WiMOD LoRaWAN EndNode Modem HCI Specification

Specification Version 1.12

Document ID: 4100/40140/0073

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

Document Information

File name	WiMOD_LoRaWAN_EndNode_Modem_HCI_Spec.docx
Created	2015-01-05
Total pages	83

Revision History

Version	Note
0.1	Created, Initial Version
0.2	Draft Version Created For Review
0.3	Preliminary Version
0.4	Chap. 2.2.4 Byte Ordering added Chap. 3.1.4.2 Firmware Information Element updated Chap. 3.1.5 Real Time Clock Support (RTC) added Chap. 3.2.1 Get Device Activation Parameters added Chap. 3.2.2 Get Join Parameters added Chap. 3.2.6 Radio Stack Configuration added Chap. 4.1 Device Management & LoRA WAN Endpoint Message IDs added
0.5	Firmware V1.3, Build Count 5 HCI output format changed 3.2.2.4, 3.2.2.5 3.2.3.2, 3.2.4.2 3.2.5.1, 3.2.5.2, 3.2.5.2 Chap. 4.1.1 Channel Indices added Chap. 4.1.2 Bandwidth Indices added

0.6	Firmware V1.5 changes Chap. 3.1.4.2 Firmware Information Field corrected Chap. 3.1.3.2 Device Information Field corrected Chap. 3.1.6 System Operation Mode Handling added Chap. 3.2.2.1 Set Join Parameters changed Chap. 3.2.2.2 Get Join Parameters changed Chap. 3.2.6 Number of Wakeup Characters changed to 40 Chap. 3.2.7 Device EUI Configuration added Chap. 3.2.8 Factory Reset added Chap. 3.2.9 Device Deactivation added Chap. 3.2.10 Network Status added Chap. 4.2.2.1 Device Managament Endpoint Msg IDs added Chap. 4.2.3.1 LoRaWAN Endpoint Msg IDs corrected and added Chap. 4.1.3 LoRa Spreading Factors added
0.7	Firmware V1.7 3.2.2.4, 3.2.2.5, 3.2.3.2, 3.2.4.2, 3.2.5.1, 3.2.5.2, 3.2.5.3 Spreading Factor changed to Data Rate Index, Bandwidth Index removed 4.1.2 Bandwidth Indices changed to Data Rate Indices 4.1.3 LoRa Spreading Factor Indices removed
0.8	Missing Rx No-Data Indication added
0.9	Chap. 3.2.6 Automatic Power Saving : Wakeup Character changed to SLIP_END (0xC0) Firmware V1.9 Frame Pending Bit signaled via HCI Ack Indication obsolete 3.2.5.2
1.0	Chapter 3.2.6 updated for - Dutcy Cycle Control - Class A & C Support - MAC Command Support Chapter 3.2.11 LoRaWAN MAC Commands Support added
1.1	Firmware V1.11 Chapter 1 updated with the reference for [3] Chapter 3.2.1 and 3.2.2 updated for "alive" message Chapter 3.2.2.5 updated for join procedure Chapter 3.2.6 updated for MAC Events Support Chapter 3.2.3.1, 3.2.4.1 and 4.2.3.2 updated for channel blocked by ducy cycle Chapter 4.2.3.2 updated for LORAWAN_STATUS_LENGTH_ERROR
1.2	Chapter 3.2.6.1 and 3.2.6.2 updated description for TX Power Level

1.3	Chapter 3.1.5 Device Status added Chapter 3.2.6 updated description for configurable number of retransmissions
1.4	Chapter 3.1.3.1 updated for iM880B-L module Chapter 3.2.6 updated for max retransmissions allowed
1.5	Chapter 3.2.6 and 4 updated for multi band support
1.6	Chapter 3.2.6 updated
1.7	Chapter 4.1.5 and 4.3.3.2 updated
1.8	Chapter 4.1.5 updated for Rx2 settings
1.9	Firmware V1.15 Chapter 3.2.1 and 3.2.2 read-out option removed Chapter 3.2.8 Custom Configuration added Chapter 4.1.6 added Chapter 4.3.3.2 updated
1.10	Chapter 2.2 HCI Message Format updated Chapter 3.2.6 updated for Extended HCI Output Support Chapter 4.4 Example Application and SLIP encoder / Decoder added
1.11	Firmware V1.16 Byte ordering clarification for multi-octet values Device Key renamed to Application Key Chapter 3.1.6 updated for new RTC Alarm Chapter 3.2.11 updated for "joining (OTAA)" network status Chapter 3.1.3.2 updated for iU880B
1.12	Firmware V1.16, Build Count 76 Chapter 3.1.3.2 updated for iM881A Chapter 3.2.1.2 added for end-device reactivation Chapter 3.2.6.4 added for default radio stack configuration Chapter 3.2.8.3 added for default custom configuration Chapter 3.2.2.4 updated for optional device address

Aim of this Document

This document describes the WiMOD LoRaWAN EndNode Modem Host Controller Interface (HCI) protocol which is part of the WiMOD LoRaWAN EndNode Modem firmware. This firmware can be used in combination with the WiMOD LoRa radio module family.

Table of Contents

1. Introduction

1.1 Overview

The WiMOD LoRaWAN EndNode Modem HCI protocol is designed to expose the radio firmware services to an external host controller. A detailed feature description is given in [3].

The communication between host and the radio (WiMOD) is based on so called HCI messages which can be sent through a UART interface (see Fig. 1-1). The WiMOD LoRaWAN EndNode Modem firmware provides several services for configuration, control and radio link access.

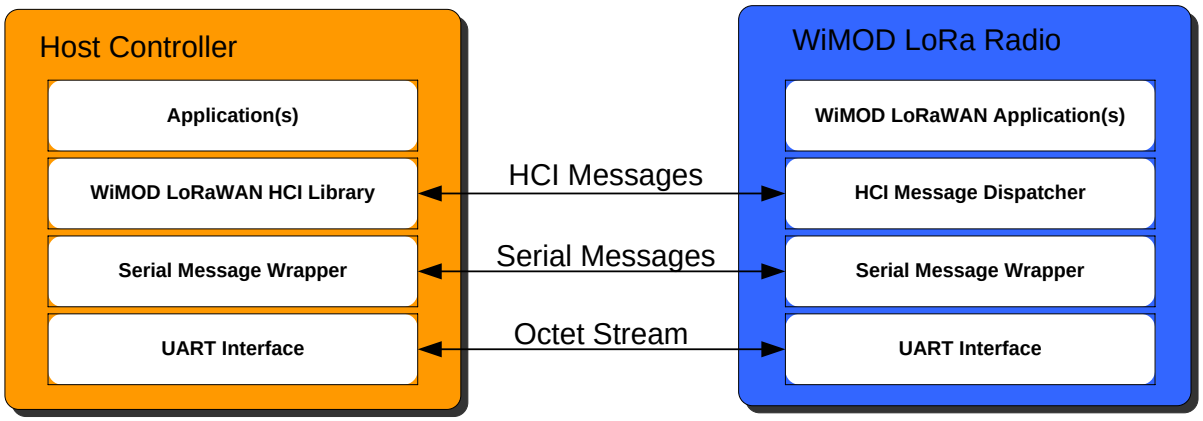

Fig. 1-1: Host Controller Communication

Document Guide

Chapter 2 explains the message flow between host controller and WiMOD LoRa radio module and describes the general message format.

Chapter 3 gives a detailed summary of the services provided by the firmware.

Chapter 4 includes some example code and several tables with defined constants.

2. HCI Communication

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

2.1 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.

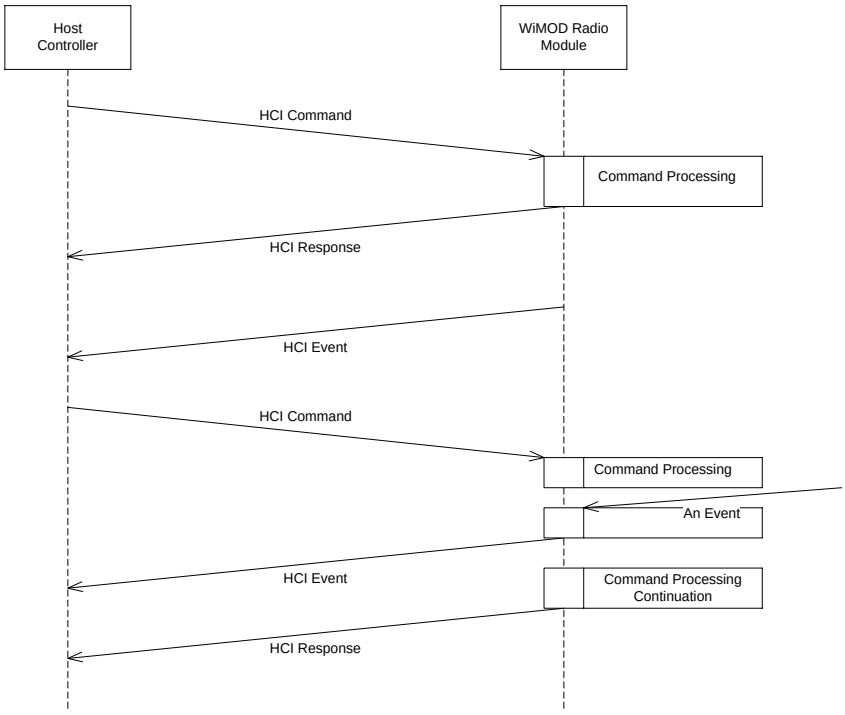

Fig. 2-1: HCI Message Flow

2.2 HCI Message Format

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

HCI Message

Fig. 2-2: HCI Message Format

2.2.1 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.

2.2.2 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.

2.2.3 Payload Field

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

2.2.4 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.

2.2.5 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 4 for CRC16 example).

2.2.6 Communication over UART

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

2.2.6.1 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.

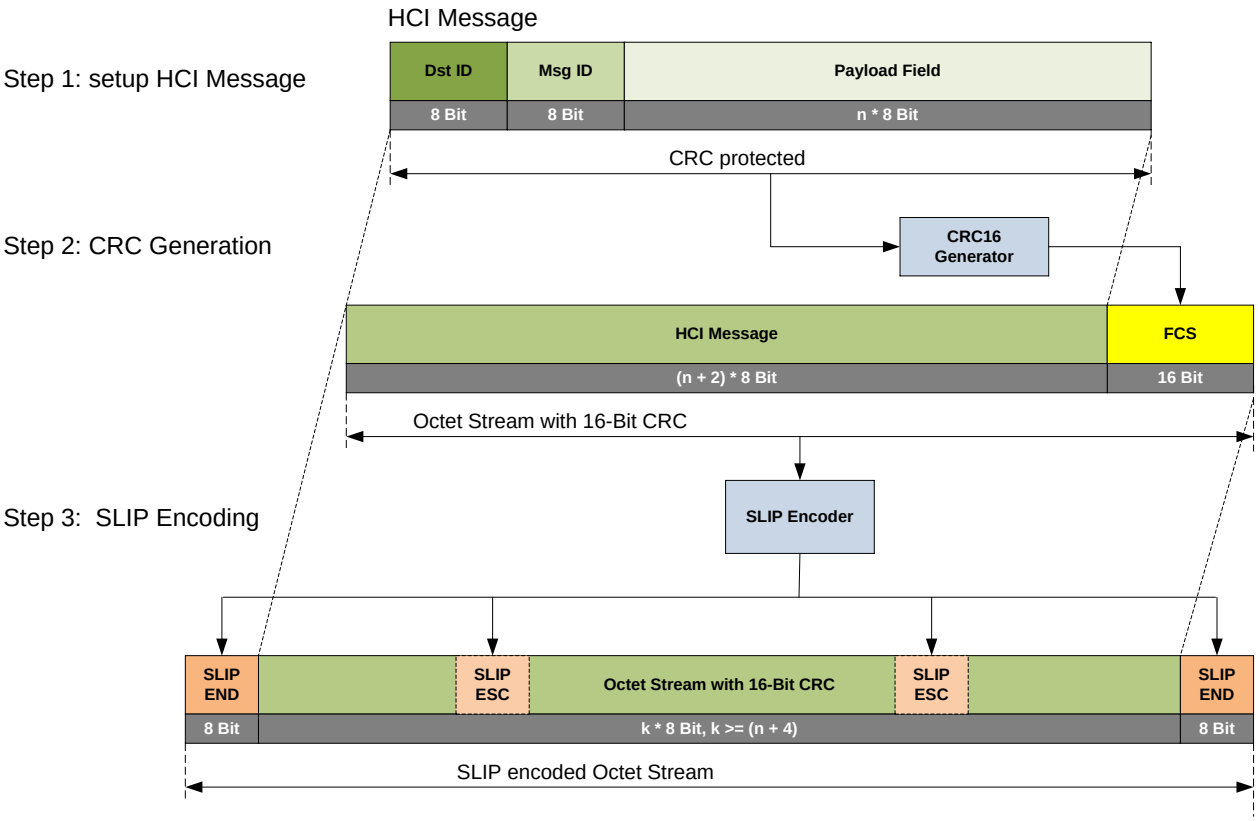

Fig. 2-3: Communication over UART

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.

2.2.6.2 Physical Parameters

The default UART settings are:

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

3. Firmware Services

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

3.1 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:

- Ping
- Reset
- Get Device Information
- Get Firmware Information
- RTC Configuration and RTC Alarm Support
- System Operation Mode Handling

3.1.1 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

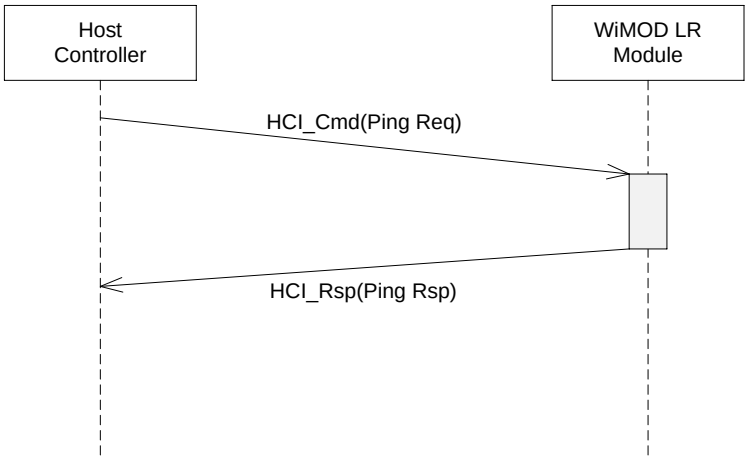

Fig. 3-1: Ping Request

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

3.1.2 Reset

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

Message Flow

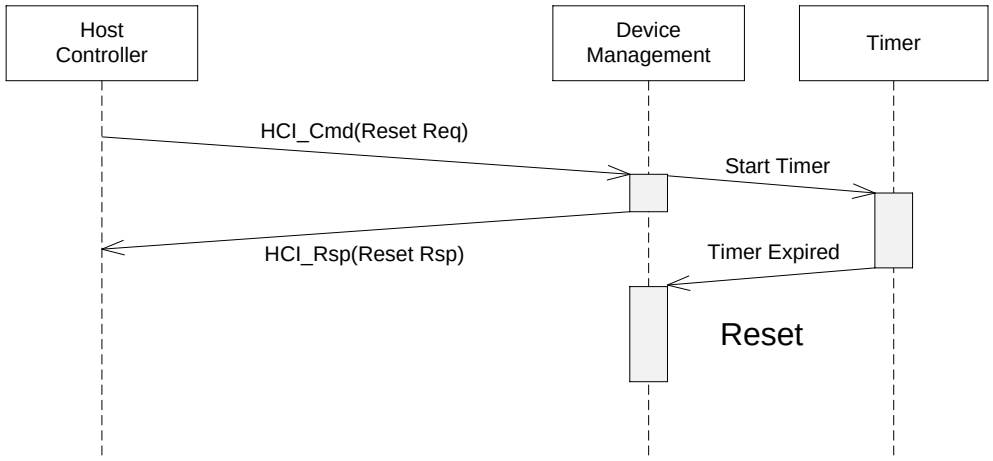

Fig. 3-2: Reset Request

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

3.1.3 Device Information

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

3.1.3.1 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
Payload[1..9]	Device Information Field	see below

3.1.3.2 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 0xA0 = iM881A
1	4	Device Address	32-Bit Device Address for radio communication
5	4	Device ID	32-Bit Device ID for identification purpose

3.1.4 Firmware Information

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

3.1.4.1 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
Payload[1..n]	Firmware Information Field	see below

3.1.4.2 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

3.1.5 Device Status

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

3.1.5.1 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	60	60 octets
Payload[0]	Status Byte	see appendix
Payload[1..59]	Device Status Field	see below

3.1.5.3 Device Status Field

The Device Status Field includes the following information elements:

Offset	Size	Name	Description
0	1	System Tick Resolution	System Tick Resolution in milliseconds (e.g.: 5 = 5ms)
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
15	4	Tx U-Data	Number of unreliable radio packets transmitted
19	4	Tx C-Data	Number of reliable radio packets transmitted
23	4	Tx Error	Number of radio packets not transmitted due to an error
27	4	Rx1 U-Data	Number of unreliable radio packets received in 1st window
31	4	Rx1 C-Data	Number of reliable radio packets received in 1st window
35	4	Rx1 MIC-Error	Number of radio packets received in 1st window with MIC error
39	4	Rx2 U-Data	Number of unreliable radio packets received in 2nd window
43	4	Rx2 C-Data	Number of reliable radio packets received in 2nd window
47	4	Rx2 MIC-Error	Number of radio packets received in 2nd window with MIC error
51	4	Tx Join	Number of join request radio packets transmitted
55	4	Rx Accept	Number of join accept radio packets received

3.1.6 Real Time Clock Support (RTC)

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

3.1.6.1 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
Payload[1-4]	32 Bit time	see RTC Time Format

3.1.6.2 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

3.1.6.3 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

3.1.6.4 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

3.1.6.5 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

3.1.6.6 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
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)

3.1.6.7 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

3.1.7 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 write access to 64-bit Device EUI

3.1.7.1 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
Payload[1]	Current System Operation Mode	see appendix

3.1.7.2 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

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

3.2 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 Multi Band support and Automatic Power Saving
- Device EUI Configuration
- Factory Reset
- Network Status
- LoRaWAN MAC Commands

3.2.1 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 unconfirmed uplink message ("alive" message) over the air.

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.

3.2.1.1 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

3.2.1.2 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
Payload[1..4]	32-Bit Device Address	32-Bit Integer (LSB first)

3.2.2 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.

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 unconfirmed uplink message ("alive" message) over the air.

3.2.2.1 Set Join Parameters

This service can be used to configure the over-the-air activation parameters which are used during the join procedure (see [2], chapter 2.1).

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

- **Application 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 Application 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

3.2.2.2 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

3.2.2.3 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 (+3)	1 (+3) 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 appendix
Payload[2]	Data Rate Index	see appendix
Payload[3]	NumTxPackets	Number of transmitted radio packets of last request

3.2.2.4 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 (the join request will be retransmitted changing the spreading factor from SF7 till SF12, reusing each spreading factor twice).

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 appendix
Payload[6]	Data Rate Index	see appendix
Payload[7]	RSSI	RSSI value in dBm
Payload[8]	SNR	SNR value in dB
Payload[9]	Rx Slot	Rx Slot value

3.2.3 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.

The following three 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

3.2.3.1 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_**U**DATA_REQ	Send Unreliable Data Request
Length	1+n	1+n octets
Payload[0]	LoRaWAN Port	LoRaWAN Port number (> 0)
Payload[1..n]	Application Payload	Application Layer Payload

Response Message

Field	Content	Description
Endpoint ID	LORAWAN _ID	Endpoint Identifier
Msg ID	LORAWAN _MSG_SEND_**U**DATA_RSP	Send Unreliable Data Response
Length	1 (+4)	1 (+4) octet
Payload[0]	Status Byte	see appendix
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)

3.2.3.2 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_**U**DATA_TX_IND	Send Unreliable Data Tx Indication
Length	1 (+2)	1 (+2) 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 appendix
Payload[2]	Data Rate Index	see appendix

3.2.4 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.

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

3.2.4.1 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_**C**DATA_REQ	Send Reliable Data Request
Length	1+n	1+n octets
Payload[0]	LoRaWAN Port	LoRaWAN Port number (>0)
Payload[1..n]	Application Payload	Application Layer Payload

Response Message

Field	Content	Description
Endpoint ID	LORAWAN _ID	Endpoint Identifier
Msg ID	LORAWAN _MSG_SEND_**C**DATA_RSP	Send Reliable Data Response
Length	1 (+4)	1 (+4) octet
Payload[0]	Status Byte	see appendix
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)

3.2.4.2 Reliable 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_**C**DATA_TX_IND	Send Reliable Data Tx Indication
Length	1 (+3)	1 (+3) 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 appendix
Payload[2]	Data Rate Index	see appendix
Payload[3]	NumTxPackets	Number of transmitted radio packets of last request

3.2.5 Ack & Data Reception

The LoRaWAN Stack is able to receive packets within dedicated Rx timeslots scheduled as defined in [2].

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
- Ack Indication
- No-Data Indication

3.2.5.1 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_**U**DATA_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
Payload[2..n]	Application Payload	Application Layer Payload
Payload[n+1]	Channel Index	see appendix
Payload[n+2]	Data Rate Index	see appendix
Payload[n+3]	RSSI	RSSI value in dBm
Payload[n+4]	SNR	SNR value in dB
Payload[n+5]	Rx Slot	Rx Slot value

3.2.5.2 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.

Event Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_RECV_**C**DATA_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
Payload[2..n]	Application Payload	Application Layer Payload
Payload[n+1]	Channel Index	see appendix
Payload[n+2]	Data Rate Index	see appendix
Payload[n+3]	RSSI	RSSI value in dBm
Payload[n+4]	SNR	SNR value in dB
Payload[n+5]	Rx Slot	Rx Slot value

3.2.5.3 Ack Indication (obsolete)

This HCI message is sent to the host after reception of a radio packet without application payload. The radio packet only confirms the reception of the last transmitted reliable packet on server side.

This message is obsolete now. The event will be indicated via Unreliable or Reliable Data Indication with empty LoRaWAN Port and empty Application Payload Field.

Event Message

Field	Content	Description
Endpoint ID	LORAWAN_ID	Endpoint Identifier
Msg ID	LORAWAN_MSG_RECV_ACK_IND	Ack Indication
Length	1(+5)	1(+5) octets
Payload[0]	Status and Format	Bit 0 : 0 = no attachment 1 = Rx Channel Info attached
Payload[1]	Channel Index	see appendix
Payload[2]	Data Rate Index	see appendix
Payload[3]	RSSI	RSSI value in dBm
Payload[4]	SNR	SNR value in dB
Payload[5]	Rx Slot	Rx Slot value

3.2.5.4 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 octet
Payload[0]	Status and Format	0 = not further attachement

3.2.6 Radio Stack Configuration

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

- **Data Rate**
  used for unreliable and confirmed data packets (not join message). This value is used in the next uplink and may change automatically during runtime or via LoRaWAN MAC commands from network server side.
- **TX Power Level**
  this value is used in the next uplink and may change automatically.
- **Adaptive Date Rate**
  this feature can be enabled to allow an automatic data rate adaption from server side (see [2]).
- **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.
- **Duty Cycle Control**
  this function can be disabled for test purpose.
- **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 Command to the corresponding host.
- **Extended HCI Output Support**
  this feature enables extended RF packet output format, where the Tx/Rx channel info is attached.
- **Number of Retransmissions**
  this value sets the maximum number of retries for a reliable radio packet where an acknowledgment is not received.
- **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.

3.2.6.1 Set Radio Stack Configuration

This service can be used to configure the integrated radio stack.

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	6	6 octets
Payload[0]	Default Data Rate Index	see appendix
Payload[1]	Default TX Power Level	Tx Power value in dBm (parameter range: 0 dBm to 20 dBm 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 Bit 2: 0 = Class A selected 1 = Class C selected 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]	Power Saving Mode	0x00 : off 0x01 : automatic
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 appendix)

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 octet
Payload[0]	Status Byte	see appendix

3.2.6.3 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	7	7 octets
Payload[0]	Status Byte	see appendix
Payload[1]	Default Data Rate Index	see appendix
Payload[2]	Default TX Power Level	Tx Power value in dBm (parameter range: 0 dBm to 20 dBm 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 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]	Power Saving Mode	0x00 : off 0x01 : automatic
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 appendix)

3.2.6.4 Default Radio Stack Configuration

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

Parameter	Value
Band Index	1 (EU 868 MHz)
Data Rate Index	3 (SF9 / BW125 kHz)
TX Power Level	14 dBm
Adaptive Data Rate	Enabled
Duty Cycle Control	Enabled
Class C Support	Disabled (Class A selected)
MAC Events Support	Enabled
Extended HCI Output Support	Enabled
Automatic Power Saving	Disabled
Number of Retransmissions	7

3.2.7 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.

3.2.7.1 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
Payload[1-8]	64-Bit Device EUI	Octet sequence (MSB first)

3.2.7.2 Set Device EUI

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

Note: this parameter 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

3.2.8 Custom Configuration

The following custom parameters can be configured via HCI:

- **TX Power Offset**
  this value defines a transmission power offset to be added in the uplinks. This could be used to compensate possible transmission losses.

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

3.2.8.1 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
Payload[1]	TX Power Offset	Tx Power Offset value in dB (parameter range: -128 dB to 127 dB in 1 dB steps)

3.2.8.2 Set Custom Configuration

This message can be used to configure the custom parameters.

Note: this parameter 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]	TX Power Offset	Tx Power Offset value in dB (parameter range: -128 dB to 127 dB 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

3.2.8.3 Default Custom Configuration

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

Parameter	Value
TX Power Offset	0 dB

3.2.9 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

3.2.10 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

3.2.11 Network/Activation Status

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

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	2 octets
Payload[0]	Status Byte	see appendix
Payload[1]	Network Status:	1 octet 0x00 : device inactive 0x01 : active (ABP) 0x02 : active (OTAA) 0x03 : joining (OTAA)

3.2.12 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.

3.2.12.1 Send MAC Command

This command can be used to send a single MAC command.

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 [2])
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

3.2.12.2 Receive MAC Command

This HCI message is sent to the host after reception of a radio packet including MAC command(s). 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 [2]
Payload[n+1]	Channel Index	see appendix
Payload[n+2]	Data Rate Index	see appendix
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. Appendix

4.1 Multi Band Support

4.1.1 Radio Band Indices

Index	Band Description
1	EU 868 MHz - Europe
2	Reserved (US 915 MHz)
3	IN 865 MHz - India
4	NZ 865 MHz - New Zealand
5	SG 923 MHz - Singapore
129	EU 868 MHz - Europe (RX2: SF9)

4.1.2 Europe 868 MHz Band

4.1.2.1 Data Rate Indices

Index	Data Rate / Spreading Factor	Bandwidth	Indicative physical bit rate [bit/s]
0	LoRa / SF12	125 kHz	250
1	LoRa / SF11	125 kHz	440
2	LoRa / SF10	125 kHz	980
3	LoRa / SF9	125 kHz	1760
4	LoRa / SF8	125 kHz	3125
5	LoRa / SF7	125 kHz	5470
6	LoRa / SF7	250 kHz	11000
7	FSK / 50kbps		50000

4.1.2.2 Channel Indices

Index	Frequency Channel	Comments
0	868 100 000 Hz	Data Rates 0 - 5
1	868 300 000 Hz	Data Rates 0 - 7
2	868 500 000 Hz	Data Rates 0 - 5
128	869 525 000 Hz	Default Frequency for Rx2 Default Data Rate: 0

4.1.3 India 865 MHz Band

4.1.3.1 Data Rate Indices

Index	Data Rate / Spreading Factor	Bandwidth	Indicative physical bit rate [bit/s]
0	LoRa / SF12	125 kHz	250
1	LoRa / SF11	125 kHz	440
2	LoRa / SF10	125 kHz	980
3	LoRa / SF9	125 kHz	1760
4	LoRa / SF8	125 kHz	3125
5	LoRa / SF7	125 kHz	5470
6	LoRa / SF7	250 kHz	11000
7	FSK / 50kbps		50000

4.1.3.2 Channel Indices

Index	Frequency Channel	Comments
0	865 062 500 Hz	Data Rates 0 - 5
1	865 402 500 Hz	Data Rates 0 - 5
2	865 985 000 Hz	Data Rates 0 - 5
128	866 550 000 Hz	Default Frequency for Rx2 Default Data Rate: 4

4.1.4 New Zealand 865 MHz Band

4.1.4.1 Data Rate Indices

Index	Data Rate / Spreading Factor	Bandwidth	Indicative physical bit rate [bit/s]
0	LoRa / SF12	125 kHz	250
1	LoRa / SF11	125 kHz	440
2	LoRa / SF10	125 kHz	980
3	LoRa / SF9	125 kHz	1760
4	LoRa / SF8	125 kHz	3125
5	LoRa / SF7	125 kHz	5470
6	LoRa / SF7	250 kHz	11000
7	FSK / 50kbps		50000

4.1.4.2 Channel Indices

Index	Frequency Channel	Comments
0	865 000 000 Hz	Data Rates 0 - 5
1	865 200 000 Hz	Data Rates 0 - 5
2	865 400 000 Hz	Data Rates 0 - 5
3	866 200 000 Hz	Data Rates 0 - 5
4	866 400 000 Hz	Data Rates 0 - 5
5	866 600 000 Hz	Data Rates 0 - 5
6	866 800 000 Hz	Data Rates 0 - 5
7	867 000 000 Hz	Data Rates 0 - 5
8	865 600 000 Hz	Data Rate 6
9	865 900 000 Hz	Data Rate 7
128	867 200 000 Hz	Default Frequency for Rx2 Default Data Rate: 0

4.1.5 Singapore 923 MHz Band

4.1.5.1 Data Rate Indices

Index	Data Rate / Spreading Factor	Bandwidth	Indicative physical bit rate [bit/s]
0	LoRa / SF12	125 kHz	250
1	LoRa / SF11	125 kHz	440
2	LoRa / SF10	125 kHz	980
3	LoRa / SF9	125 kHz	1760
4	LoRa / SF8	125 kHz	3125
5	LoRa / SF7	125 kHz	5470
6	LoRa / SF7	250 kHz	11000
7	FSK / 50kbps		50000

4.1.5.2 Channel Indices

Index	Frequency Channel	Comments
0	923 500 000 Hz	Data Rates 0 - 5
1	923 700 000 Hz	Data Rates 0 - 5
2	923 900 000 Hz	Data Rates 0 - 5
3	924 100 000 Hz	Data Rates 0 - 5
4	924 300 000 Hz	Data Rates 0 - 5
5	924 500 000 Hz	Data Rates 0 - 5
6	924 700 000 Hz	Data Rates 0 - 5
7	924 900 000 Hz	Data Rates 0 - 5
8	923 800 000 Hz	Data Rates 6 - 7
128	922 500 000 Hz	Default Frequency for Rx2 Default Data Rate: 0

4.1.6 Europe 868 MHz (RX2: SF9) Band

This band has the same settings as the Europe 868 MHz band (described in 4.1.2), excluding the data rate used for Rx2.

Note that this band is not compliant to the LoRaWAN specification.

4.1.6.1 Data Rate Indices

Index	Data Rate / Spreading Factor	Bandwidth	Indicative physical bit rate [bit/s]
0	LoRa / SF12	125 kHz	250
1	LoRa / SF11	125 kHz	440
2	LoRa / SF10	125 kHz	980
3	LoRa / SF9	125 kHz	1760
4	LoRa / SF8	125 kHz	3125
5	LoRa / SF7	125 kHz	5470
6	LoRa / SF7	250 kHz	11000
7	FSK / 50kbps		50000

4.1.6.2 Channel Indices

Index	Frequency Channel	Comments
0	868 100 000 Hz	Data Rates 0 - 5
1	868 300 000 Hz	Data Rates 0 - 7
2	868 500 000 Hz	Data Rates 0 - 5
128	869 525 000 Hz	Default Frequency for Rx2 Default Data Rate: 3

4.2 System Operation Modes

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

4.3 List of Constants

4.3.1 List of Endpoint Identifier

Name	Value
DEVMGMT_ID	0x01
LORAWAN_ID	0x10

4.3.2 Device Management Endpoint Identifier

4.3.2.1 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_SET_RTC_ALARM_REQ	0x31
DEVMGMT_MSG_SET_RTC_ALARM_RSP	0x32
DEVMGMT_MSG_CLEAR_RTC_ALARM_REQ	0x33
DEVMGMT_MSG_CLEAR_RTC_ALARM_REQ	0x34
DEVMGMT_MSG_GET_RTC_ALARM_REQ	0x35
DEVMGMT_MSG_GET_RTC_ALARM_RSP	0x36
DEVMGMT_MSG_RTC_ALARM_IND	0x38

4.3.2.2 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

4.3.3 LoRaWAN Endpoint Identifier

4.3.3.1 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_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

4.3.3.2 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)

4.4 Example Code for Host Controller

4.4.1 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
//------------------------------------------------------------------------------
```

4.4.2 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
//------------------------------------------------------------------------------
```

4.4.3 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
//------------------------------------------------------------------------------
```

4.4.4 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
//------------------------------------------------------------------------------
```

4.4.5 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
//------------------------------------------------------------------------------
```

4.5 List of Abbreviations

ABP	Activation by (direct) Personalization
FW	Firmware
HCI	Host Controller Interface
LR	Long Range
LoRa	Long Range
OTAA	Over The Air Activation
RAM	Random Access Memory
RF	Radio Frequency
RSSI	Received Signal Strength Indicator
RTC	Real Time Clock
SLIP	Serial Line Internet Protocol
SNR	Signal to Noise Ratio
UART	Universal Asynchronous Receiver/Transmitter
WiMOD	Wireless Module by IMST

4.6 List of References

[1] iM880A_AN012_RFSettings.pdf

[2] LoRa WAN Specification.pdf

[3] WiMOD_LoRaWAN_EndNode_Modem_Feature_Spec.pdf

4.7 List of Figures

5. Regulatory Compliance Information

The use of radio frequencies is limited by national regulations. The radio module has been designed to comply with the European Union's R&TTE (Radio & Telecommunications Terminal Equipment) directive 1999/5/EC and can be used free of charge within the European Union. Nevertheless, restrictions in terms of maximum allowed RF power or duty cycle may apply.

The radio module has been designed to be embedded into other products (referred as "final products"). According to the R&TTE directive, the declaration of compliance with essential requirements of the R&TTE directive is within the responsibility of the manufacturer of the final product. A declaration of conformity for the radio module is available from IMST GmbH on request.

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.

6. Important Notice

6.1 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.

6.2 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