output (decryption error or decryption disabled)

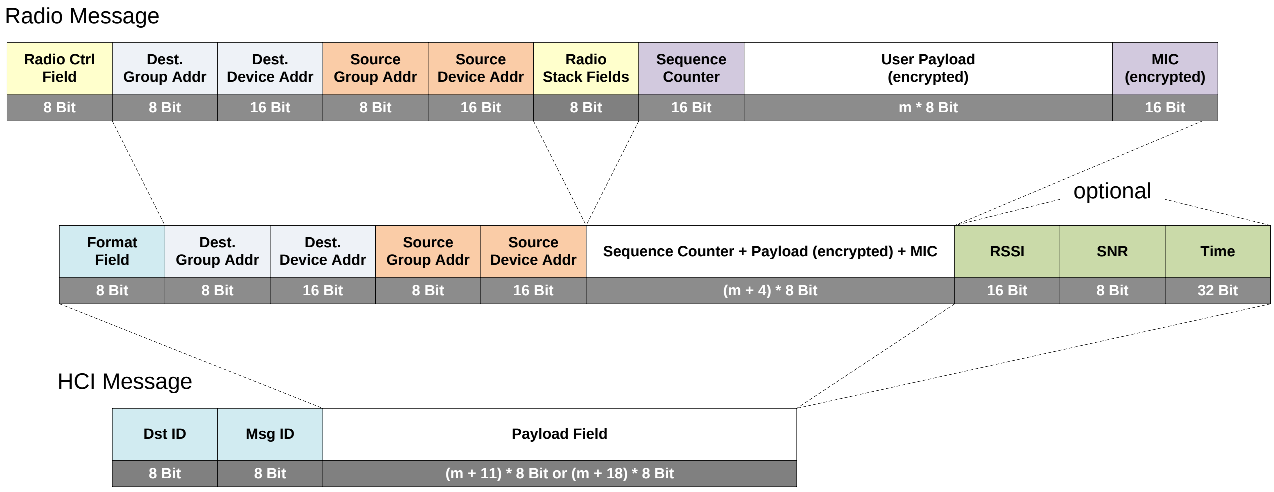

Fig. 4-8: Rx Radio Message and HCI Message (encrypted radio data, not decrypted HCI output)

The HCI Payload Field has the following content:

Offset	Size	Name	Description
0	1	Format & Status Field	Defines the packet output format (see chap. HCI Format & Status Field)
1	1	Dest. Group Address	Destination Group Address (0xFF = BROADCAST) of message receiver
2	2	Dest. Device Address	Destination Device Address (0xFFFF = BROADCAST) of message receiver
4	1	Source Group Address	Group Address of message sender
5	2	Source Device Address	Device Address of message sender
7	2	Sequence Counter	16 bit Sequence Counter
9	N	Payload	User defined **encrypted** payload
9+N	2	MIC	Message Integrity Code
11+N	2	RSSI (optional)	Received Signal Strength Indicator [dBm], signed integer
13+N	1	SNR (optional)	Signal to Noise Ratio [dB], signed integer
14+N	4	Rx Time (optional)	Timestamp from RTC

5. Known Limitations

This chapter lists the current known limitations related to the ProLink LoRaWAN® EndNode Modem firmware:

- No official LoRaWAN® certification for multicast class C, V1.0.4
- Note that a new configuration request may require access to the non-volatile memory, therefore a power loss or HW reset must be avoided in the next 500 ms to ensure that the stored parameters are correct
- Automatic activation of the bootloader via the HCI interface (for future firmware updates) not supported by iM881A-XL
- Activation By Personalization only available for testing purposes

6. Appendix

6.1 System Operation Modes

Index	Description
0	Standard Application Mode / Default Mode
1	Reserved
2	Reserved
3	Customer Mode

6.2 RF Gain Examples

6.2.1 iM880B-L Radio Module configured in EU868 Band[1]

For this example a maximum RF power (limited by the radio module) of 20dBm and a maximum allowed EIRP of 16dBm have been considered.

Max. RF power	Max. allowed EIRP	RF Gain	Max. EIRP	Configured EIRP	Configured TRX power
20dBm	16dBm	0dBd	16dBm	16dBm	14dBm
20dBm	16dBm	+6dBd	16dBm	16dBm	8dBm
20dBm	16dBm	-6dBd	16dBm	16dBm	20dBm

Table. 6-1: Example for RF Gain settings - iM880B-L & EU868

6.2.2 iM880B-L Radio Module configured in IN865 Band[2]

For this example a maximum RF power (limited by the radio module) of 20dBm and a maximum allowed EIRP of 30dBm have been considered.

Max. RF power	Max. allowed EIRP	RF Gain	Max. EIRP	Configured EIRP	Configured TRX power
20dBm	30dBm	0dBd	22dBm	22dBm	20dBm
20dBm	30dBm	+6dBd	28dBm	28dBm	20dBm
20dBm	30dBm	-6dBd	16dBm	16dBm	20dBm

Table. 6-2: Example for RF Gain settings - iM880B-L & IN865

[1] AS923 and RU868 channel plans similar to EU868

[2] US915 and AU915 channel plans similar to IN865

6.2.3 iM881A Radio Module configured in EU868 Band

For this example a maximum RF power (limited by the radio module) of 14dBm and a maximum allowed EIRP of 16dBm have been considered.

Max. RF power	Max. allowed EIRP	RF Gain	Max. EIRP	Configured EIRP	Configured TRX power
14dBm	16dBm	0dBd	16dBm	16dBm	14dBm
14dBm	16dBm	+6dBd	16dBm	16dBm	8dBm
14dBm	16dBm	-6dBd	10dBm	10dBm	14dBm

Table. 6-3: Example for RF Gain settings - iM881A & EU868

6.2.4 iM881A Radio Module configured in IN865 Band

For this example a maximum RF power (limited by the radio module) of 14dBm and a maximum allowed EIRP of 30dBm have been considered.

Max. RF power	Max. allowed EIRP	RF Gain	Max. EIRP	Configured EIRP	Configured TRX power
14dBm	30dBm	0dBd	16dBm	16dBm	14dBm
14dBm	30dBm	+6dBd	22dBm	22dBm	14dBm
14dBm	30dBm	-6dBd	10dBm	10dBm	14dBm

Table. 6-4: Example for RF Gain settings - iM881A & IN865

6.3 LoRaWAN Sequence Charts

This chapter contains exemplary sequence charts which show the typical behavior of the LoRaWAN stack.

6.3.1 Join Procedure

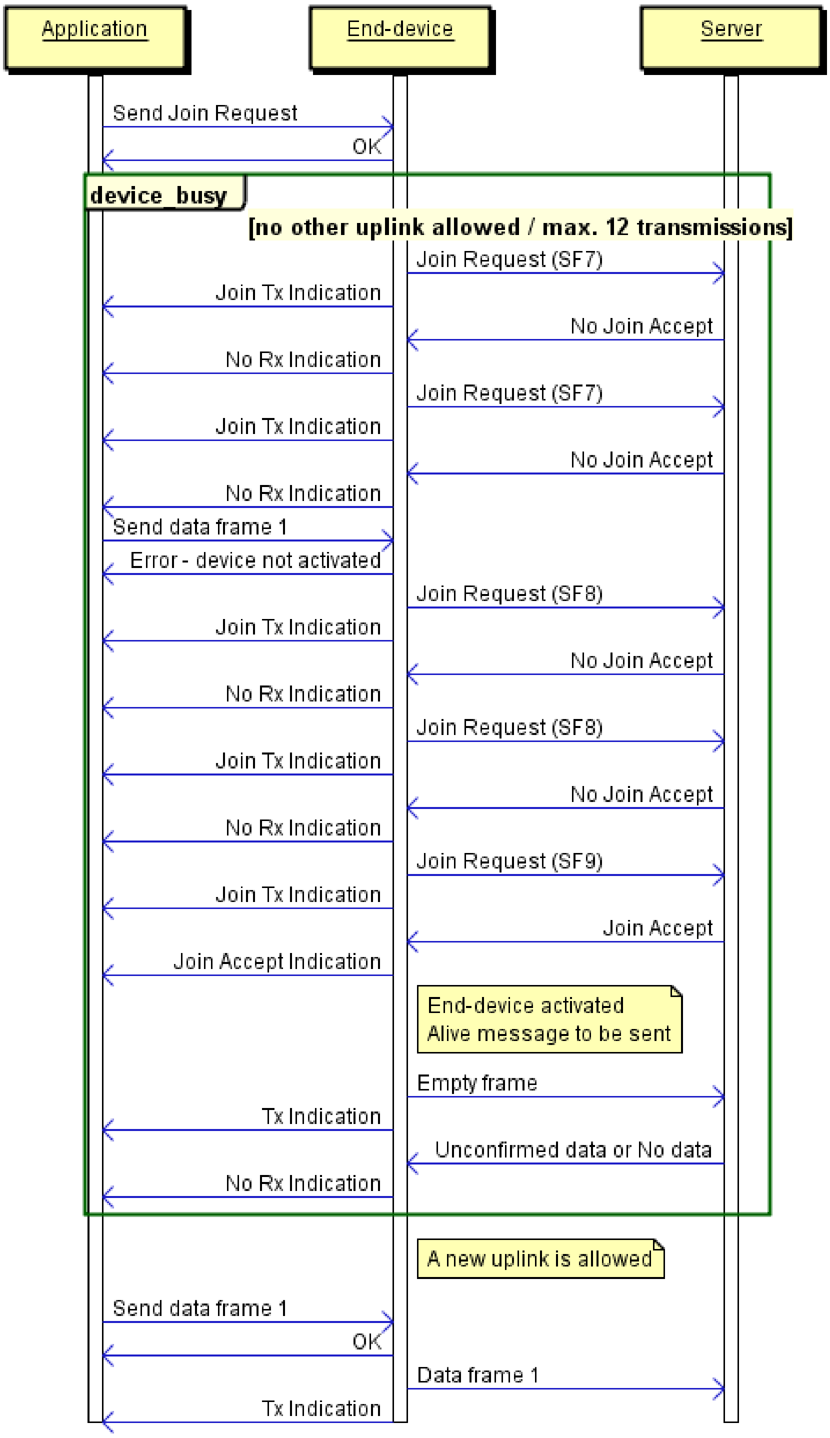

Fig. 6-1: Sequence chart - Join procedure example (ADR enabled/SF7BW125 configured)

6.3.2 Unconfirmed Data Retransmission

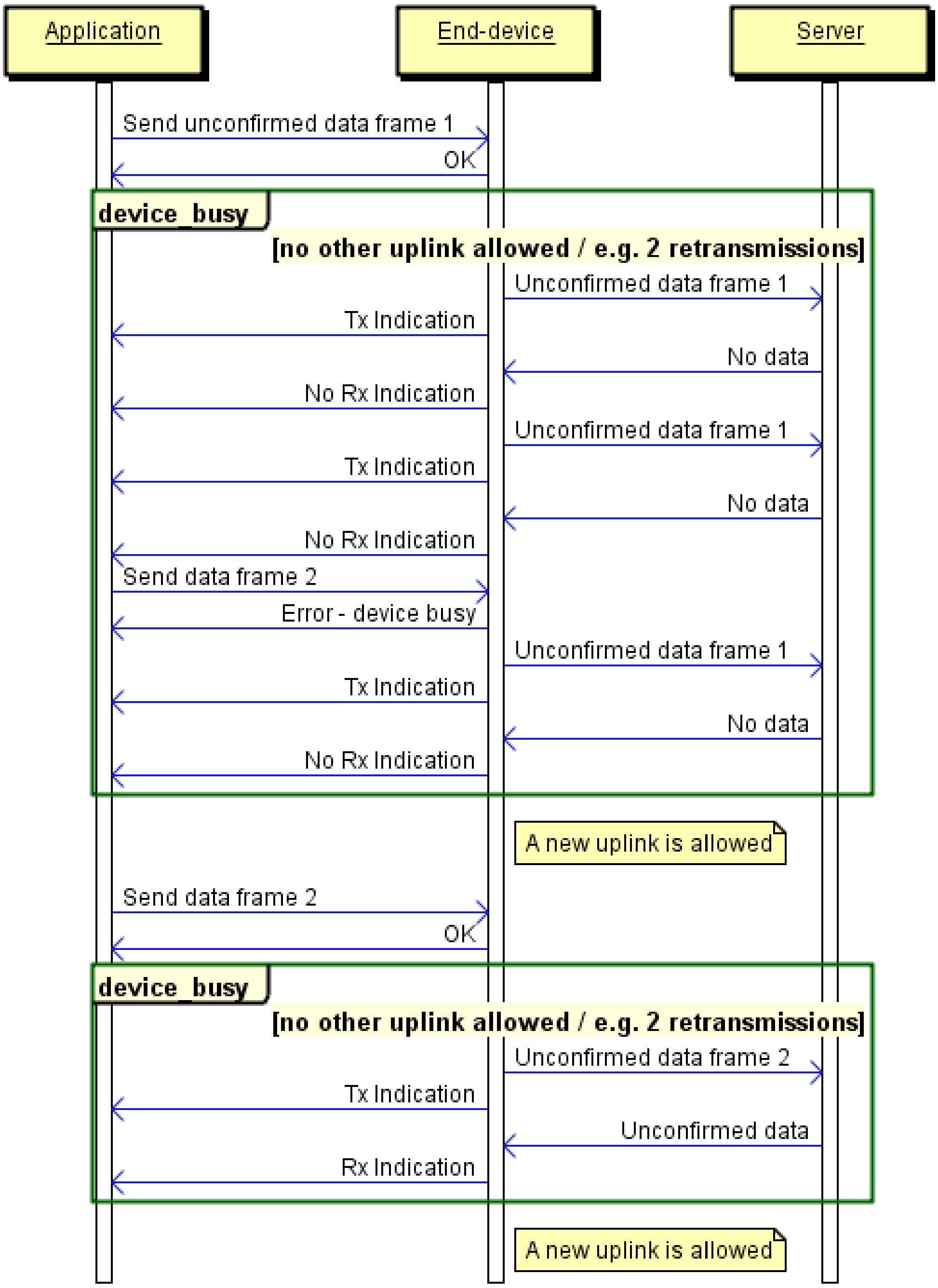

Fig. 6-2: Sequence chart - Unconfirmed data retransmission example (NbTrans=2)

6.3.3 Confirmed Data Retransmission

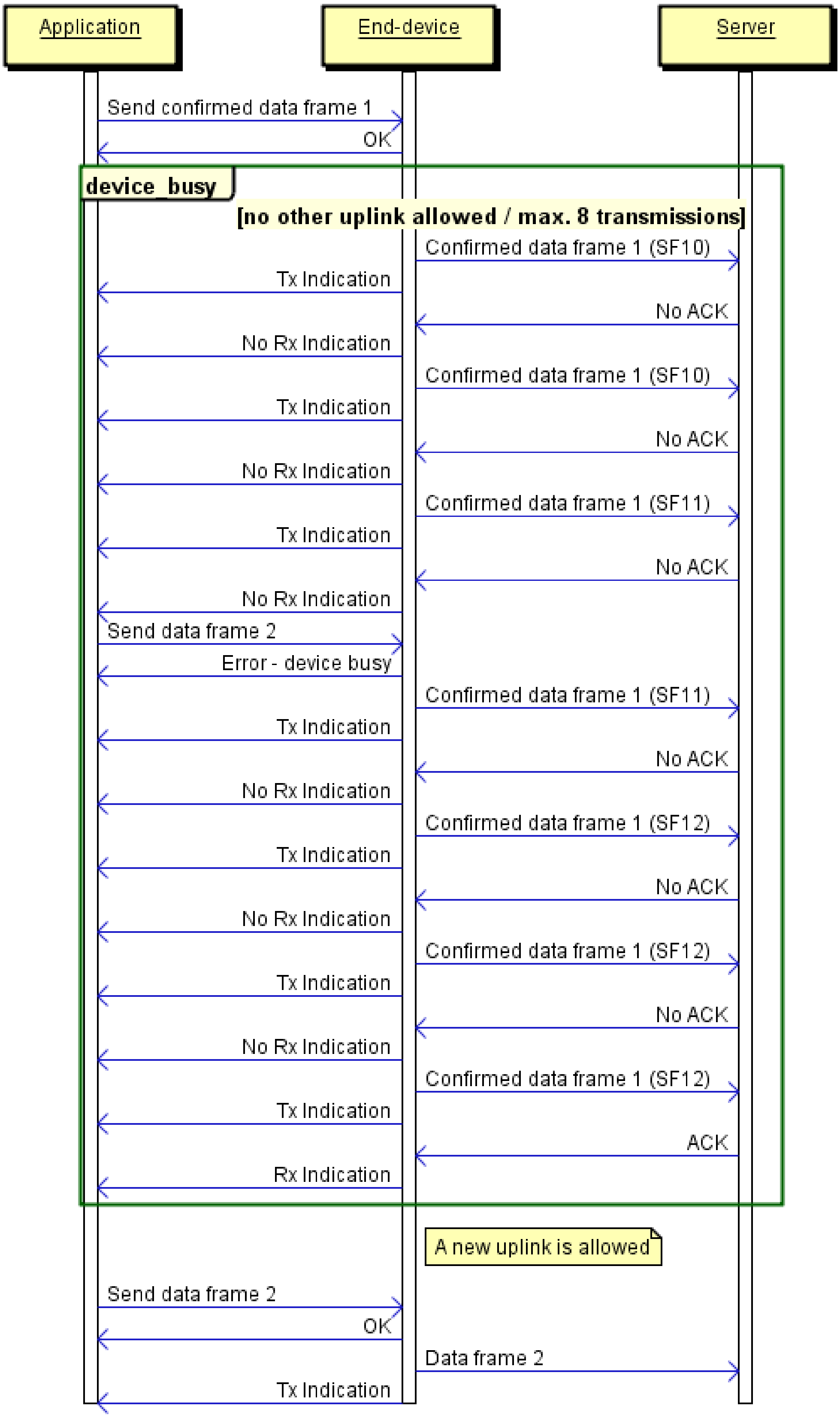

Fig. 6-3: Sequence chart - Retransmission procedure example (NbTrans=1)

6.3.4 Duty Cycle

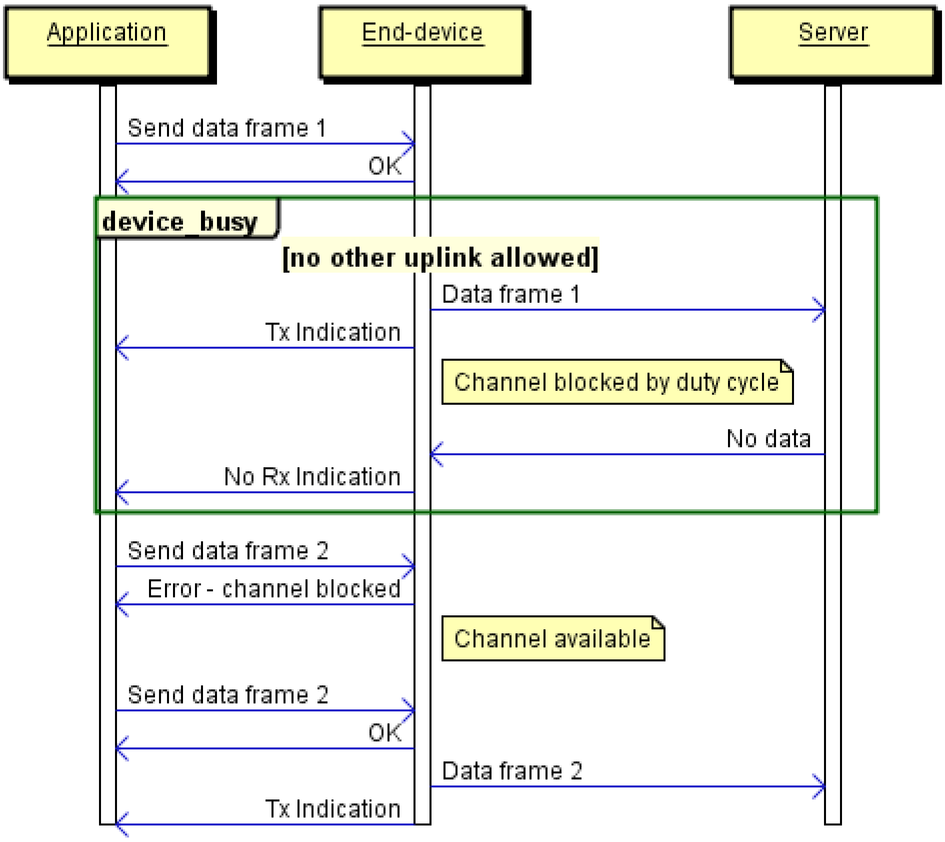

Fig. 6-4: Sequence chart - Duty Cycle

6.3.5 Message Acknowledge Procedure

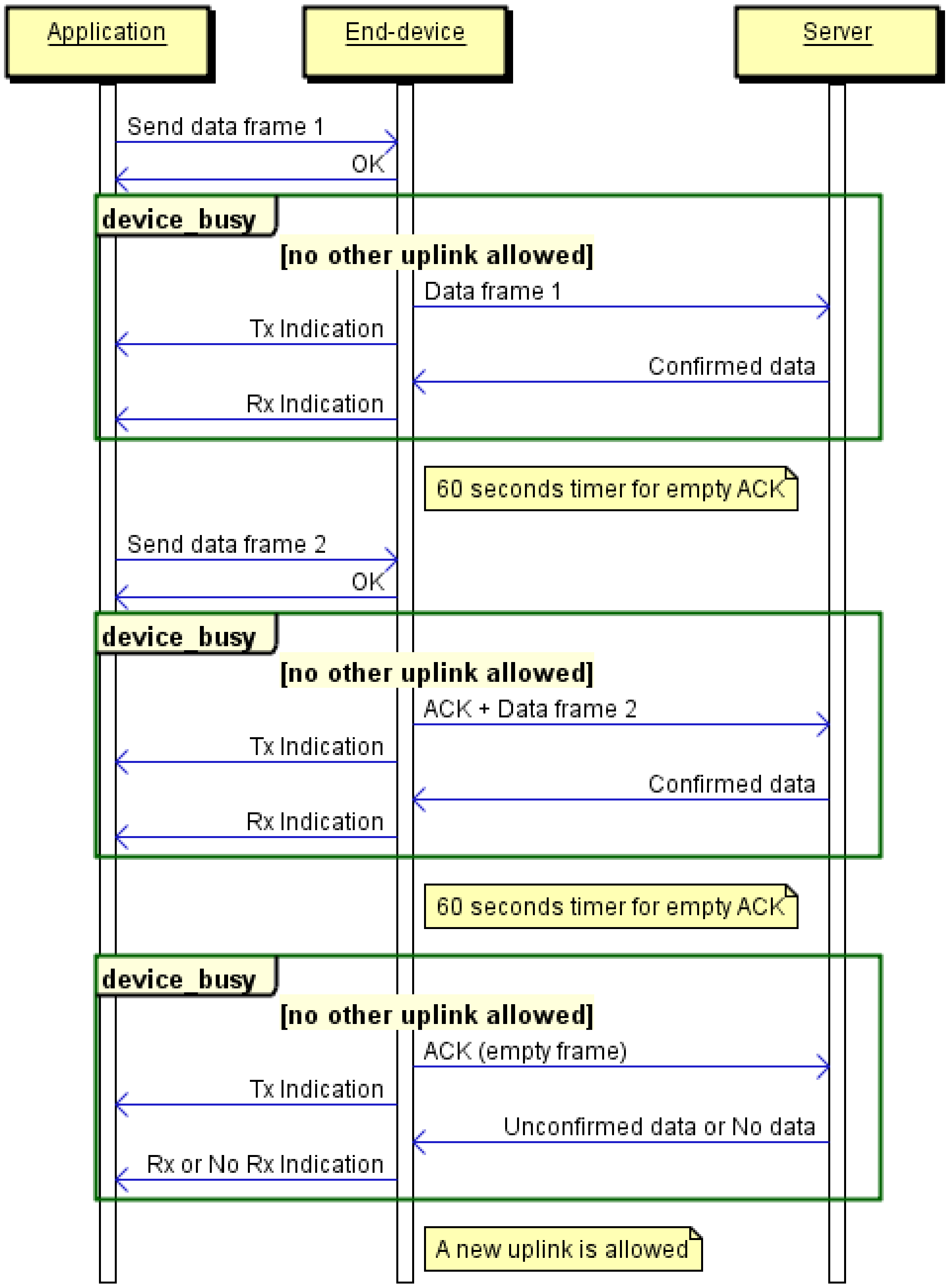

Fig. 6-5: Sequence chart - Acknowledgement procedure (Class A)

6.3.6 Frame Pending Bit

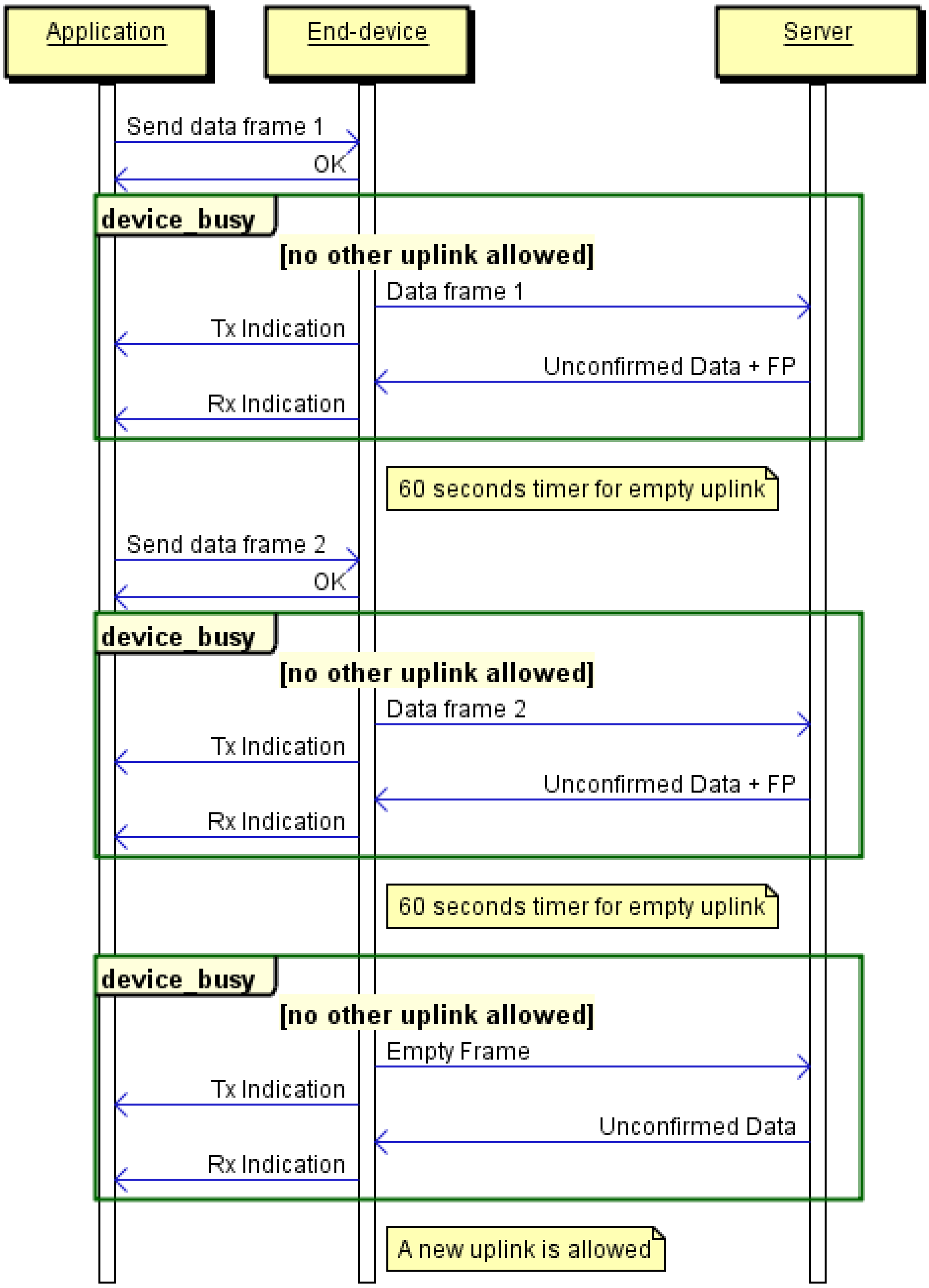

Fig. 6-6: Sequence chart - Frame pending bit

6.3.7 MAC Commands

6.3.7.1 MAC Commands - Piggybacked in Header

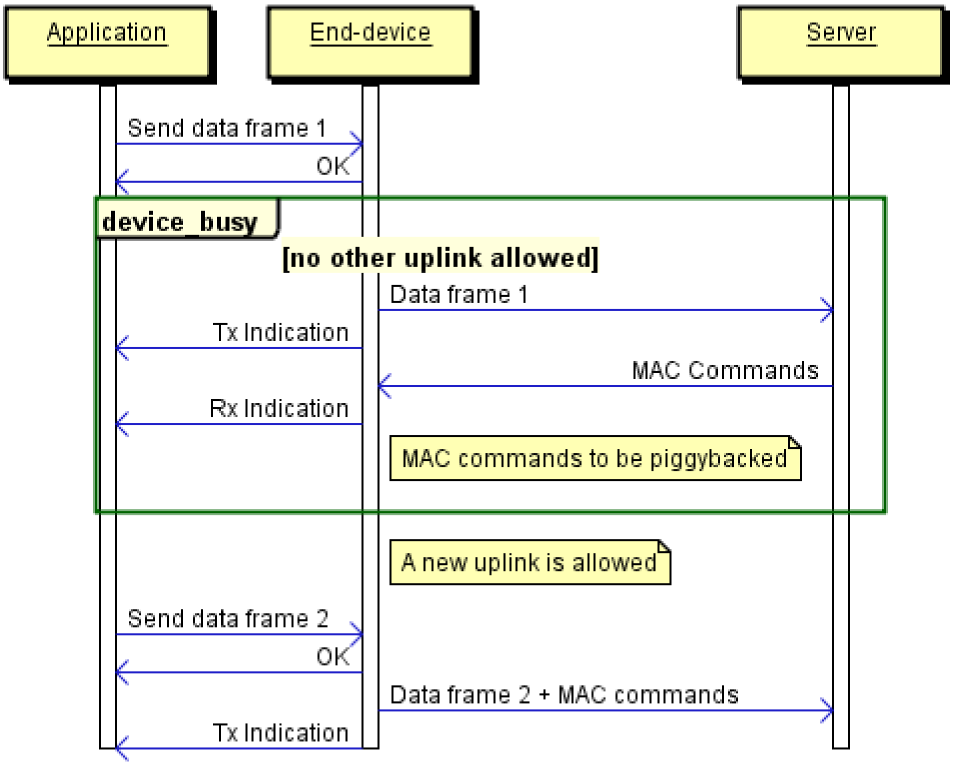

Fig. 6-7: Sequence chart - MAC Commands (piggybacked in header)

6.3.7.2 MAC Commands - Port 0

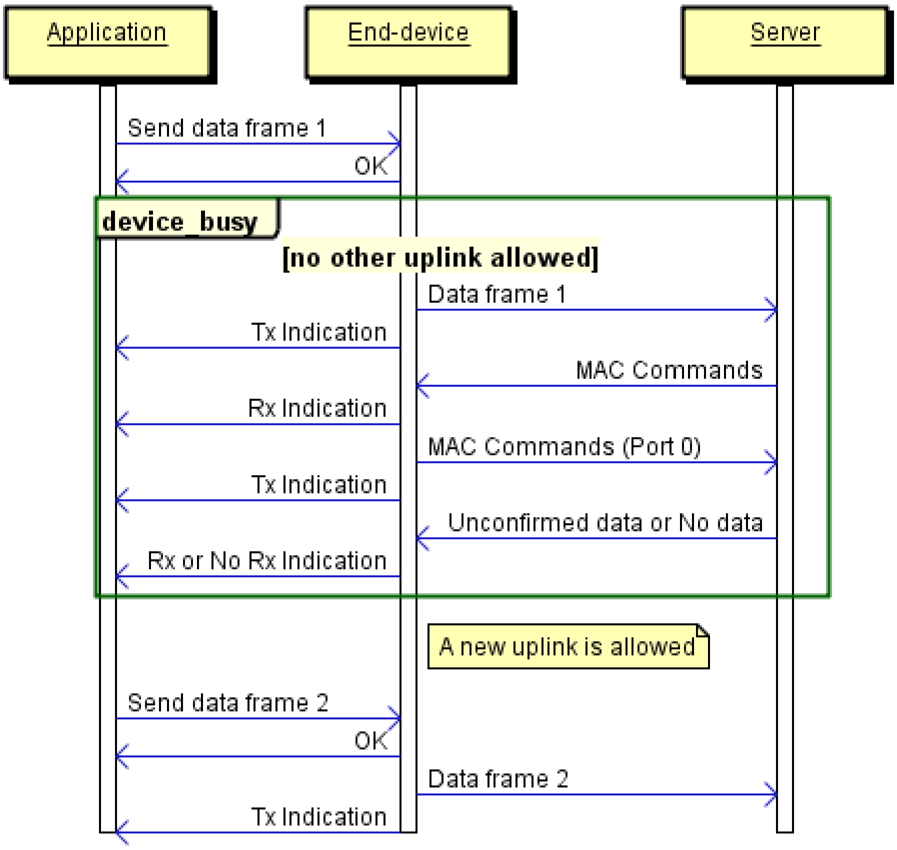

Fig. 6-8: Sequence chart - MAC Commands (using port 0)

6.4 Frequency Setting

The WiMOD radio modules use a 32 MHz crystal for its RF oscillator. The carrier frequency f_{RF} is given by:

$$f_{RF} = f_{STEP} * F_{rf}[23,0],$$

where F_{rf} is a 24 bit register value of Sx1272 and the frequency synthesizer step given by:

$$f_{STEP} = 32 \text{ MHz} / 2^{19}$$

⇨ $F_{rf}[23,0] = \text{floor}(f_{RF} / f_{STEP})$

6.5 List of Constants

6.5.1 List of Endpoint Identifier

Name	Value
DEVMGMT_ID	0x01
RADIOLINK_ID	0x03
LORAWAN_ID	0x10

6.5.2 Device Management Endpoint Identifier

6.5.2.1 Device Management Endpoint Message Identifier

Name	Value
DEVMGMT_MSG_PING_REQ	0x01
DEVMGMT_MSG_PING_RSP	0x02
DEVMGMT_MSG_GET_DEVICE_INFO_REQ	0x03
DEVMGMT_MSG_GET_DEVICE_INFO_RSP	0x04
DEVMGMT_MSG_GET_FW_INFO_REQ	0x05
DEVMGMT_MSG_GET_FW_INFO_RSP	0x06
DEVMGMT_MSG_RESET_REQ	0x07
DEVMGMT_MSG_RESET_RSP	0x08
DEVMGMT_MSG_SET_OPMODE_REQ	0x09
DEVMGMT_MSG_SET_OPMODE_RSP	0x0A
DEVMGMT_MSG_GET_OPMODE_REQ	0x0B
DEVMGMT_MSG_GET_OPMODE_RSP	0x0C
DEVMGMT_MSG_SET_RTC_REQ	0x0D

DEVMGMT_MSG_SET_RTC_RSP	0x0E
DEVMGMT_MSG_GET_RTC_REQ	0x0F
DEVMGMT_MSG_GET_RTC_RSP	0x10
DEVMGMT_MSG_GET_DEVICE_STATUS_REQ	0x17
DEVMGMT_MSG_GET_DEVICE_STATUS_RSP	0x18
DEVMGMT_MSG_POWER_UP_IND	0x20
DEVMGMT_MSG_SET_DEVICE_CONFIG_REQ	0x25
DEVMGMT_MSG_SET_DEVICE_CONFIG_RSP	0x26
DEVMGMT_MSG_GET_DEVICE_CONFIG_REQ	0x27
DEVMGMT_MSG_GET_DEVICE_CONFIG_RSP	0x28
DEVMGMT_MSG_RESET_DEVICE_CONFIG_REQ	0x29
DEVMGMT_MSG_RESET_DEVICE_CONFIG_RSP	0x2A
DEVMGMT_MSG_SET_RTC_ALARM_REQ	0x31
DEVMGMT_MSG_SET_RTC_ALARM_RSP	0x32
DEVMGMT_MSG_CLEAR_RTC_ALARM_REQ	0x33
DEVMGMT_MSG_CLEAR_RTC_ALARM_RSP	0x34
DEVMGMT_MSG_GET_RTC_ALARM_REQ	0x35
DEVMGMT_MSG_GET_RTC_ALARM_RSP	0x36
DEVMGMT_MSG_RTC_ALARM_IND	0x38
DEVMGMT_MSG_SET_RADIO_STACK_REQ	0x39
DEVMGMT_MSG_SET_RADIO_STACK_RSP	0x3A
DEVMGMT_MSG_GET_RADIO_STACK_REQ	0x3B
DEVMGMT_MSG_GET_RADIO_STACK_RSP	0x3C
DEVMGMT_MSG_SET_HCI_CFG_REQ	0x41
DEVMGMT_MSG_SET_HCI_CFG_RSP	0x42
DEVMGMT_MSG_GET_HCI_CFG_REQ	0x43
DEVMGMT_MSG_GET_HCI_CFG_RSP	0x44

6.5.2.2 Device Management Endpoint Status Byte

Name	Value	Description
DEVMGMT_STATUS_OK	0x00	Operation successful
DEVMGMT_STATUS_ERROR	0x01	Operation failed
DEVMGMT_STATUS_CMD_NOT_SUPPORTED	0x02	Command is not supported
DEVMGMT_STATUS_WRONG_PARAMETER	0x03	HCI message contains wrong parameter

6.5.3 Radio Link Endpoint Identifier

6.5.3.1 Radio Link Endpoint Message Identifier

Name	Value
RADIOLINK_MSG_SEND_U_DATA_REQ	0x01
RADIOLINK_MSG_SEND_U_DATA_RSP	0x02
RADIOLINK_MSG_U_DATA_RX_IND	0x04
RADIOLINK_MSG_U_DATA_TX_IND	0x06
RADIOLINK_MSG_SET_RADIO_CONFIG_REQ	0x17
RADIOLINK_MSG_SET_RADIO_CONFIG_RSP	0x18
RADIOLINK_MSG_GET_RADIO_CONFIG_REQ	0x19
RADIOLINK_MSG_GET_RADIO_CONFIG_RSP	0x1A
RADIOLINK_MSG_RESET_RADIO_CONFIG_REQ	0x1B
RADIOLINK_MSG_RESET_RADIO_CONFIG_RSP	0x1C
RADIOLINK_MSG_SET_AES_KEY_REQ	0x21
RADIOLINK_MSG_SET_AES_KEY_RSP	0x22
RADIOLINK_MSG_GET_AES_KEY_REQ	0x23
RADIOLINK_MSG_GET_AES_KEY_RSP	0x24

6.5.3.2 Radio Link Endpoint Status Byte

Name	Value	Description
RADIOLINK_STATUS_OK	0x00	Operation successful
RADIOLINK_STATUS_ERROR	0x01	Operation failed
RADOLINK_STATUS_CMD_NOT_SUPPORTED	0x02	Command is not supported (check system operation mode)
RADIOLINK_STATUS_WRONG_PARAMETER	0x03	HCI message contains wrong parameter
RADIOLINK_STATUS_WRONG_RADIO_MODE	0x04	Module operates in wrong radio mode
RADIOLINK_STATUS_BUFFER_FULL	0x07	No buffer for radio transmission available
RADIOLINK_STATUS_LENGTH_ERROR	0x08	Radio packet length invalid

6.5.4 LoRaWAN® Endpoint Identifier

6.5.4.1 LoRaWAN® Endpoint Message Identifier

Name	Value
LORAWAN_MSG_ACTIVATE_DEVICE_REQ	0x01
LORAWAN_MSG_ACTIVATE_DEVICE_RSP	0x02
LORAWAN_MSG_SET_JOIN_PARAM_REQ	0x05
LORAWAN_MSG_SET_JOIN_PARAM_RSP	0x06
LORAWAN_MSG_JOIN_NETWORK_REQ	0x09
LORAWAN_MSG_JOIN_NETWORK_RSP	0x0A
LORAWAN_MSG_JOIN_NETWORK_TX_IND	0x0B
LORAWAN_MSG_JOIN_NETWORK_IND	0x0C
LORAWAN_MSG_SEND_UDATA_REQ	0x0D
LORAWAN_MSG_SEND_UDATA_RSP	0x0E
LORAWAN_MSG_SEND_UDATA_TX_IND	0x0F
LORAWAN_MSG_RECV_UDATA_IND	0x10
LORAWAN_MSG_SEND_CDATA_REQ	0x11
LORAWAN_MSG_SEND_CDATA_RSP	0x12
LORAWAN_MSG_SEND_CDATA_TX_IND	0x13
LORAWAN_MSG_RECV_CDATA_IND	0x14
LORAWAN_MSG_RECV_ACK_IND	0x15
LORAWAN_MSG_RECV_NO_DATA_IND	0x16
LORAWAN_MSG_SET_RSTACK_CONFIG_REQ	0x19
LORAWAN_MSG_SET_RSTACK_CONFIG_RSP	0x1A
LORAWAN_MSG_GET_RSTACK_CONFIG_REQ	0x1B
LORAWAN_MSG_GET_RSTACK_CONFIG_RSP	0x1C
LORAWAN_MSG_REACTIVATE_DEVICE_REQ	0x1D
LORAWAN_MSG_REACTIVATE_DEVICE_RSP	0x1E
LORAWAN_MSG_DEACTIVATE_DEVICE_REQ	0x21
LORAWAN_MSG_DEACTIVATE_DEVICE_RSP	0x22
LORAWAN_MSG_FACTORY_RESET_REQ	0x23
LORAWAN_MSG_FACTORY_RESET_RSP	0x24
LORAWAN_MSG_SET_DEVICE_EUI_REQ	0x25
LORAWAN_MSG_SET_DEVICE_EUI_RSP	0x26
LORAWAN_MSG_GET_DEVICE_EUI_REQ	0x27
LORAWAN_MSG_GET_DEVICE_EUI_RSP	0x28

LORAWAN_MSG_GET_NWK_STATUS_REQ	0x29
LORAWAN_MSG_GET_NWK_STATUS_RSP	0x2A
LORAWAN_MSG_SEND_MAC_CMD_REQ	0x2B
LORAWAN_MSG_SEND_MAC_CMD_RSP	0x2C
LORAWAN_MSG_RECV_MAC_CMD_IND	0x2D
LORAWAN_MSG_SET_BATTERY_LEVEL_REQ	0x2E
LORAWAN_MSG_SET_BATTERY_LEVEL_RSP	0x2F
LORAWAN_MSG_SET_CUSTOM_CFG_REQ	0x31
LORAWAN_MSG_SET_CUSTOM_CFG_RSP	0x32
LORAWAN_MSG_GET_CUSTOM_CFG_REQ	0x33
LORAWAN_MSG_GET_CUSTOM_CFG_RSP	0x34
LORAWAN_MSG_GET_SUPPORTED_BANDS_REQ	0x35
LORAWAN_MSG_GET_SUPPORTED_BANDS_RSP	0x36
LORAWAN_MSG_LINK_DISCONNECT_IND	0x40
LORAWAN_MSG_SET_MCAST_CONFIG_REQ	0x41
LORAWAN_MSG_SET_MCAST_CONFIG_RSP	0x42
LORAWAN_MSG_GET_MCAST_CONFIG_REQ	0x43
LORAWAN_MSG_GET_MCAST_CONFIG_RSP	0x44
LORAWAN_MSG_DEL_MCAST_CONFIG_REQ	0x45
LORAWAN_MSG_DEL_MCAST_CONFIG_RSP	0x46
LORAWAN_MSG_RECV_MCAST_DATA_IND	0x48
LORAWAN_MSG_RECV_MCAST_NO_DATA_IND	0x4A
LORAWAN_MSG_SET_MCAST_RXC_CONFIG_REQ	0x4B
LORAWAN_MSG_SET_MCAST_RXC_CONFIG_RSP	0x4C
LORAWAN_MSG_GET_MCAST_RXC_CONFIG_REQ	0x4D
LORAWAN_MSG_GET_MCAST_RXC_CONFIG_RSP	0x4E
LORAWAN_MSG_DEVNONCE_RESET_IND	0x60
LORAWAN_MSG_SET_DEVNONCE_REQ	0x61
LORAWAN_MSG_SET_DEVNONCE_RSP	0x62
LORAWAN_MSG_GET_DEVNONCE_REQ	0x63
LORAWAN_MSG_GET_DEVNONCE_RSP	0x64
LORAWAN_MSG_SET_JOINNONCE_REQ	0x65
LORAWAN_MSG_SET_JOINNONCE_RSP	0x66
LORAWAN_MSG_GET_JOINNONCE_REQ	0x67
LORAWAN_MSG_GET_JOINNONCE_RSP	0x68
LORAWAN_MSG_SEND_DEVICETIMEREQ_REQ	0x71

LORAWAN_MSG_SEND_DEVICETIMEREQ_RSP	0x72
LORAWAN_MSG_DEVICETIMEANS_IND	0x74

6.5.4.2 LoRaWAN® Endpoint Status Byte

Name	Value	Description
LORAWAN_STATUS_OK	0x00	Operation successful
LORAWAN_STATUS_ERROR	0x01	Operation failed
LORAWAN_STATUS_CMD_NOT_SUPPORTED	0x02	Command is not supported
LORAWAN_STATUS_WRONG_PARAMETER	0x03	HCI message contains wrong parameter
LORAWAN_STATUS_WRONG_DEVICE_MODE	0x04	Stack is running in a wrong mode
LORAWAN_STATUS_DEVICE_NOT_ACTIVATED	0x05	Device is not activated
LORAWAN_STATUS_DEVICE_BUSY	0x06	Device is busy, command rejected
LORAWAN_STATUS_QUEUE_FULL	0x07	Message queue is full, command rejected
LORAWAN_STATUS_LENGTH_ERROR	0x08	HCI message length is invalid or radio payload size is too large
LORAWAN_STATUS_NO_FACTORY_SETTINGS	0x09	Factory Settings EEPROM block missing
LORAWAN_STATUS_CHANNEL_BLOCKED	0x0A	Channel blocked by Duty Cycle
LORAWAN_STATUS_CHANNEL_NOT_AVAILABLE	0x0B	No channel available (e.g. no channel defined for the configured spreading factor)

6.6 Example Code for Host Controller

6.6.1 Example Application

```
//------------------------------------------------------------------------------
-----
//
//  File:       main.cpp
//
//  Abstract:   main module
//
//  Version:    0.1
//
//  Date:       18.05.2016
//
//  Disclaimer:This example code is provided by IMST GmbH on an "AS IS"
//              basis without any warranties.
//
//------------------------------------------------------------------------------
-----

//------------------------------------------------------------------------------
-----
//
//  Include Files
//
//------------------------------------------------------------------------------
-----

#include "WiMOD_LoRaWAN_API.h"
#include <conio.h>
#include <stdio.h>

//------------------------------------------------------------------------------
-----
//
//  Declarations
//
//------------------------------------------------------------------------------
-----

// forward declarations
static void     ShowMenu();
static void     Ping();
static void     SendUData();
static void     SendCData();

//------------------------------------------------------------------------------
-----
//
//  Section Code
//
//------------------------------------------------------------------------------
-----

//------------------------------------------------------------------------------
-----
//
// main
```

```
//
//------------------------------------------------------------------------------
int
main(int argc, char *argv[])
{
    bool run = true;

    // show menu
    ShowMenu();

    // init interface
    WiMOD_LoRaWAN_Init("COM128");

    // main loop
    while(run)
    {
        // handle receiver process
        WiMOD_LoRaWAN_Process();

        // keyboard pressed ?
        if(kbhit())
        {
            // get command
            char cmd = getch();

            // handle commands
            switch(cmd)
            {
                case    'e':
                case    'x':
                        run = false;
                        break;

                case    'p':
                        // ping device
                        Ping();
                        break;

                case    'u':
                        // send u-data
                        SendUData();
                        break;

                case    'c':
                        // send c-data
                        SendCData();
                        break;

                case    ' ':
                        ShowMenu();
                        break;
            }
        }
    }
    return 1;
}
//------------------------------------------------------------------------------
//
//  ShowMenu
```

```
//
//   @brief: show main menu
//
//----------------------------------------------------------------------------
-----
void
ShowMenu()
{
    printf("\n\r");
    printf("------------------------------\n\r");
    printf("[SPACE] : show this menu\n\r");
    printf("[p]     : ping device\n\r");
    printf("[u]     : send unconfirmed radio message\n\r");
    printf("[c]     : send confirmed radio message\n\r");
    printf("[e|x]   : exit program\n\r");
    printf("\n\r-> enter command: ");

}
//----------------------------------------------------------------------------
-----
//
//   Ping
//
//   @brief: ping device
//
//----------------------------------------------------------------------------
-----
void
Ping()
{
    printf("Ping Device\n\r");

    WiMOD_LoRaWAN_SendPing();
}
//----------------------------------------------------------------------------
-----
//
//   SendUData
//
//   @brief: send unconfirmed radio message
//
//----------------------------------------------------------------------------
-----
void
SendUData()
{
    // port 0x21
    UINT8 port = 0x21;

    UINT8 data[4];

    data[0] = 0x01;
    data[1] = 0x02;
    data[2] = 0x03;
    data[3] = 0x04;

    // send unconfirmed radio message
    WiMOD_LoRaWAN_SendURadioData(port, data, 4);
}
//--------------------------------------------------------------------------
-------
```

```
//
//  SendCData
//
//  @brief: send confirmed radio message
//
//------------------------------------------------------------------------------
void
SendCData()
{
    // port 0x21
    UINT8 port = 0x23;

    UINT8 data[6];

    data[0] = 0x0A;
    data[1] = 0x0B;
    data[2] = 0x0C;
    data[3] = 0x0D;
    data[4] = 0x0E;
    data[5] = 0x0F;

    // send unconfirmed radio message
    WiMOD_LoRaWAN_SendCRadioData(port, data, 6);
}
//------------------------------------------------------------------------------
// end of file
//------------------------------------------------------------------------------
```

6.6.2 LoRaWAN HCI API Layer

```
//------------------------------------------------------------------------------
//
//  File:       WiMOD_LoRaWAN_API.h
//
//  Abstract:   API Layer of LoRaWAN Host Controller Interface
//
//  Version:    0.1
//
//  Date:       18.05.2016
//
//  Disclaimer: This example code is provided by IMST GmbH on an "AS IS"
//              basis without any warranties.
//
//------------------------------------------------------------------------------

#ifndef WIMOD_LORAWAN_API_H
#define WIMOD_LORAWAN_API_H

//------------------------------------------------------------------------------
//
//  Include Files
//
```

```
//------------------------------------------------------------------------------
-----

#include <stdint.h>

//------------------------------------------------------------------------------
-----
//
//   General Declarations
//
//------------------------------------------------------------------------------
-----

typedef uint8_t     UINT8;
typedef uint16_t    UINT16;

//------------------------------------------------------------------------------
-----
//
//   Endpoint (SAP) Identifier
//
//------------------------------------------------------------------------------
-----

#define DEVMGMT_SAP_ID                      0x01
#define LORAWAN_SAP_ID                      0x10

//------------------------------------------------------------------------------
-----
//
//   Device Management SAP Message Identifier
//
//------------------------------------------------------------------------------
-----

#define DEVMGMT_MSG_PING_REQ                0x01
#define DEVMGMT_MSG_PING_RSP                0x02

//------------------------------------------------------------------------------
-----
//
//   LoRaWAN SAP Message Identifier
//
//------------------------------------------------------------------------------
-----

#define LORAWAN_MSG_SEND_UDATA_REQ          0x0D
#define LORAWAN_MSG_SEND_UDATA_RSP          0x0E
#define LORAWAN_MSG_SEND_UDATA_IND          0x0F
#define LORAWAN_MSG_RECV_UDATA_IND          0x10

#define LORAWAN_MSG_SEND_CDATA_REQ          0x11
#define LORAWAN_MSG_SEND_CDATA_RSP          0x12
#define LORAWAN_MSG_SEND_CDATA_IND          0x13
#define LORAWAN_MSG_RECV_CDATA_IND          0x14

#define LORAWAN_MSG_RECV_ACK_IND            0x15
#define LORAWAN_MSG_RECV_NODATA_IND         0x16

//----------------------------------------------------------------------------
-------
```

```
//
//  Function Prototypes
//
//------------------------------------------------------------------------
-----

// Init
void
WiMOD_LoRaWAN_Init(const char* comPort);

// Send Ping
int
WiMOD_LoRaWAN_SendPing();

// Send unconfirmed radio data
int
WiMOD_LoRaWAN_SendURadioData(UINT8 port, UINT8* data, int length);

// Send confirmed radio data
int
WiMOD_LoRaWAN_SendCRadioData(UINT8 port, UINT8* data, int length);

// Receiver Process
void
WiMOD_LoRaWAN_Process();

#endif // WIMOD_LORAWAN_API_H

//------------------------------------------------------------------------
-----
// end of file
//------------------------------------------------------------------------
-----


//------------------------------------------------------------------------
-----
//
//  File:       WiMOD_LoRaWAN_API.cpp
//
//  Abstract:   API Layer of LoRaWAN Host Controller Interface
//
//  Version:    0.1
//
//  Date:       18.05.2016
//
//  Disclaimer: This example code is provided by IMST GmbH on an "AS IS"
//              basis without any warranties.
//
//------------------------------------------------------------------------
-----


//------------------------------------------------------------------------
-----
//
// Include Files
//
//------------------------------------------------------------------------
-----

#include "WiMOD_LoRaWAN_API.h"
```

```
#include "WiMOD_HCI_Layer.h"
#include <string.h>
#include <stdio.h>

//------------------------------------------------------------------------------
-----
//
//  Forward Declarations
//
//------------------------------------------------------------------------------
-----

// HCI Message Receiver callback
static TWiMOD_HCI_Message*
WiMOD_LoRaWAN_Process_RxMessage(TWiMOD_HCI_Message*  rxMessage);

static void
WiMOD_LoRaWAN_Process_DevMgmt_Message(TWiMOD_HCI_Message*  rxMessage);

static void
WiMOD_LoRaWAN_Process_LoRaWAN_Message(TWiMOD_HCI_Message*  rxMessage);

//------------------------------------------------------------------------------
-----
//
//  Section RAM
//
//------------------------------------------------------------------------------
-----

// reserve one TxMessage
TWiMOD_HCI_Message TxMessage;

// reserve one RxMessage
TWiMOD_HCI_Message RxMessage;

//------------------------------------------------------------------------------
-----
//
//  Section Code
//
//------------------------------------------------------------------------------
-----

//------------------------------------------------------------------------------
-----
//
//  Init
//
//  @brief: init complete interface
//
//------------------------------------------------------------------------------
-----

void
WiMOD_LoRaWAN_Init(const char* comPort)
{
    // init HCI layer
    WiMOD_HCI_Init(comPort,                         // comPort
                   WiMOD_LoRaWAN_Process_RxMessage, // receiver callback
                   &RxMessage);                     // rx message
```

```
}

//------------------------------------------------------------------------------
-----
//
//  Ping
//
//  @brief: send a ping message
//
//------------------------------------------------------------------------------
-----

int
WiMOD_LoRaWAN_SendPing()
{
    // 1. init header
    TxMessage.SapID     = DEVMGMT_SAP_ID;
    TxMessage.MsgID     = DEVMGMT_MSG_PING_REQ;
    TxMessage.Length    = 0;

    // 2. send HCI message without payload
    return WiMOD_HCI_SendMessage(&TxMessage);
}

//------------------------------------------------------------------------------
-----
//
//  SendURadioData
//
//  @brief: send unconfirmed radio message
//
//------------------------------------------------------------------------------
-----

int
WiMOD_LoRaWAN_SendURadioData(UINT8  port,
                             UINT8* srcData,
                             int    srcLength)
{
    // 1. check length
    if (srcLength > (WIMOD_HCI_MSG_PAYLOAD_SIZE - 1))
    {
        // error
        return -1;
    }

    // 2. init header
    TxMessage.SapID     = LORAWAN_SAP_ID;
    TxMessage.MsgID     = LORAWAN_MSG_SEND_UDATA_REQ;
    TxMessage.Length    = 1 + srcLength;

    // 3.  init payload
    // 3.1 init port
    TxMessage.Payload[0] = port;

    // 3.2 init radio message payload
    memcpy(&TxMessage.Payload[1], srcData, srcLength);

    // 4. send HCI message with payload
    return WiMOD_HCI_SendMessage(&TxMessage);
}
```

```
//------------------------------------------------------------------------------
-----
//
//  SendCRadioData
//
//  @brief: send confirmed radio message
//
//------------------------------------------------------------------------------
-----

int
WiMOD_LoRaWAN_SendCRadioData(UINT8  port,
                             UINT8* srcData,
                             int    srcLength)
{
    // 1. check length
    if (srcLength > (WIMOD_HCI_MSG_PAYLOAD_SIZE - 1))
    {
        // error
        return -1;
    }

    // 2. init header
    TxMessage.SapID     = LORAWAN_SAP_ID;
    TxMessage.MsgID     = LORAWAN_MSG_SEND_CDATA_REQ;
    TxMessage.Length    = 1 + srcLength;

    // 3.  init payload
    // 3.1 init port
    TxMessage.Payload[0] = port;

    // 3.2 init radio message payload
    memcpy(&TxMessage.Payload[1], srcData, srcLength);

    // 4. send HCI message with payload
    return WiMOD_HCI_SendMessage(&TxMessage);
}

//------------------------------------------------------------------------------
-----
//
//  Process
//
//  @brief: handle receiver process
//
//------------------------------------------------------------------------------
-----

void
WiMOD_LoRaWAN_Process()
{
    // call HCI process
    WiMOD_HCI_Process();
}

//------------------------------------------------------------------------------
-----
//
//  Process
//
```

```
//  @brief: handle receiver process
//
//----------------------------------------------------------------------------
-----

static TWiMOD_HCI_Message*
WiMOD_LoRaWAN_Process_RxMessage(TWiMOD_HCI_Message*  rxMessage)
{
    switch(rxMessage->SapID)
    {
        case DEVMGMT_SAP_ID:
            WiMOD_LoRaWAN_Process_DevMgmt_Message(rxMessage);
            break;


        case LORAWAN_SAP_ID:
            WiMOD_LoRaWAN_Process_LoRaWAN_Message(rxMessage);
            break;
    }
    return &RxMessage;
}

//----------------------------------------------------------------------------
-----
//
//  Process_DevMgmt_Message
//
//  @brief: handle received Device Management SAP messages
//
//----------------------------------------------------------------------------
-----

static void
WiMOD_LoRaWAN_Process_DevMgmt_Message(TWiMOD_HCI_Message*  rxMessage)
{
    switch(rxMessage->MsgID)
    {
        case    DEVMGMT_MSG_PING_RSP:
                printf("Ping Response, Status : 0x%02X\n\r",
(UINT8)rxMessage->Payload[0]);
                break;

        default:
                printf("unhandled DeviceMgmt message received - MsgID :
0x%02X\n\r", (UINT8)rxMessage->MsgID);
                break;
    }
}

//----------------------------------------------------------------------------
-----
//
//  Process_LoRaWAN_Message
//
//  @brief: handle received LoRaWAN SAP messages
//
//----------------------------------------------------------------------------
-----

static void
WiMOD_LoRaWAN_Process_LoRaWAN_Message(TWiMOD_HCI_Message*  rxMessage)
```

```
{
    switch(rxMessage->MsgID)
    {
        case    LORAWAN_MSG_SEND_UDATA_RSP:
                printf("Send U-Data Response, Status : 0x%02X\n\r",
(UINT8)rxMessage->Payload[0]);
                break;

        case    LORAWAN_MSG_SEND_CDATA_RSP:
                printf("Send C-Data Response, Status : 0x%02X\n\r",
(UINT8)rxMessage->Payload[0]);
                break;

        default:
                printf("unhandled LoRaWAN SAP message received - MsgID :
0x%02X\n\r", (UINT8)rxMessage->MsgID);
                break;
    }
}

//------------------------------------------------------------------------------
-----
// end of file
//------------------------------------------------------------------------------
-----
```

6.6.3 WiMOD HCI Message Layer

```
//------------------------------------------------------------------------------
-----
//
//  File:       WiMOD_HCI_Layer.h
//
//  Abstract:   WiMOD HCI Message Layer
//
//  Version:    0.1
//
//  Date:       18.05.2016
//
//  Disclaimer: This example code is provided by IMST GmbH on an "AS IS"
//              basis without any warranties.
//
//------------------------------------------------------------------------------
-----

#ifndef WIMOD_HCI_LAYER_H
#define WIMOD_HCI_LAYER_H

//------------------------------------------------------------------------------
-----
//
//  Include Files
//
//------------------------------------------------------------------------------
-----

#include <stdint.h>

//----------------------------------------------------------------------------
-------
```

```
//
//  General Declarations & Definitions
//
//------------------------------------------------------------------------------
-----

typedef unsigned char                    UINT8;
typedef uint16_t                         UINT16;

#define WIMOD_HCI_MSG_HEADER_SIZE        2
#define WIMOD_HCI_MSG_PAYLOAD_SIZE       300
#define WIMOD_HCI_MSG_FCS_SIZE           2

#define LOBYTE(x)                        (x)
#define HIBYTE(x)                        ((UINT16)(x) >> 8)

//------------------------------------------------------------------------------
-----
//
//  HCI Message Structure (internal software usage)
//
//------------------------------------------------------------------------------
-----

typedef struct
{
    // Payload Length Information,
    // this field not transmitted over UART interface !!!
    UINT16  Length;

    // Service Access Point Identifier
    UINT8   SapID;

    // Message Identifier
    UINT8   MsgID;

    // Payload Field
    UINT8   Payload[WIMOD_HCI_MSG_PAYLOAD_SIZE];

    // Frame Check Sequence Field
    UINT8   CRC16[WIMOD_HCI_MSG_FCS_SIZE];

}TWiMOD_HCI_Message;

//------------------------------------------------------------------------------
-----
//
//  Function Prototypes
//
//------------------------------------------------------------------------------
-----

// Message receiver callback
typedef TWiMOD_HCI_Message* (*TWiMOD_HCI_CbRxMessage)(TWiMOD_HCI_Message*
rxMessage);

// Init HCI Layer
bool
WiMOD_HCI_Init(const char*               comPort,
               TWiMOD_HCI_CbRxMessage    cbRxMessage,
               TWiMOD_HCI_Message*       rxMessage);
```

```
// Send HCI Message
int
WiMOD_HCI_SendMessage(TWiMOD_HCI_Message* txMessage);

// Receiver Process
void
WiMOD_HCI_Process();

#endif // WIMOD_HCI_LAYER_H

//------------------------------------------------------------------------------
-----
// end of file
//------------------------------------------------------------------------------
-----


//------------------------------------------------------------------------------
-----
//
//  File:       WiMOD_HCI_Layer.cpp
//
//  Abstract:   WiMOD HCI Message Layer
//
//  Version:    0.1
//
//  Date:       18.05.2016
//
//  Disclaimer:This example code is provided by IMST GmbH on an "AS IS"
//              basis without any warranties.
//
//------------------------------------------------------------------------------
-----

//------------------------------------------------------------------------------
-----
//
// Include Files
//
//------------------------------------------------------------------------------
-----

#include "WiMOD_HCI_Layer.h"
#include "CRC16.h"
#include "SLIP.h"
#include "SerialDevice.h"
#include <string.h>

//------------------------------------------------------------------------------
-----
//
//  Forward Declaration
//
//------------------------------------------------------------------------------
-----

// SLIP Message Receiver Callback
static UINT8*   WiMOD_HCI_ProcessRxMessage(UINT8* rxData, int rxLength);

//----------------------------------------------------------------------------
-------
```

```
//
// Declare Layer Instance
//
//----------------------------------------------------------------------------
-----

typedef struct
{
    // CRC Error counter
    UINT32                  CRCErrors;

    // RxMessage
    TWiMOD_HCI_Message*     RxMessage;

    // Receiver callback
    TWiMOD_HCI_CbRxMessage  CbRxMessage;

}TWiMOD_HCI_MsgLayer;

//----------------------------------------------------------------------------
-----
//
//  Section RAM
//
//----------------------------------------------------------------------------
-----

// reserve HCI Instance
static TWiMOD_HCI_MsgLayer  HCI;

// reserve one TxBuffer
static UINT8                TxBuffer[sizeof( TWiMOD_HCI_Message ) * 2 + 2];

//----------------------------------------------------------------------------
-----
//
//  Init
//
//  @brief: Init HCI Message layer
//
//----------------------------------------------------------------------------
-----

bool
WiMOD_HCI_Init(const char*             comPort,        // comPort
               TWiMOD_HCI_CbRxMessage   cbRxMessage,    // HCI msg receiver
callback
               TWiMOD_HCI_Message*      rxMessage)      // intial rxMessage
{
    // init error counter
    HCI.CRCErrors    =    0;

    // save receiver callback
    HCI.CbRxMessage =    cbRxMessage;

    // save RxMessage
    HCI.RxMessage    =    rxMessage;

    // init SLIP
    SLIP_Init(WiMOD_HCI_ProcessRxMessage);
```

```
    // init first RxBuffer to SAP_ID of HCI message, size without 16-Bit
Length Field
    SLIP_SetRxBuffer(&rxMessage->SapID, sizeof(TWiMOD_HCI_Message) -
sizeof(UINT16));

    // init serial device
    return SerialDevice_Open(comPort, Baudrate_115200, DataBits_8,
Parity_None);
}

//------------------------------------------------------------------------------
-----
//
//  SendMessage
//
//  @brief: Send a HCI message (with or without payload)
//
//------------------------------------------------------------------------------
-----

int
WiMOD_HCI_SendMessage(TWiMOD_HCI_Message* txMessage)
{
    // 1. check parameter
    //
    // 1.1 check ptr
    //
    if (!txMessage)
    {
        // error
        return -1;
    }

    // 2. Calculate CRC16 over header and optional payload
    //
    UINT16 crc16 = CRC16_Calc(&txMessage->SapID,
                              txMessage->Length +
WIMOD_HCI_MSG_HEADER_SIZE,
                              CRC16_INIT_VALUE);

    // 2.1 get 1's complement !!!
    //
    crc16 = ~crc16;

    // 2.2 attach CRC16 and correct length, LSB first
    //
    txMessage->Payload[txMessage->Length]     = LOBYTE(crc16);
    txMessage->Payload[txMessage->Length + 1] = HIBYTE(crc16);

    // 3. perform SLIP encoding
    //    - start transmission with SAP ID
    //    - correct length by header size

    int txLength = SLIP_EncodeData(TxBuffer,
                                   sizeof(TxBuffer),
                                   &txMessage->SapID,
                                   txMessage->Length +
WIMOD_HCI_MSG_HEADER_SIZE + WIMOD_HCI_MSG_FCS_SIZE);
    // message ok ?
    if (txLength > 0)
    {
```

```
        // 4. send octet sequence over serial device
        if (SerialDevice_SendData(TxBuffer, txLength) > 0)
        {
            // return ok
            return 1;
        }
    }

    // error - SLIP layer couldn't encode message - buffer to small ?
    return -1;
}

//------------------------------------------------------------------------------
-----
//
//  Process
//
//  @brief: read incoming serial data
//
//------------------------------------------------------------------------------
-----

void
WiMOD_HCI_Process()
{
    UINT8   rxBuf[20];

    // read small chunk of data
    int rxLength = SerialDevice_ReadData(rxBuf, sizeof(rxBuf));

    // data available ?
    if (rxLength > 0)
    {
        // yes, forward to SLIP decoder, decoded SLIP message will be
passed to
        // function "WiMOD_HCI_ProcessRxMessage"
        SLIP_DecodeData(rxBuf, rxLength);
    }
}

//------------------------------------------------------------------------------
-----
//
//  WiMOD_HCI_ProcessRxMessage
//
//  @brief: process received SLIP message and return new rxBuffer
//
//------------------------------------------------------------------------------
-----

static UINT8*
WiMOD_HCI_ProcessRxMessage(UINT8* rxData, int rxLength)
{
    // check min length
    if (rxLength >= (WIMOD_HCI_MSG_HEADER_SIZE + WIMOD_HCI_MSG_FCS_SIZE))
    {
        if (CRC16_Check(rxData, rxLength, CRC16_INIT_VALUE))
        {
            // receiver registered ?
            if (HCI.CbRxMessage)
            {
```

```
                // yes, complete message info
                HCI.RxMessage->Length = rxLength -
(WIMOD_HCI_MSG_HEADER_SIZE + WIMOD_HCI_MSG_FCS_SIZE);

                // call upper layer receiver and save new RxMessage
                HCI.RxMessage = (*HCI.CbRxMessage)(HCI.RxMessage);
            }
        }
        else
        {
            HCI.CRCErrors++;
        }
    }

    // free HCI message available ?
    if (HCI.RxMessage)
    {
        // yes, return pointer to first byte
        return &HCI.RxMessage->SapID;
    }

    // error, disable SLIP decoder
    return 0;
}

//---------------------------------------------------------------------------
-----
// end of file
//---------------------------------------------------------------------------
-----
```

6.6.4 SLIP Encoder / Decoder

```
//------------------------------------------------------------------------------
-----
//
//  File:       SLIP.h
//
//  Abstract:   SLIP Encoder / Decoder
//
//  Version:    0.2
//
//  Date:       18.05.2016
//
//  Disclaimer: This example code is provided by IMST GmbH on an "AS IS"
//              basis without any warranties.
//
//------------------------------------------------------------------------------
-----

#ifndef SLIP_H
#define SLIP_H

//------------------------------------------------------------------------------
-----
//
// Include Files
//
//------------------------------------------------------------------------------
-----

#include <stdint.h>

//------------------------------------------------------------------------------
-----
//
// General Definitions
//
//------------------------------------------------------------------------------
-----

typedef uint8_t     UINT8;

//------------------------------------------------------------------------------
-----
//
//  Function Prototypes
//
//------------------------------------------------------------------------------
-----

// SLIP message receiver callback
typedef UINT8*  (*TSLIP_CbRxMessage)(UINT8* message, int length);

// Init SLIP layer
void
SLIP_Init(TSLIP_CbRxMessage cbRxMessage);

// Init first receiver buffer
bool
SLIP_SetRxBuffer(UINT8* rxBuffer, int rxBufferSize);
```

```
// Encode outgoing Data
int
SLIP_EncodeData(UINT8* dstBuffer, int txBufferSize, UINT8* srcData,int
srcLength);

// Decode incoming Data
void
SLIP_DecodeData(UINT8*  srcData, int srcLength);


#endif // SLIP_H

//------------------------------------------------------------------------
-----
// end of file
//------------------------------------------------------------------------
-----
//------------------------------------------------------------------------
-----
//
//  File:               SLIP.cpp
//
//  Abstract:   SLIP Encoder / Decoder
//
//  Version:    0.2
//
//  Date:               18.05.2016
//
//  Disclaimer:This example code is provided by IMST GmbH on an "AS IS"
//              basis without any warranties.
//
//------------------------------------------------------------------------
-----

//------------------------------------------------------------------------
-----
//
//  Include Files
//
//------------------------------------------------------------------------
-----

#include "SLIP.h"

//------------------------------------------------------------------------
-----
//
//  Protocol Definitions
//
//------------------------------------------------------------------------
-----

// SLIP Protocol Characters
#define SLIP_END                0xC0
#define SLIP_ESC                0xDB
#define SLIP_ESC_END            0xDC
#define SLIP_ESC_ESC            0xDD

// SLIP Receiver/Decoder States
#define SLIPDEC_IDLE_STATE      0
#define SLIPDEC_START_STATE     1
```

```
#define SLIPDEC_IN_FRAME_STATE 2
#define SLIPDEC_ESC_STATE      3

//----------------------------------------------------------------------------
-----
//
// Declare SLIP Variables
//
//----------------------------------------------------------------------------
-----

typedef struct
{
    // Decoder
    int                    RxState;
    int                    RxIndex;
    int                    RxBufferSize;
    UINT8*                 RxBuffer;
    TSLIP_CbRxMessage      CbRxMessage;

    // Encoder
    int                    TxIndex;
    int                    TxBufferSize;
    UINT8*                 TxBuffer;
}TSLIP;

//----------------------------------------------------------------------------
-----
//
// Section RAM
//
//----------------------------------------------------------------------------
-----

// SLIP Instance
static TSLIP   SLIP;

//----------------------------------------------------------------------------
-----
//
// Section Code
//
//----------------------------------------------------------------------------
-----

//----------------------------------------------------------------------------
-----
//
//  Init
//
//  @brief: init SLIP decoder
//
//----------------------------------------------------------------------------
-----

void
SLIP_Init(TSLIP_CbRxMessage cbRxMessage)
{
    // init decoder to idle state, no rx-buffer avaliable
    SLIP.RxState         =   SLIPDEC_IDLE_STATE;
    SLIP.RxIndex         =   0;
```

```
    SLIP.RxBuffer          =   0;
    SLIP.RxBufferSize      =   0;

    // save message receiver callback
    SLIP.CbRxMessage       =   cbRxMessage;

    // init encoder
    SLIP.TxIndex           =   0;
    SLIP.TxBuffer          =   0;
    SLIP.TxBufferSize      =   0;
}

//----------------------------------------------------------------------------
-----
//
//  SLIP_StoreTxByte
//
//  @brief: store a byte into TxBuffer
//
//----------------------------------------------------------------------------
-----

static void
SLIP_StoreTxByte(UINT8 txByte)
{
  if (SLIP.TxIndex < SLIP.TxBufferSize)
      SLIP.TxBuffer[SLIP.TxIndex++] = txByte;
}

//----------------------------------------------------------------------------
-----
//
//  EncodeData
//
//  @brief: encode a messages into dstBuffer
//
//----------------------------------------------------------------------------
-----

int
SLIP_EncodeData(UINT8* dstBuffer, int dstBufferSize, UINT8* srcData, int
srcLength)
{
    // save start pointer
    int txLength = 0;

    // init TxBuffer
    SLIP.TxBuffer = dstBuffer;

    // init TxIndex
    SLIP.TxIndex  = 0;

    // init size
    SLIP.TxBufferSize = dstBufferSize;

    // send start of SLIP message
    SLIP_StoreTxByte(SLIP_END);

    // iterate over all message bytes
    while(srcLength--)
    {
```

```
        switch (*srcData)
        {
                case SLIP_END:
                    SLIP_StoreTxByte(SLIP_ESC);
                    SLIP_StoreTxByte(SLIP_ESC_END);
                    break;

                case SLIP_ESC:
                    SLIP_StoreTxByte(SLIP_ESC);
                    SLIP_StoreTxByte(SLIP_ESC_ESC);
                    break;

                default:
                    SLIP_StoreTxByte(*srcData);
                    break;
        }
        // next byte
        srcData++;
    }

    // send end of SLIP message
    SLIP_StoreTxByte(SLIP_END);

    // length ok ?
    if (SLIP.TxIndex <= SLIP.TxBufferSize)
        return SLIP.TxIndex;

    // return tx length error
    return -1;
}

//---------------------------------------------------------------------------
-----
//
//  SetRxBuffer
//
//  @brief: configure a rx-buffer and enable receiver/decoder
//
//---------------------------------------------------------------------------
-----

bool
SLIP_SetRxBuffer(UINT8* rxBuffer, int rxBufferSize)
{
    // receiver in IDLE state and client already registered ?
    if ((SLIP.RxState == SLIPDEC_IDLE_STATE) && SLIP.CbRxMessage)
    {
        // same buffer params
        SLIP.RxBuffer         = rxBuffer;
        SLIP.RxBufferSize     = rxBufferSize;

        // enable decoder
        SLIP.RxState = SLIPDEC_START_STATE;

        return true;
    }
    return false;
}

//-------------------------------------------------------------------------
-------
```

```
//
//  SLIP_StoreRxByte
//
//  @brief: store SLIP decoded rxByte
//
//----------------------------------------------------------------------------
-----

static void
SLIP_StoreRxByte(UINT8 rxByte)
{
    if (SLIP.RxIndex < SLIP.RxBufferSize)
        SLIP.RxBuffer[SLIP.RxIndex++] = rxByte;
}

//----------------------------------------------------------------------------
-----
//
//  DecodeData
//
//  @brief: process received byte stream
//
//----------------------------------------------------------------------------
-----

void
SLIP_DecodeData(UINT8* srcData, int srcLength)
{
    // init result
    int result = 0;

    // iterate over all received bytes
    while(srcLength--)
    {
        // get rxByte
        UINT8 rxByte = *srcData++;

        // decode according to current state
        switch(SLIP.RxState)
        {
            case    SLIPDEC_START_STATE:
                    // start of SLIP frame ?
                    if(rxByte == SLIP_END)
                    {
                        // init read index
                        SLIP.RxIndex = 0;

                        // next state
                        SLIP.RxState = SLIPDEC_IN_FRAME_STATE;
                    }
                    break;

            case    SLIPDEC_IN_FRAME_STATE:
                    switch(rxByte)
                    {
                        case    SLIP_END:
                                // data received ?
                                if(SLIP.RxIndex > 0)
                                {
                                    // yes, receiver registered ?
                                    if (SLIP.CbRxMessage)
```

```
                                        {
                                            // yes, call message receive
                                            SLIP.RxBuffer =
(*SLIP.CbRxMessage)(SLIP.RxBuffer, SLIP.RxIndex);

                                            // new buffer available ?
                                            if (!SLIP.RxBuffer)
                                            {
                                                SLIP.RxState =
SLIPDEC_IDLE_STATE;
                                            }
                                            else
                                            {
                                                SLIP.RxState =
SLIPDEC_START_STATE;
                                            }
                                        }
                                        else
                                        {
                                            // disable decoder, temp. no buffer
avaliable
                                            SLIP.RxState = SLIPDEC_IDLE_STATE;
                                        }
                                    }
                                    // init read index
                                    SLIP.RxIndex = 0;
                                    break;

                            case  SLIP_ESC:
                                    // enter escape sequence state
                                    SLIP.RxState = SLIPDEC_ESC_STATE;
                                    break;

                            default:
                                    // store byte
                                    SLIP_StoreRxByte(rxByte);
                                    break;
                        }
                        break;

                case    SLIPDEC_ESC_STATE:
                        switch(rxByte)
                        {
                            case    SLIP_ESC_END:
                                    SLIP_StoreRxByte(SLIP_END);
                                    // quit escape sequence state
                                    SLIP.RxState = SLIPDEC_IN_FRAME_STATE;
                                    break;

                            case    SLIP_ESC_ESC:
                                    SLIP_StoreRxByte(SLIP_ESC);
                                    // quit escape sequence state
                                    SLIP.RxState = SLIPDEC_IN_FRAME_STATE;
                                    break;

                            default:
                                    // abort frame receiption
                                    SLIP.RxState = SLIPDEC_START_STATE;
                                    break;
                        }
                        break;
```

```
            default:
                    break;
        }
    }
}

//------------------------------------------------------------------------------
-----
// end of file
//------------------------------------------------------------------------------
-----
```

6.6.5 CRC16 Calculation

```
//------------------------------------------------------------------------------
-----
//
//  File:       CRC16.h
//
//  Abstract:   CRC16 calculation
//
//  Version:    0.2
//
//  Date:       18.05.2016
//
//  Disclaimer:This example code is provided by IMST GmbH on an "AS IS"
//             basis without any warranties.
//
//------------------------------------------------------------------------------
-----

#ifndef     __CRC16_H__
#define     __CRC16_H__

//------------------------------------------------------------------------------
-----
//
//  Section Include Files
//
//------------------------------------------------------------------------------
-----

#include <stdint.h>

//------------------------------------------------------------------------------
-----
//
//  Section Defines & Declarations
//
//------------------------------------------------------------------------------
-----

typedef uint8_t     UINT8;
typedef uint16_t    UINT16;

#define CRC16_INIT_VALUE    0xFFFF    // initial value for CRC algorithem
#define CRC16_GOOD_VALUE    0x0F47    // constant compare value for check
#define CRC16_POLYNOM       0x8408    // 16-BIT CRC CCITT POLYNOM
```

```
//------------------------------------------------------------------------
-----
//
//  Function Prototypes
//
//------------------------------------------------------------------------
-----

// Calc CRC16
UINT16
CRC16_Calc  (UINT8*     data,
             UINT16     length,
             UINT16     initVal);

// Calc & Check CRC16
bool
CRC16_Check (UINT8*     data,
             UINT16     length,
             UINT16     initVal);


#endif // __CRC16_H__
//------------------------------------------------------------------------
-----
// end of file
//------------------------------------------------------------------------
-----
//------------------------------------------------------------------------
-----
//
//  File:       CRC16.cpp
//
//  Abstract:   CRC16 calculation
//
//  Version:    0.2
//
//  Date:       18.05.2016
//
//  Disclaimer:This example code is provided by IMST GmbH on an "AS IS"
//              basis without any warranties.
//
//------------------------------------------------------------------------
-----

//------------------------------------------------------------------------
-----
//
//  Section Include Files
//
//------------------------------------------------------------------------
-----

#include "crc16.h"

// use fast table algorithm
#define __CRC16_TABLE__
//------------------------------------------------------------------------
-----
//
//  Section CONST
//
```

```
//------------------------------------------------------------------------
-----

#ifdef    __CRC16_TABLE__
//------------------------------------------------------------------------
-----
//
//  Lookup Table for fast CRC16 calculation
//
//------------------------------------------------------------------------
-----
const UINT16
CRC16_Table[] =
{
    0x0000, 0x1189, 0x2312, 0x329B, 0x4624, 0x57AD, 0x6536, 0x74BF,
    0x8C48, 0x9DC1, 0xAF5A, 0xBED3, 0xCA6C, 0xDBE5, 0xE97E, 0xF8F7,
    0x1081, 0x0108, 0x3393, 0x221A, 0x56A5, 0x472C, 0x75B7, 0x643E,
    0x9CC9, 0x8D40, 0xBFDB, 0xAE52, 0xDAED, 0xCB64, 0xF9FF, 0xE876,
    0x2102, 0x308B, 0x0210, 0x1399, 0x6726, 0x76AF, 0x4434, 0x55BD,
    0xAD4A, 0xBCC3, 0x8E58, 0x9FD1, 0xEB6E, 0xFAE7, 0xC87C, 0xD9F5,
    0x3183, 0x200A, 0x1291, 0x0318, 0x77A7, 0x662E, 0x54B5, 0x453C,
    0xBDCB, 0xAC42, 0x9ED9, 0x8F50, 0xFBEF, 0xEA66, 0xD8FD, 0xC974,
    0x4204, 0x538D, 0x6116, 0x709F, 0x0420, 0x15A9, 0x2732, 0x36BB,
    0xCE4C, 0xDFC5, 0xED5E, 0xFCD7, 0x8868, 0x99E1, 0xAB7A, 0xBAF3,
    0x5285, 0x430C, 0x7197, 0x601E, 0x14A1, 0x0528, 0x37B3, 0x263A,
    0xDECD, 0xCF44, 0xFDDF, 0xEC56, 0x98E9, 0x8960, 0xBBFB, 0xAA72,
    0x6306, 0x728F, 0x4014, 0x519D, 0x2522, 0x34AB, 0x0630, 0x17B9,
    0xEF4E, 0xFEC7, 0xCC5C, 0xDDD5, 0xA96A, 0xB8E3, 0x8A78, 0x9BF1,
    0x7387, 0x620E, 0x5095, 0x411C, 0x35A3, 0x242A, 0x16B1, 0x0738,
    0xFFCF, 0xEE46, 0xDCDD, 0xCD54, 0xB9EB, 0xA862, 0x9AF9, 0x8B70,
    0x8408, 0x9581, 0xA71A, 0xB693, 0xC22C, 0xD3A5, 0xE13E, 0xF0B7,
    0x0840, 0x19C9, 0x2B52, 0x3ADB, 0x4E64, 0x5FED, 0x6D76, 0x7CFF,
    0x9489, 0x8500, 0xB79B, 0xA612, 0xD2AD, 0xC324, 0xF1BF, 0xE036,
    0x18C1, 0x0948, 0x3BD3, 0x2A5A, 0x5EE5, 0x4F6C, 0x7DF7, 0x6C7E,
    0xA50A, 0xB483, 0x8618, 0x9791, 0xE32E, 0xF2A7, 0xC03C, 0xD1B5,
    0x2942, 0x38CB, 0x0A50, 0x1BD9, 0x6F66, 0x7EEF, 0x4C74, 0x5DFD,
    0xB58B, 0xA402, 0x9699, 0x8710, 0xF3AF, 0xE226, 0xD0BD, 0xC134,
    0x39C3, 0x284A, 0x1AD1, 0x0B58, 0x7FE7, 0x6E6E, 0x5CF5, 0x4D7C,
    0xC60C, 0xD785, 0xE51E, 0xF497, 0x8028, 0x91A1, 0xA33A, 0xB2B3,
    0x4A44, 0x5BCD, 0x6956, 0x78DF, 0x0C60, 0x1DE9, 0x2F72, 0x3EFB,
    0xD68D, 0xC704, 0xF59F, 0xE416, 0x90A9, 0x8120, 0xB3BB, 0xA232,
    0x5AC5, 0x4B4C, 0x79D7, 0x685E, 0x1CE1, 0x0D68, 0x3FF3, 0x2E7A,
    0xE70E, 0xF687, 0xC41C, 0xD595, 0xA12A, 0xB0A3, 0x8238, 0x93B1,
    0x6B46, 0x7ACF, 0x4854, 0x59DD, 0x2D62, 0x3CEB, 0x0E70, 0x1FF9,
    0xF78F, 0xE606, 0xD49D, 0xC514, 0xB1AB, 0xA022, 0x92B9, 0x8330,
    0x7BC7, 0x6A4E, 0x58D5, 0x495C, 0x3DE3, 0x2C6A, 0x1EF1, 0x0F78,
};
#endif
//------------------------------------------------------------------------
-----
//
//  Section Code
//
//------------------------------------------------------------------------
-----

//------------------------------------------------------------------------
-----
//
//  CRC16_Calc
//
```

```
//-----------------------------------------------------------------------------
-----
//
//  @brief:  calculate CRC16
//
//-----------------------------------------------------------------------------
-----
//
//  This function calculates the one's complement of the standard
//  16-BIT CRC CCITT polynomial G(x) = 1 + x^5 + x^12 + x^16
//
//-----------------------------------------------------------------------------
-----

#ifdef    __CRC16_TABLE__
UINT16
CRC16_Calc  (UINT8*             data,
             UINT16             length,
             UINT16             initVal)
{
    // init crc
    UINT16    crc = initVal;

    // iterate over all bytes
    while(length--)
    {
        // calc new crc
        crc = (crc >> 8) ^ CRC16_Table[(crc ^ *data++) & 0x00FF];
    }

    // return result
    return crc;
}
#else
UINT16
CRC16_Calc  (UINT8*             data,
             UINT16             length,
             UINT16             initVal)
{
    // init crc
    UINT16    crc = initVal;

    // iterate over all bytes
    while(length--)
    {
        int     bits    = 8;
        UINT8   byte    = *data++;

        // iterate over all bits per byte
        while(bits--)
        {
            if((byte & 1) ^ (crc & 1))
            {
                crc = (crc >> 1) ^ CRC16_POLYNOM;
            }
            else
            {
                crc >>= 1;
            }

            byte >>= 1;
```

```
        }
    }

    // return result
    return crc;
}
#endif
//------------------------------------------------------------------------------
-----
//
//  CRC16_Check
//
//------------------------------------------------------------------------------
-----
//
//  @brief   calculate & test CRC16
//
//------------------------------------------------------------------------------
-----
//
//  This function checks a data block with attached CRC16
//
//------------------------------------------------------------------------------
-----
bool
CRC16_Check     (UINT8*                     data,
                 UINT16                     length,
                 UINT16                     initVal)
{
    // calc ones complement of CRC16
    UINT16 crc = ~CRC16_Calc(data, length, initVal);

    // CRC ok ?
    return (bool)(crc == CRC16_GOOD_VALUE);
}
//------------------------------------------------------------------------------
-----
// end of file
//------------------------------------------------------------------------------
-----
```

6.7 List of Abbreviations

EEPROM	Electrically Erasable Programmable Read-Only Memory
EIRP	Equivalent Isotropically Radiated Power
EU	Europe
FCC	Federal Communications Commission
FW	Firmware
HCI	Host Controller Interface
GUI	Graphical User Interface
LoRaWAN	Long Range Wide Area Network
LoRa	Long Range
LR	Long Range
RF	Radio Frequency
RSSI	Received Signal Strength Indicator
SLIP	Serial Line Internet Protocol
SNR	Signal to Noise Ratio
UART	Universal Asynchronous Receiver/Transmitter
US	United States
USB	Universal Serial Bus
WiMOD	Wireless Module

6.8 List of References

[1] LoRaWAN® L2 1.0.4 Specification (LoRa Alliance).

[2] RP002-1.0.1 LoRaWAN® Regional Parameters document (LoRa Alliance).

[3] iMxxx_Datasheet.pdf.

[4] WiMOD_LoRaWAN_EndNode_Modem_Region_xxx_HCI_Spec.pdf.

6.9 List of Figures

7. Regulatory Compliance Information

The use of radio frequencies is limited by national regulations. The applicable regulation requirements are subject to change. IMST GmbH does not take any responsibility for the correctness and accuracy of the aforementioned information. National laws and regulations, as well as their interpretation can vary with the country. In case of uncertainty, it is recommended to contact either IMST's accredited Test Center or to consult the local authorities of the relevant countries.

8. Important Notice

8.1 Disclaimer

IMST GmbH points out that all information in this document is given on an "as is" basis. No guarantee, neither explicit nor implicit is given for the correctness at the time of publication. IMST GmbH reserves all rights to make corrections, modifications, enhancements, and other changes to its products and services at any time and to discontinue any product or service without prior notice. It is recommended for customers to refer to the latest relevant information before placing orders and to verify that such information is current and complete. All products are sold and delivered subject to "General Terms and Conditions" of IMST GmbH, supplied at the time of order acknowledgment.

IMST GmbH assumes no liability for the use of its products and does not grant any licenses for its patent rights or for any other of its intellectual property rights or third-party rights. It is the customer's duty to bear responsibility for compliance of systems or units in which products from IMST GmbH are integrated with applicable legal regulations. Customers should provide adequate design and operating safeguards to minimize the risks associated with customer products and applications. The products are not approved for use in life supporting systems or other systems whose malfunction could result in personal injury to the user. Customers using the products within such applications do so at their own risk.

Any reproduction of information in datasheets of IMST GmbH is permissible only if reproduction is without alteration and is accompanied by all given associated warranties, conditions, limitations, and notices. Any resale of IMST GmbH products or services with statements different from or beyond the parameters stated by IMST GmbH for that product/solution or service is not allowed and voids all express and any implied warranties. The limitations on liability in favor of IMST GmbH shall also affect its employees, executive personnel and bodies in the same way. IMST GmbH is not responsible or liable for any such wrong statements.

8.2 Contact Information

IMST GmbH

Carl-Friedrich-Gauss-Str. 2-4
47475 Kamp-Lintfort
Germany

T +49 2842 981 0
F +49 2842 981 299
E wimod@imst.de
I www.wireless-solutions.de