

WiMOD LR Base Host Controller Interface

Specification Version 1.10

Document ID: 4100/40140/0062

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



Document Information

File name	WiMOD_LR_Base_HCI_Spec.docx
Created	2013-06-10
Total pages	81

Revision History

Version	Note
0.1	Created, Initial Version
0.2	Draft Version Created For Review
0.3	Preliminary Version
1.0	First Release
1.1	Format of several messages changed
1.2	Radio Configuration updated : RTC enable/disable option, Signal Bandwidth 500 kHz added
1.3	Chapter Remote Control added New configuration options added for Firmware V1.6 : <ul style="list-style-type: none"> - HCI Power-Up Indication - HCI Tx Indication - HCI Button Pressed Indication - LED Button Pressed Indication
1.4	Chapter 4.2 Example Code for Host Controller added
1.5	Chapter 3.1.5 Radio Configuration updated Chapter 3.2.2 Confirmed Message Exchange added Chapter 3.4.1.1 Radio Test Parameter updated
1.6	Firmware V1.9 extensions : <ul style="list-style-type: none"> - FSK - Mode added see chap. 3.1.5 Radio Configuration - Automatic Power Saving added see chap 3.1.5 Radio Configuration and chap. 3.1.9 Power Saving - Fixed some typos

1.7	<p>Firmware V1.10 extensions :</p> <ul style="list-style-type: none"> - Listen Before Talk added see chap. 3.1.5 Radio Configuration, chap. 3.1.6 System Status and chap. 3.2.6 Listen Before Talk - AES Encryption/Decryption added see chap. 3.1.5 Radio Configuration, 3.1.12 AES Key Configuration and chap. 3.2.5 Radio Packet Encryption - Automatic Power Saving improved see chap. 3.1.9 Power Saving <p>Chapter 3.2.15 Radio Control Field added Chapter 3.2.16 HCI Format & Status Field added Document renamed to WiMOD LR Base HCI Spec.</p>
1.8	Chapter 3.1.3 : new module types added
1.9	<p>Chapter 2.2 HCI Message Format upated Chapter 4.2 Exampe Code updated</p>
1.10	<p>Chapter 3.2.1 and 3.2.2 : Airtime information of transmitted packet added in HCI event messages Chapter 3.1.4.2 : fix in Firmware Verion Number (byte ordering)</p>

Aim of this Document

This document describes the WiMOD LR Host Controller Interface (HCI) protocol which is part of the WiMOD LR Base firmware. This firmware can be used in combination with the WiMOD LR radio module family (e.g. iM880A-L, iU880A, iM880B).

Table of Contents

1. INTRODUCTION	5
1.1 Overview	5
2. HCI COMMUNICATION	6
2.1 Message Flow	6
2.2 HCI Message Format	7
2.2.1 Destination Endpoint Identifier (DstID)	7
2.2.2 Message Identifier (MsgID)	7
2.2.3 Payload Field	7
2.2.4 Byte Ordering	7
2.2.5 Frame Check Sequence Field (FCS)	7
2.2.6 Communication over UART	7
3. FIRMWARE SERVICES	9
3.1 Device Management Services	9
3.1.1 Ping	10
3.1.2 Reset	11
3.1.3 Device Information	12
3.1.4 Firmware Information	13
3.1.5 Radio Configuration	14
3.1.6 System Status	23
3.1.7 Real Time Clock Support (RTC)	25
3.1.8 Set Radio Mode	27
3.1.9 Power Saving	28
3.1.10 System Operation Modes	28
3.1.11 Power-Up Indication	31
3.1.12 AES Key Configuration (V1.10 and later)	31
3.2 Radio Link Services	33
3.2.1 Unreliable Data Exchange	33
3.2.2 Confirmed Data Exchange	38
3.2.3 Sniffer Mode	42
3.2.4 Echo Mode	44
3.2.5 Radio Packet Encryption	45
3.2.6 Listen Before Talk	48

3.3 Radio Link Test	49
3.3.1 Start Radio Link Test	50
3.3.2 Radio Link Test Status Message	51
3.3.3 Stop Radio Link Test	52
3.4 Hardware Tests	53
3.4.1 Radio Tests	53
3.5 Remote Control	54
3.5.1 Remote Control Button Pressed Indication	54
4. APPENDIX	55
4.1 List of Constants	55
4.1.1 List of Endpoint Identifier	55
4.1.2 Device Management Identifier	55
4.1.3 Radio Link Identifier	57
4.1.4 Radio Link Test Identifier	58
4.1.5 Hardware Test Identifier	59
4.1.6 Remote Control Identifier	59
4.2 Example Code for Host Controller	60
4.2.1 WiMOD HCI Message Layer	60
4.2.2 SLIP Encoder / Decoder	65
4.2.3 CRC16 Calculation	72
4.3 List of Abbreviations	77
4.4 List of References	77
4.5 List of Figures	78
5. REGULATORY COMPLIANCE INFORMATION	79
6. IMPORTANT NOTICE	80
6.1 Disclaimer	80
6.2 Contact Information	80

1. Introduction

1.1 Overview

The WiMOD LR Base HCI protocol is designed to expose the radio module services to an external host controller. The communication between host and the radio (WiMOD) is based on so called HCI messages which can be sent through a UART interface (see Fig. 1-1). The WiMOD LR firmware provides several services for configuration, control and radio link access.

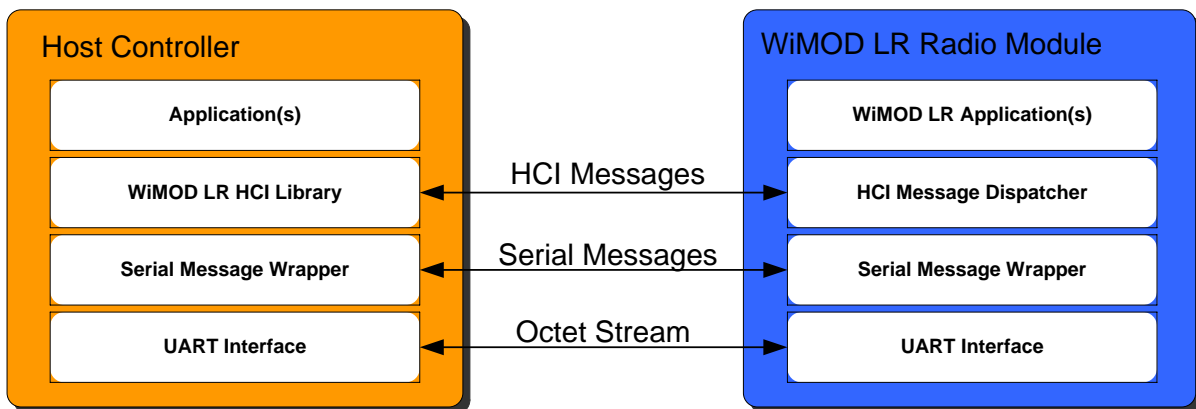


Fig. 1-1: Host Controller Communication

Document Guide

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

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

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

2. HCI Communication

The communication between the WiMOD LR 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 LR 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.

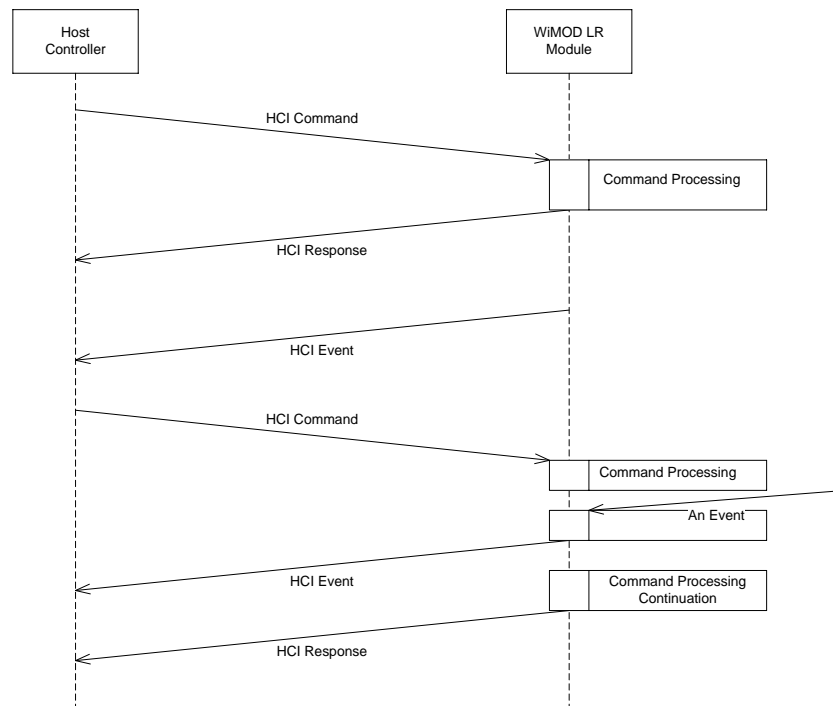


Fig. 2-1: HCI Message Flow

2.2 HCI Message Format

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

HCI Message

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

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 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 LR HCI Protocol uses a 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 figure explains how a HCI message is embedded in a SLIP packet.

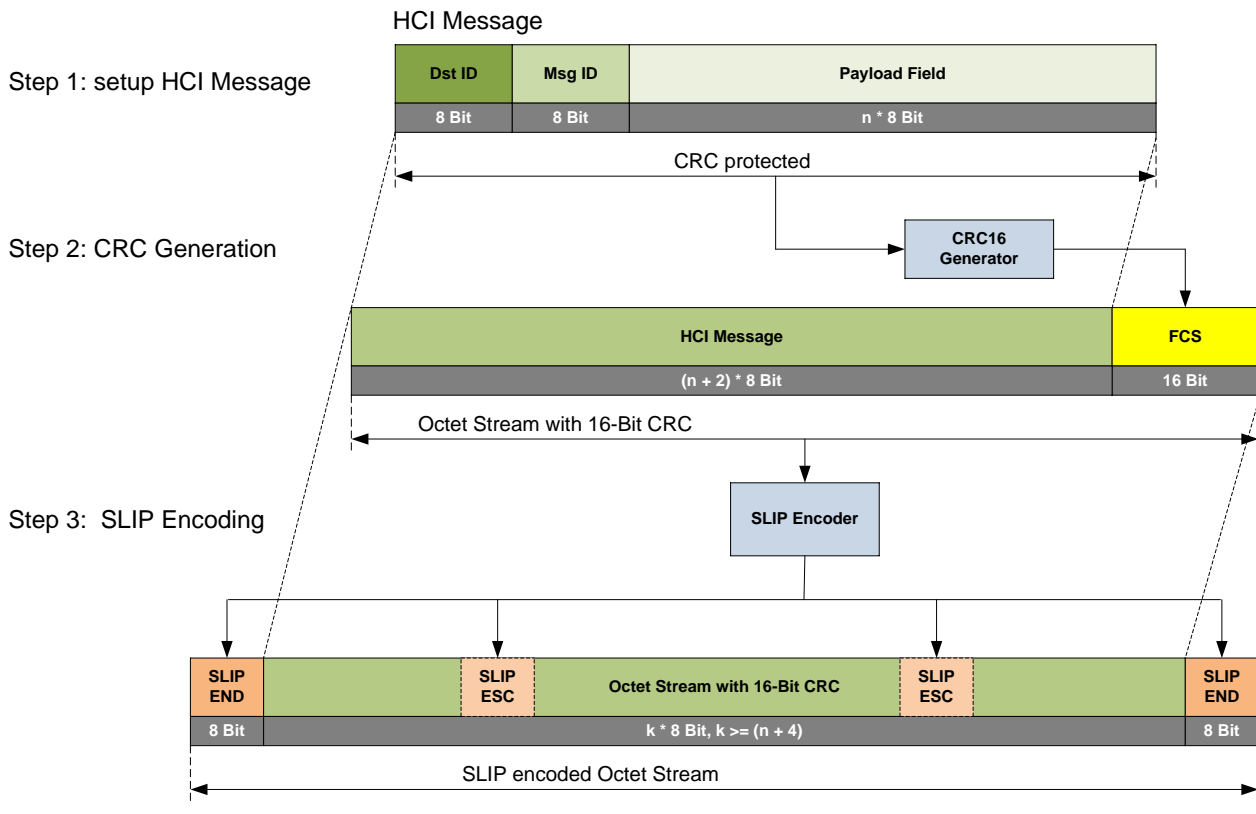


Fig. 2-3: Communication over UART

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 for the iM880A 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. The following endpoints are defined (see chapter 4 for defined constants):

- Device Management (DEVMGMT_ID)
- Radio Link Services (RADIOLINK_ID)
- Radio Link Test (RLT_ID)
- Hardware Test (HWTEST_ID)
- Remote Control (REMOTE_CTRL_ID)

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
- Device Information
- Firmware Information
- Radio Configuration
- System Status
- Real Time Clock Support (RTC)
- Automatic Power Saving
- System Operation Modes

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

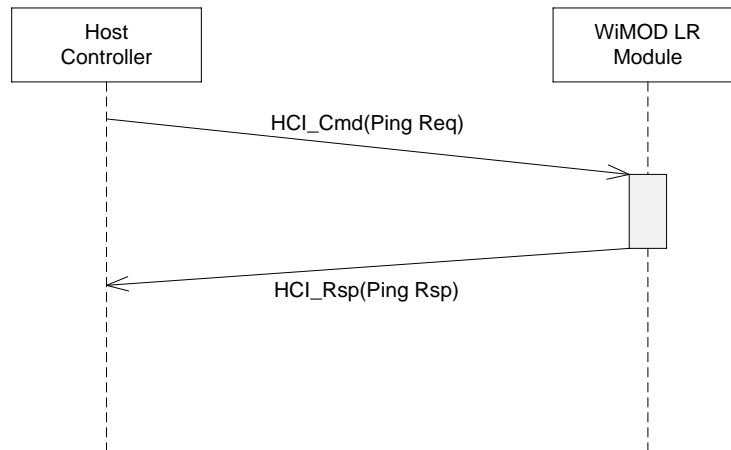


Fig. 3-1: Ping Request

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

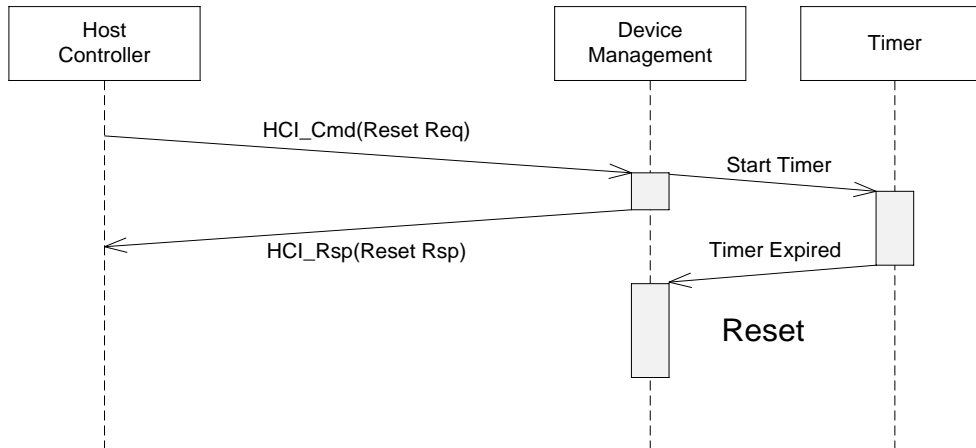


Fig. 3-2: Reset Request

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 Element.

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

3.1.3.2 Device Information Element

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 0x99 = iU880B 0xA0 = iM881A 0xA1 = iU881A
1	2	Device Address	Device Address for radio communication
3	1	Group Address	Group Address for radio communication
4	1	Reserved	Reserved for future usage
5	4	32 Bit Device ID	Unique Device ID for module identification

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 element.

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 Element	see below

3.1.4.2 Firmware Information Element

The Device 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
4	m	Firmware Image	Name of Firmware Image

3.1.5 Radio Configuration

The radio firmware supports several configurable parameters which are stored in the non-volatile flash memory. The configuration parameters are read during start-up to configure the firmware components and hardware units. The following items can be configured:

Item	Description
Radio Mode	Determines if the radio module operates in Standard , Sniffer or Echo mode.
Group Address	Used to separate groups of radio modules. This value is compared against the TxGroupAddress field of a received radio message to filter radio packets in Standard mode (0xFF = BROADCAST address).
Tx Group Address	Reserved for future use
Device Address	Used to address a specific radio device. This value is compared against the TxDeviceAddress field of a received radio message to filter radio packets in Standard mode (0xFFFF = BROADCAST address).
Tx Device Address	Reserved for future use
Modulation	0 = LoRa, 1 = FSK (50000 bps)
RF Carrier Frequency	Defines the used radio frequency. See [1] for further details.
LoRa Signal Bandwidth	Defines the LoRa signal bandwidth 0 = 125 kHz, 1 = 250 kHz, 2 = 500 kHz
LoRa Spreading Factor	Defines the LoRa spreading factor 0 – 7 = SF7, 8 = SF8, 9 = SF9, 10 = SF10, 11 = SF11, 12 = SF12
Error Coding	Defines the radio error coding format 0 = 4/5, 1 = 4/5, 2 = 4/6, 3 = 4/7, 4 = 4/8
Power Level	Defines the transmit power level from 5 dBm to 20 dBm: 0 – 5 = 5 dBm, 6 = 6 dBm, ..., 20 = 20 dBm
Tx Control	Transmitter Control Option: Bit 0: 0 = Tx Narrow Filter off 1 = Tx Narrow Filter on Firmware V1.10 extension Bit 1: 0 = Listen Before Talk (LBT) off 1 = Listen Before Talk (LBT) on See [1] for further details.
Rx Control	Receiver Control Option: 0 = Receiver off 1 = Receiver always on (except during packet transmission) 2 = Receiver on for limited time defined by Rx Window parameter
Rx Window Time	Configurable time for radio receive mode after radio packet transmission. Note: Rx Window option must be enabled in the Rx Control parameter. A value of zero (0) disables the receive mode.

LED Control	<p>Bit field to configure LED control options:</p> <p>Bit 0: 0 = no GPIO access 1 = toggle LED D3 as "Rx Indicator"</p> <p>Bit 1: 0 = no GPIO access 1 = toggle LED D2 as "Tx Indicator"</p> <p>Bit 2: 0 = no GPIO access 1 = toggle LED D4 as "Alive Indicator"</p> <p>Firmware V1.6 extension</p> <p>Bit 3: 0 = no GPIO access 1 = toggle LED D1 as "Button Pressed Indicator"</p>
Misc. Options	<p>Bit field to configure further radio firmware options:</p> <p>Bit 0: 0 = standard RF packet output format 1 = extended RF packet output format: attached RSSI, SNR and Timestamp</p> <p>Bit 1: 0 = RTC disabled 1 = RTC enabled</p> <p>Firmware V1.6 extensions</p> <p>Bit 2: HCI Tx Indication - this message is sent to the host after an RF message was sent over the air. 0 = disabled 1 = enabled</p> <p>Bit 3: HCI Power-Up Indication - this message is sent to the host when the module is ready to communicate after a power-up reset. 0 = disabled 1 = enabled</p> <p>Bit 4: HCI Button Pressed Indication – this message is sent to the host on reception of a button pressed message via RF (see Remote Control). 0 = disabled 1 = enabled</p> <p>Firmware V1.10 extension</p> <p>Bit 5: 0 = AES Encryption/Decryption off 1 = AES Encryption/Decryption on</p>
FSK Datarate	<p>Determines the datarate if FSK modulation is enabled</p> <p>0 = 50000bps 1 = 100000 bps 2 = 250000 bps</p>
<p>Firmware V1.9 extension</p> <p>Power Saving Mode</p>	<p>Defines the Power Saving Mode</p> <p>0 = off 1 = Automatic Mode enabled (RTC off recommended!)</p>
<p>Firmware V1.0 extension</p> <p>Listen Before Talk (LBT) Threshold</p>	<p>Defines the LBT Threshold in dBm. Typical range -120 to 0 dBm</p>

3.1.5.1 Get Radio Configuration

This message can be used to read the configuration parameters.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_RADIO_CONFIG_REQ	Get Radio Config Request
Length	0	no payload

Response Message

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

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_RADIO_CONFIG_RSP	Get Radio Config Response
Length	1 + 25	1 + 25 octets
Payload[0]	Status Byte	see appendix
Payload[1..25]	Radio Configuration Field	see Radio Configuration Field

3.1.5.2 Set Radio Configuration

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

Command Message

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

Response Message

This message acknowledges the Set Radio Configuration Request message.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_RADIO_CONFIG_RSP	Get Radio Config Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix

3.1.5.3 Radio Configuration Field

The Radio Configuration Field contains the following configurable radio parameters:

Offset	Size	Name	Description
0	1	Radio Mode	0x00 = Standard mode: Device & Group address used for packet filtering 0x01 = Echo mode: allows simple packet repetition 0x02 = Sniffer mode: radio packet output via HCI without packet filtering
1	1	Group Address	Own group address (0x01 – 0xFE) for packet filtering (0xFF reserved as BROADCAST address)
2	1	Tx Group Address	Reserved for future use
3	2	Device Address	Own device address (0x0001 – 0xFFFE) for packet filtering (0xFFFF reserved as BROADCAST address)
5	2	Tx Device Address	Reserved for future use
7	1	Modulation	0 = LoRa, 1 = FSK (50000 bps)
8	1	RF Carrier Frequency Least Significant Bits	Defines the used radio frequency. See [1] for details.
9	1	RF Carrier Frequency Intermediate Bits	Defines the used radio frequency. See [1] for details.
10	1	RF Carrier Frequency Most Significant Bits	Defines the used radio frequency. See [1] for details.
11	1	LoRa Signal Bandwidth	0 = 125 kHz, 1 = 250 kHz, 2 = 500 kHz
12	1	LoRa Spreading Factor	0 – 7 = SF7 8 = SF8 9 = SF9 10 = SF10 11 = SF11 12 = SF12
13	1	Error Coding	0 = 4/5 1 = 4/5 2 = 4/6 3 = 4/7 4 = 4/8
14	1	Power Level	0 – 5 = 5 dBm 6 = 6 dBm 7 = 7 dBm ... 20 = 20 dBm

15	1	Tx Control	<p>Transmitter Control Option: Bit 0: 0 = Tx Narrow Filter off 1 = Tx Narrow Filter on</p> <p>Firmware V1.10 extension Bit 1: 0 = LBT off 1 = LBT on</p> <p>See [1] for further details.</p>
16	1	Rx Control	<p>Receiver Control Option: 0 = Receiver off 1 = Receiver always on (except during packet transmission) 2 = Receiver on for limited time defined by Rx Window parameter</p>
17	2	Rx Window Time	<p>0 = receiver disabled, no Rx Window 1 – 65535 = 1 - 65535 ms</p>
19	1	LED Control	<p>Bit 0: 0 = no GPIO access 1 = toggle LED D3 as "Rx Indicator"</p> <p>Bit 1: 0 = no GPIO access 1 = toggle LED D2 on "Tx Indicator"</p> <p>Bit 2: 0 = no GPIO access 1 = toggle LED D4 as "Alive Indicator"</p> <p>Firmware V1.6 extension Bit 3: 0 = no GPIO access 1 = toggle LED D1 as "Button Pressed Indicator"</p>
20	1	Misc. Options	<p>Bit 0: 0 = standard RF packet output format 1 = extended RF packet output format: attached RSSI, SNR and Timestamp</p> <p>Bit 1: 0 = RTC disabled 1 = RTC enabled</p> <p>Firmware V1.6 extensions: Bit 2: 0 = HCI Tx Indication disabled 1 = HCI Tx Indication enabled</p> <p>Bit 3: 0 = HCI Power-Up Indication disabled 1 = HCI Power-Up Indication enabled</p> <p>Bit 4: 0 = HCI Button Pressed Indicator disabled 1 = HCI Button Pressed Indicator enabled</p> <p>Firmware V1.10 extension Bit 5: 0 = AES Encryption/Decryption off 1 = AES Encryption/Decryption on</p>
21	1	FSK Datarate	<p>0 = 50000 bps 1 = 100000 bps 2 = 250000 bps</p>
22	1	Firmware V1.9 ext. Power Saving Mode	<p>0 = off 1 = Automatic Power Saving (RTC off recommended!)</p>

23	2	Firmware V1.10 ext. LBT Threshold	16 Bit signed Integer (LSB first) -120 to 0 [dBm]
----	---	---	--

3.1.5.4 Reset Radio Configuration

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

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_RESET_RADIO_CONFIG_REQ	Reset Radio Configuration
Length	0	no payload

Response Message

This message acknowledges the Reset Radio Configuration Request message.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_RESET_RADIO_CONFIG_RSP	Reset Radio Config Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix

3.1.5.5 Default Configuration

The following table lists the default configuration.

Parameter	Value
Radio Mode	0 = Standard Mode
Group Address	0x10
Tx Group Address	0x10
Device Address	0x1234
Tx Device Address	0xFFFF
Modulation	0 = LoRa
RF Carrier Frequency	869.525 MHz
Signal Bandwidth	0 = 125 kHz
Spreading Factor	7 = SF11
Error Coding	2 = 4/6
Power Level	17 = 17 dBm
Tx Control	0 = - Tx Narrow filter off - LBT off
Rx Control	1 = Rx always on
Rx-Window Time	3000 = 3000ms
LED Control	7 = Alive Indicator + Rx Indicator + Tx Indicator
Misc. Options	0x03: <ul style="list-style-type: none"> - extended RF packet output format enabled - RTC enabled - HCI Tx Indication disabled (V1.6) - HCI Power-Up Indication disabled (V1.6) - HCI Button Pressed Indication disabled (V1.6) - AES Encryption/Decryption off (V1.10)
FSK Datarate	0 = 50000 bps
Automatic Power Saving	0 =off (V1.9)
LBT Threshold	-90 [dBm] (V1.10)

3.1.6 System Status

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

3.1.6.1 Get System Status

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

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_SYSTEM_STATUS_REQ	Get System 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_SYSTEM_STATUS_RSP	Get System Status Response
Length	1+39	1+39 octets
Payload[0]	Status Byte	see appendix
Payload[1..39]	System Status Field	see below

3.1.6.2 System Status Field

The System 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	RTC Time	RTC Time (see RTC Time Format)
9	2	NVM State	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	Supply Voltage	Measured Supply Voltage in mV
13	2	Extra Status	Reserved Bit Field

15	4	Rx Packets	Number of received radio packets with CRC OK
19	4	Rx Address Match	Number of received radio packets with CRC and Address OK
23	4	Rx CRC Error	Number of received radio packets with CRC Error
27	4	Tx Packets	Number of transmitted radio packets
31	4	Tx Error	Number of not transmitted radio packets
35	4	Tx Media Busy Events	Number of not transmitted packets due to LBT result "media busy"

3.1.7 Real Time Clock Support (RTC)

The radio module provides an embedded Real Time Clock which can be used to determine the module operating hours or to generate timestamps for every received radio link message.

3.1.7.1 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	1 + 4	1 + 4 octets
Payload[0]	Status Byte	see appendix
Payload[1..4]	32 Bit time	see RTC Time Format

3.1.7.2 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[1..4]	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

3.1.7.3 RTC Time Format

The RTC time is transmitted as a 32-Bit value. The byte order within the message payload is low byte first (Little-Endian):

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 code 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

3.1.8 Set Radio Mode

This message can be used to change the radio mode without changing other configuration parameters.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_RADIO_MODE_REQ	Set Radio Mode Req.
Length	1	1 octet
Payload[0]	Radio Mode	See Radio Configuration for details

Response Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_RADIO_MODE_RSP	Set Radio Mode Rsp.
Length	1	1 octet
Payload[0]	Status Byte	see appendix

3.1.9 Power Saving

The radio firmware provides a low power mode with minimum current consumption which is useful for battery powered devices. It's possible to wake-up the radio module from low power mode by means of some dummy characters over the serial interface. The radio module needs ~3ms (30 SLIP_END (0xC0) characters @115200bps) until the clock system is stable enough to decode the next valid character on the serial interface.

3.1.9.1 Enter Low Power Mode (obsolete in V1.9 and later)

This message is used enter the low power mode.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_ENTER_LPM_REQ	Enter Low Power Mode Req.
Length	1	1 octet
Payload[0]	Mode	Reserved, must be 0x00

Response Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_ENTER_LPM_RSP	Enter Low Power Mode Rsp.
Length	1	1 octet
Payload[0]	Status Byte	see appendix

3.1.9.2 Automatic Power Saving (V1.9 and later)

The Automatic Power Saving (APS) can be enabled / disabled via Radio Configuration. If APS is enabled, the module will enter a low power mode whenever it is possible, e.g. after a packet transmission or expiration of an optional configured Rx Window.

Notes:

- #1 the configuration "Rx Always On" will prevent the module to enter the low power mode
- #2 Firmware V1.9 requires an enabled RTC!
- #3 Firmware V1.10 provides a lower current consumption if RTC is disabled on user level.
(The RTC will still be used internally if needed, e.g. for Rx Window timing etc..)

3.1.10 System Operation Modes

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.

3.1.10.1 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 Req.
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 Rsp.
Length	1 + 1	2 octets
Payload[0]	Status Byte	see appendix
Payload[1]	Current System Operation Mode	see below

3.1.10.2 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 Req.
Length	1	1 octet
Payload[0]	Next Operation Mode	see below

Response Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_OPMODE_RSP	Set Operation Mode Rsp.
Length	1	1 octet
Payload[0]	Status Byte	see appendix

3.1.10.3 System Operation Modes

The following System Operation Modes are supported:

Value	Description
0	Standard Application Mode / Default Mode
1	Hardware Test Mode for special test purposes

3.1.11 Power-Up Indication

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

This message is available in firmware V1.6 and later.

Event Message

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

3.1.12 AES Key Configuration (V1.10 and later)

The radio firmware can perform an automatic radio packet encryption and decryption (see chap. 3.2 Radio Link Services for more details).

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

3.1.12.1 Set AES Key

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

Command Message

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

Response Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_AES_KEY_RSP	Set AES Key Rsp.
Length	1	1 octets
Payload[0]	Status Byte	see appendix

3.1.12.2 Get AES Key

This message is used to read the configured AES key.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_AES_KEY_REQ	Get AES Key Req.
Length	0	no payload

Response Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_AES_KEY_RSP	Get AES Key Rsp.
Length	1 + 16	17 octets
Payload[0]	Status Byte	see appendix
Payload[1..16]	128 bit AES Key	octet sequence

3.2 Radio Link Services

The Radio Link Service Access Point provides functions for transmission and reception of radio link messages. The radio firmware part can operate in three different operation modes which are configurable (see Radio Configuration):

- Standard Mode: support for unreliable and confirmed radio message exchange with address filtering, listen before talk and packet encryption/decryption
- Sniffer Mode: forwarding of received radio messages via host interface without address filtering
- Echo Mode: simple retransmission of received radio messages (just for test purposes)

3.2.1 Unreliable Data Exchange

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

3.2.1.1 Send Unreliable Message

This command can be used to send a radio message either as broadcast message to all other radios in range or to a certain radio device with given address. Depending on the chosen radio settings, the transmission of a single radio message can take several hundred milliseconds. Firmware release V1.6 (and later) supports an HCI Tx Indication message which is sent to the host controller when the radio transmission has finished.

Command Message

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

Response Message

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

Event Message

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

3.2.1.2 Tx Radio Message Field

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

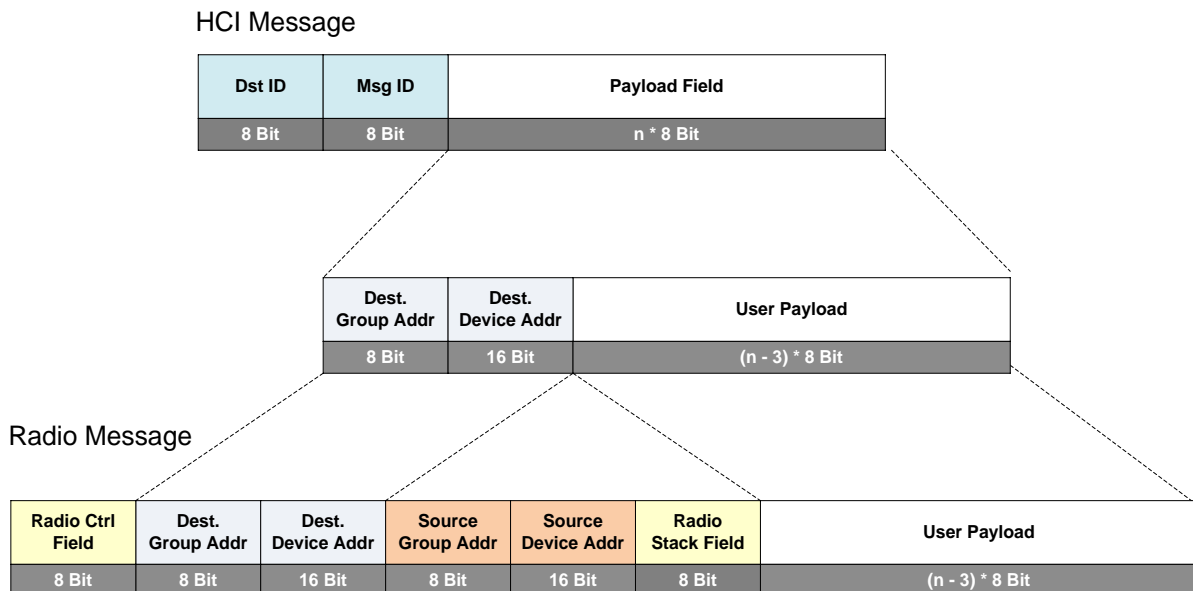


Fig. 3-3: Tx Radio Message and HCI Message(not encrypted format)

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

The HCI Payload field content is defined as follows:

Offset	Size	Name	Description
0	1	Dest. Group Address	Destination Group Address (0xFF = BROADCAST) of message receiver
1	2	Dest. Device Address	Destination Device Address (0xFFFF = BROADCAST) of message receiver

3	N	User Payload	<p>N bytes user defined payload with $1 \leq N \leq N1$</p> <p>$N1 = 255 - 8 = 247$ bytes (LoRa Mode, not encrypted data)</p> <p>$N1 = 255 - 12 = 243$ bytes (LoRa Mode, encrypted data)</p> <p>$N1 = 63 - 8 = 55$ bytes (FSK Mode, not encrypted data)</p> <p>$N1 = 63 - 12 = 51$ bytes (FSK Mode, encrypted data)</p>
---	---	--------------	--

3.2.1.3 Unreliable Radio Message Reception

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

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

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

Event Message

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

3.2.1.4 Rx Radio Message Field

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

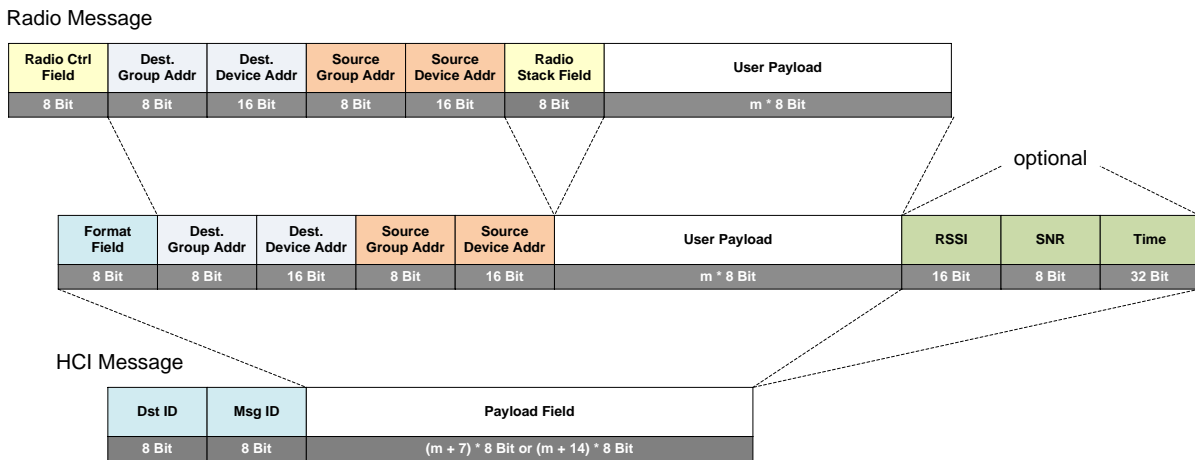


Fig. 3-4: Rx Radio Message and HCI Message(not encrypted format)

The HCI Payload Field has the following content:

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

3.2.1.5 Radio Control Field

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

Bit	Name	Description
0	ACK REQUEST BIT	"1" : Acknowledgement requested from peer device This bit is set to: "0" : in unconfirmed radio messages "1" : in confirmed radio messages (see below)
1	ACK BIT	"1" : Indicates an ACK message
2	CIPHER BIT	"1" : Indicates an encrypted radio message
3 - 7	reserved	

3.2.1.6 HCI Format & Status Field

The HCI Format & Status Field has the following meaning:

Bit	Name	Description
0	EXTENDED_OTUPUT	"0" : standard output format, no attachment "1" : extended output format with attached RSSI, SNR and RTC Timestamp
1 - 5	reserved	
6	DECRYPTION_ERROR	"1" : indicates a decryption error
7	ENCRYPTED_DATA	"1" : indicates encrypted data output

3.2.2 Confirmed Data Exchange

This service can be used to exchange radio messages in a more reliable way, i.e. a received radio message will be acknowledged automatically by the peer device. The following figure outlines the sequence of possible HCI messages.

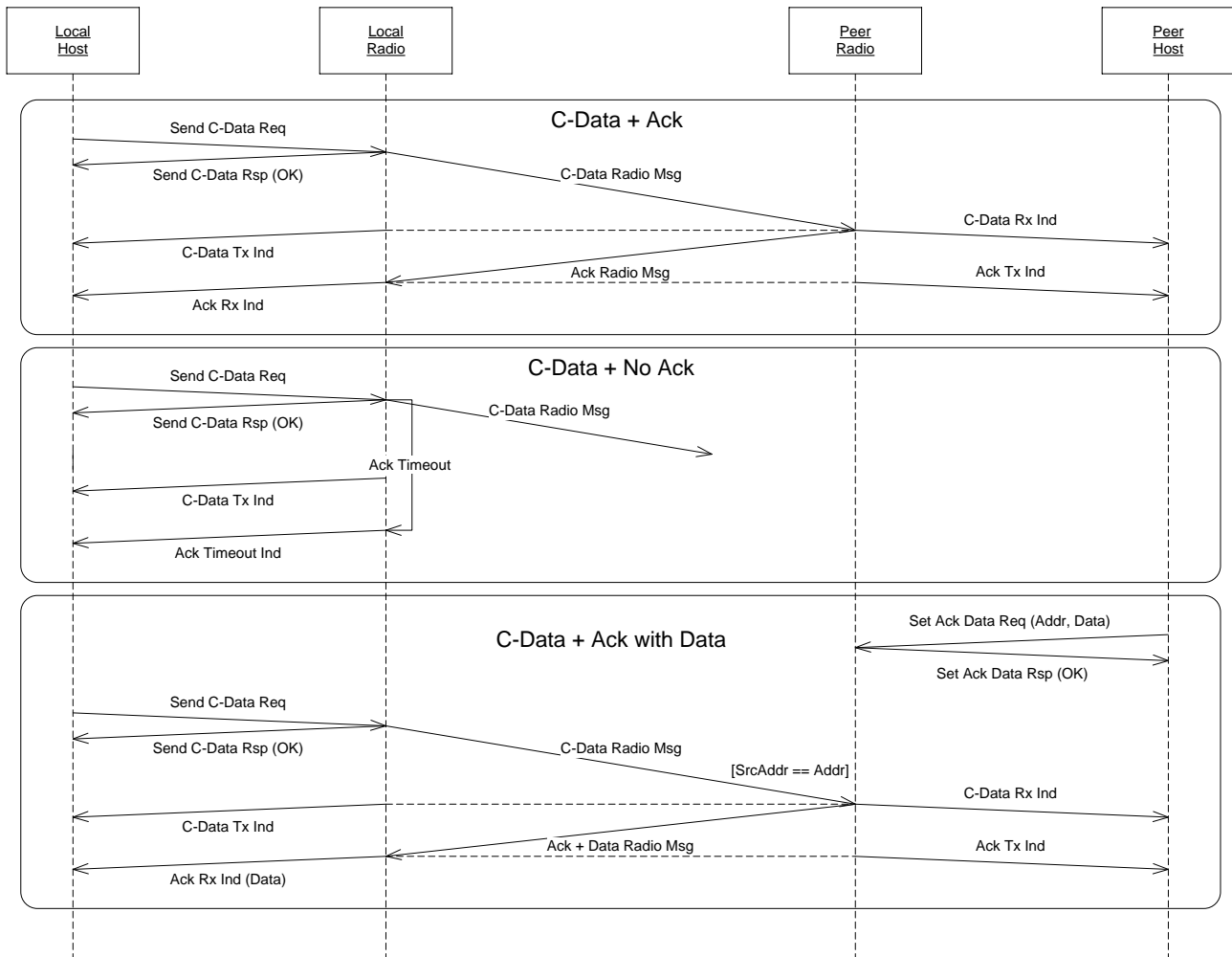


Fig. 3-5: Confirmed Data Exchange

3.2.2.1 Send Confirmed Message

This command can be used to send a radio message to a certain radio device with given address (group cast and broadcast are not supported!). Depending on the chosen radio settings, the transmission of a single radio message can take several hundred milliseconds. An optional HCI Tx Indication message is sent to the host controller when the radio transmission has finished. A further ACK Indication message is sent to the host after reception of an acknowledgement from the peer device. The ACK message may contain optional payload data from the peer side. If no acknowledgement has been received within a given time, an ACK Timeout indication is sent to the host.

Command Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_SEND_C_DATA_REQ	Send confirmed radio message request
Length	n	n octets
Payload	Tx Radio Message Field	see chapter Tx Radio Message Field

Response Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_SEND_C_DATA_RSP	Send confirmed radio message response
Length	1	1 octet
Payload[0]	Status Byte	see appendix

Event Message

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

Event Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_ACK_RX_IND	ACK radio message indication
Length	N	n octets
Payload	Rx Radio Message Field with optional payload from peer side	See chapter Rx Radio Message Field

Event Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier

Msg ID	RADIOLINK_MSG_ACK_TIMEOUT_IND	ACK Timeout indication
Length	0	No payload

3.2.2.2 Confirmed Radio Message Reception

The radio module is able to receive messages as long as the receiver is enabled. While operating in Standard Mode, the received messages are forwarded to the host controller when they contain the specific device address of the receiver.

Note: The receiver of a confirmed radio message will automatically send an acknowledgement to the initiator side. This ACK message can contain further user payload (see Set ACK Data).

Event Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_C_DATA_RX_IND	confirmed message indication
Length	N	n octets
Payload	Rx Radio Message Field	see chapter Rx Radio Message Field

The following message is sent to the host after transmission of an ACK message.

Event Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_ACK_TX_IND	ACK message transmission finished indication
Length	1	1 octet
Payload	Status Byte	see appendix

3.2.2.3 Set ACK Data

This message can be used to pre-set a limited number of additional payload octets for the next transmitted ACK message. The payload can be set for a limited number of device addresses which have to match to the source address of a confirmed radio message.

Command Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_SET_ACK_DATA_REQ	Set ACK Data request
Length	n	n octets
Payload[0]	Destination Group Address	
Payload[1..2]	Destination Device Address	
Payload[3..10]	Max. 8 Byte Ack Data	

Response Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_SET_ACK_DATA_RSP	Set ACK Data response
Length	1	1 octet
Payload[0]	Status Byte	see appendix

3.2.3 Sniffer Mode

The Sniffer Mode allows to monitor the radio link between other radio devices which use the same radio settings. The address filtering is disabled and every received packet is forwarded to the host controller using a raw radio packet output format. The Sniffer Mode can be enabled via Radio Configuration service.

Notes:

- #1 other services are disabled while the module operates in Sniffer Mode
- #2 radio packet decryption is not supported in Sniffer mode

Event Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_RAW_DATA_RX_IND	Raw radio message indication
Length	n	n octets
Payload	Rx Raw Radio Message Field	see below

3.2.3.1 Rx Raw Radio Message Field

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

Radio Message

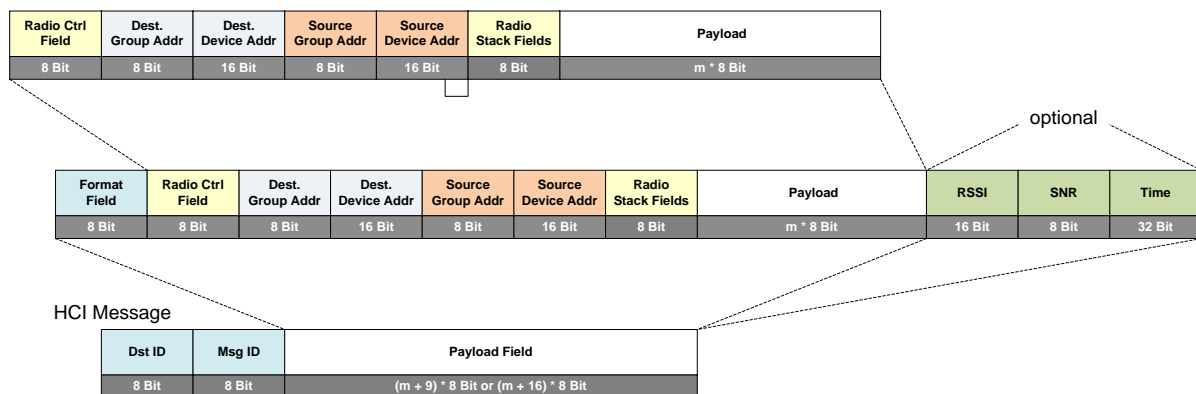


Fig. 3-6: Sniffer Raw Radio Message output format

The HCI Payload Field has the following content:

Offset	Size	Name	Description
0	1	Format & Status Field	Defines the packet output format (see chap. HCI Format & Status Field). Note: The ENCRYPTION_DATA bit is not maintained in Sniffer Mode
1	1	Radio Ctrl Field	see 3.2.1.5 Radio Control Field

2	1	Dest. Group Address	Destination Group Address (0xFF = BROADCAST) of message receiver
3	2	Dest. Device Address	Destination Device Address (0xFFFF = BROADCAST) of message receiver
5	1	Source Group Address	Group Address of message sender
6	2	Source Device Address	Device Address of message sender
8	1	Radio Stack Fields	Reserved for internal usage
9	N	Payload	N bytes message payload, can contain encrypted or not encrypted data (indicated by Radio Ctrl Field)
9+N	2	RSSI (optional)	Received Signal Strength Indicator [dBm]
11+N	1	SNR (optional)	Signal to Noise Ratio [dB]
12+N	4	Rx Time (optional)	Timestamp from RTC

3.2.4 Echo Mode

This feature can be used to implement a simple loopback test with one device under test (DUT) and a second reference radio device which is configured to run in Echo mode. While the reference module is operating in Echo mode, every received radio packet is simply transmitted again.

The Echo Mode can be enabled via Radio Configuration service. Note: other services are disabled while a module operates in Echo mode.

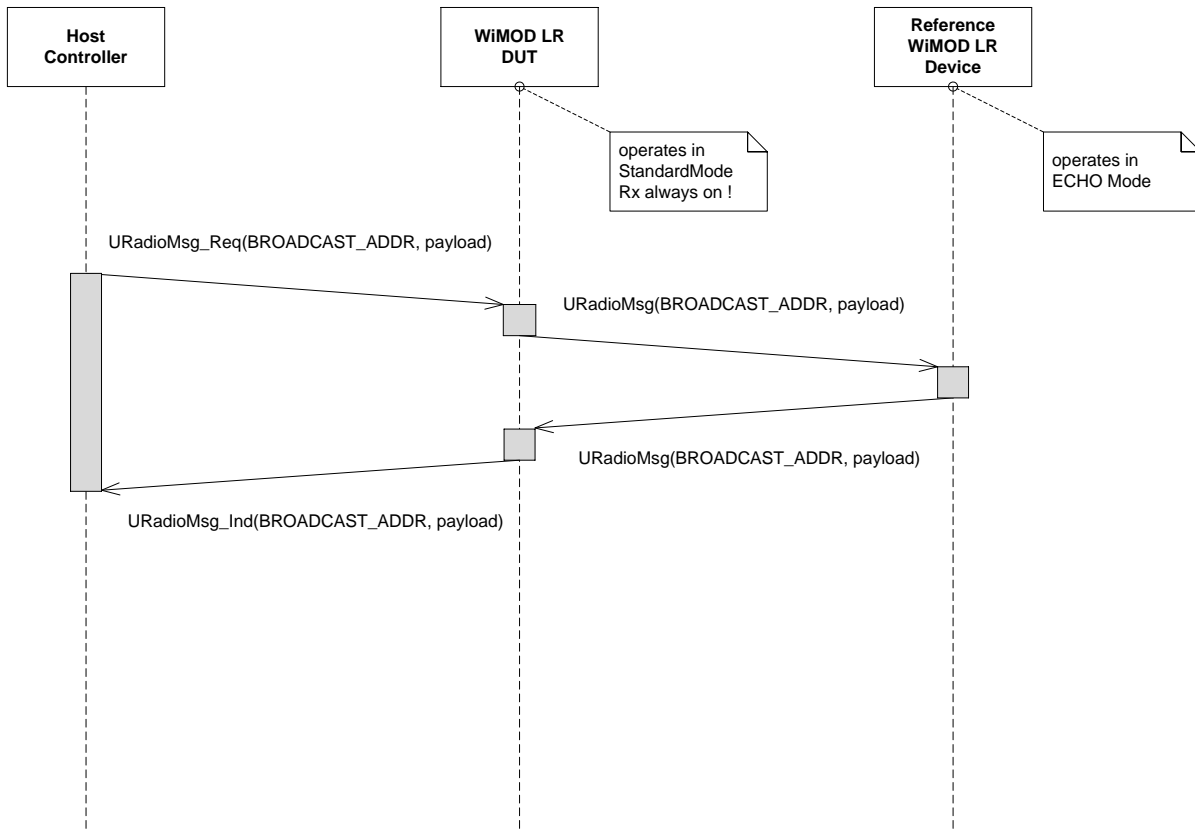


Fig. 3-7: Echo Mode for Loopback test

3.2.5 Radio Packet Encryption

The automatic radio packet encryption & decryption can be activated (see Radio Configuration) for every unconfirmed and confirmed radio message.

Note: ACK messages are not encrypted.

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

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

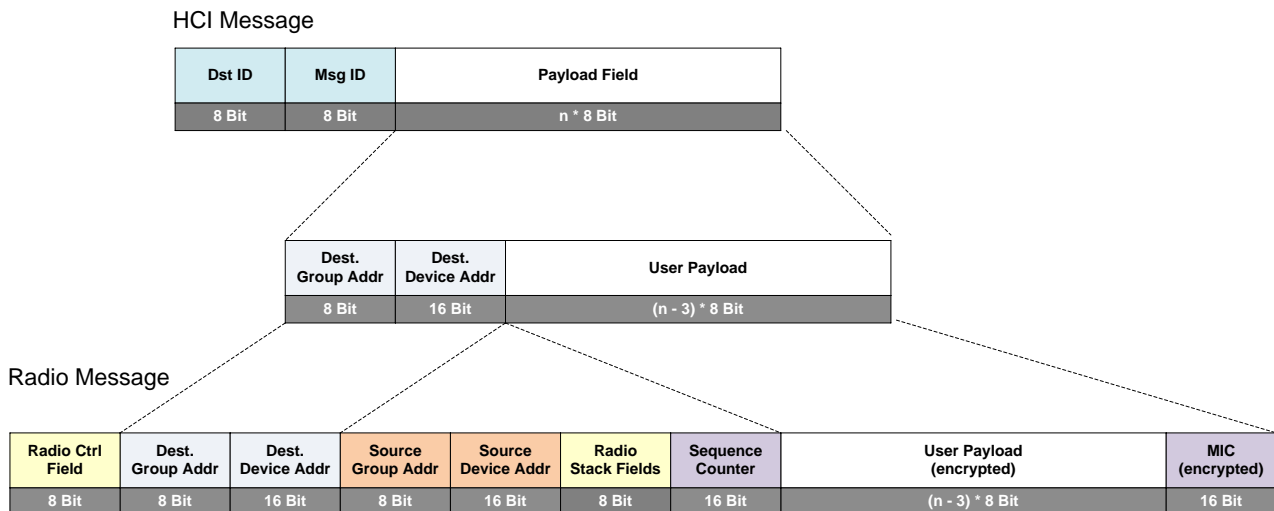


Fig. 3-8: Radio Packet Format for encrypted messages

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

- Sequence Counter
An automatic incrementing 16 Bit counter used as input for the AES 128 bit counter mode encryption
- MIC
A 16 bit message integrity code, used to verify a successful packet decryption on receiver side

Receiver Side:

On receiver side the following scenarios are possible:

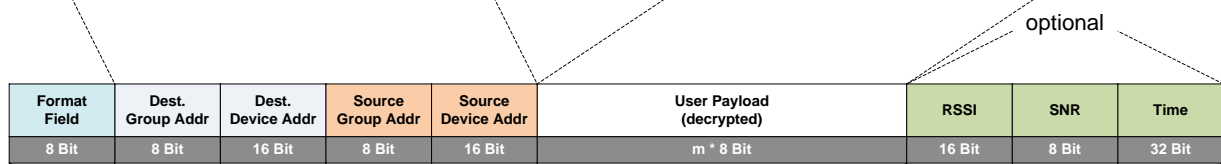
- Received message was successfully decrypted:
The forwarded HCI message uses the same output format as for not encrypted messages.
- Decryption on receiver side is disabled:
The forwarded HCI messages includes the sequence counter, encrypted user payload and attached MIC. The HCI Status & format Field indicates that the payload is encrypted.
- Decryption is enabled but a decryption error was detected (MIC error):
The forwarded HCI messages includes the sequence counter, encrypted user payload and attached MIC. The HCI Status & format Field indicates that the payload is encrypted and that a decryption error was detected.

The packet format for those three cases is outlined here:

Successful Decryption

Radio Message

Radio Ctrl Field	Dest. Group Addr	Dest. Device Addr	Source Group Addr	Source Device Addr	Radio Stack Fields	Sequence Counter	User Payload (encrypted)	MIC (encrypted)
8 Bit	8 Bit	16 Bit	8 Bit	16 Bit	8 Bit	16 Bit	m * 8 Bit	16 Bit



HCI Message

Dst ID	Msg ID	Payload Field
8 Bit	8 Bit	(m + 7) * 8 Bit or (m + 14) * 8 Bit

Fig. 3-9: : Rx Radio Message and HCI Message (encrypted radio data, decrypted HCI output)

The HCI Payload Field has the following content:

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

Not Decrypted Output (decryption error or decryption disabled)

Radio Message

Radio Ctrl Field	Dest. Group Addr	Dest. Device Addr	Source Group Addr	Source Device Addr	Radio Stack Fields	Sequence Counter	User Payload (encrypted)	MIC (encrypted)
8 Bit	8 Bit	16 Bit	8 Bit	16 Bit	8 Bit	16 Bit	m * 8 Bit	16 Bit

Format Field	Dest. Group Addr	Dest. Device Addr	Source Group Addr	Source Device Addr	Sequence Counter + Payload (encrypted) + MIC	RSSI	SNR	Time
8 Bit	8 Bit	16 Bit	8 Bit	16 Bit	(m + 4) * 8 Bit	16 Bit	8 Bit	32 Bit

optional

HCI Message

Dst ID	Msg ID	Payload Field
8 Bit	8 Bit	(m + 11) * 8 Bit or (m + 18) * 8 Bit

Fig. 3-10: : Rx Radio Message and HCI Message (encrypted radio data, not decrypted HCI output)

The HCI Payload Field has the following content:

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

3.2.6 Listen Before Talk

The Listen Before Talk option can be activated (see Radio Configuration) for every unconfirmed and confirmed radio message.

The LBT threshold level for the decision process is also configurable from the host controller.

In case of a "Media Busy" detection the radio module can send a HCI Tx Indication message with "Media Busy" status indication, if enabled. Thus it is recommended to enable the HCI Tx Indication (see Misc. Option Field of Radio Configuration) if LBT should be used.

3.3 Radio Link Test

The Radio Link Test feature can be used to analyze the radio link quality in a given environment. The test enables to measure the Packet Error Rate (PER) and RSSI level. The test can be started with several parameters by the Host Controller. The test operation is controlled by the local connected radio module itself. A second module in range is required, which must be configured with same radio settings.

Note: This feature is optional and not available in all firmware versions.

Message Flow

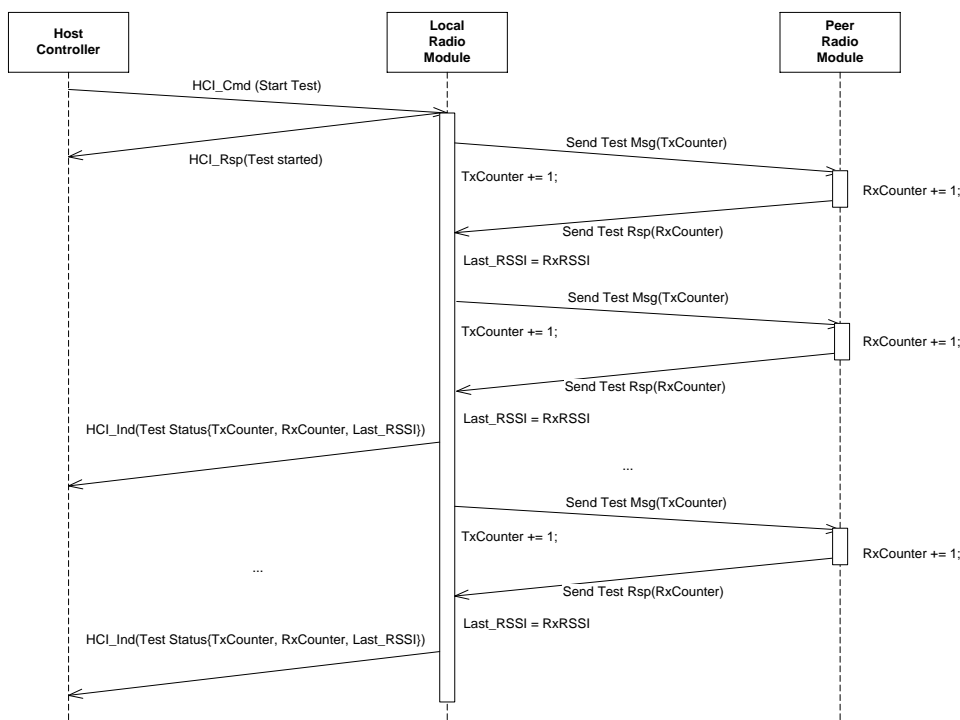


Fig. 3-11: Radio Link Test

During test operation the local connected module sends status messages to the Host Controller containing the packet counters and RSSI values of both devices. The status message includes the following quality values:

- LocalTxCounter - number of transmitted messages by local device
- LocalRxCounter - number of received messages by local device
- PeerTxCounter - number of transmitted messages by peer device
- PeerRxCounter - number of received messages by peer device

The Packet Error Rate(s) can be calculated by means of the following formula:

Downlink PER:

$$DL\ PER[\%] = (1 - PeerRxCounter / LocalTxCounter) * 100$$

Uplink PER:

$$UL\ PER[\%] = (1 - LocalRxCounter / PeerTxCounter) * 100$$

3.3.1 Start Radio Link Test

This message can be used to start a Radio Link Test session.

Command Message

Field	Content	Description
Endpoint ID	RLT_ID	Endpoint Identifier
Msg ID	RLT_MSG_START_REQ	Start Test Request
Length	N	n octets
Payload	Radio Link Test Parameter Field	see below

Response Message

Field	Content	Description
Endpoint ID	RLTT_ID	Endpoint Identifier
Msg ID	RLT_MSG_START_RSP	Start Test Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix

3.3.1.1 Radio Link Test Parameter Field

The following test parameter can be configured:

Offset	Size	Name	Description
0	1	Dest. Group Address	Destination Group Address
1	2	Dest. Device Address	Destination Device Address
3	1	Packet Size	Number of octets per test message
4	2	NumPackets	Number of test messages per test run
6	1	Test Mode	0x00 = Single test run 0x01 = Repeated test runs (Note : repeated test runs must be stopped by the host controller)

3.3.2 Radio Link Test Status Message

This message is sent from the radio module to the host controller during test operation.

Field	Content	Description
Endpoint ID	RLT_ID	Endpoint Identifier
Msg ID	RLT_MSG_STATUS_IND	Test Status Indication
Length	15	15 octets
Payload	Radio Link Test Status Field	see below

3.3.2.1 Radio Link Test Status Field

The Payload Field content looks as follows:

Offset	Size	Name	Description
0	1	Test Status	0x00 = OK 0x01 = start of new test run
1	2	Local Tx Counter	Number of transmitted packets from local device to peer device
3	2	Local Rx Counter	Number of received packets on local device
5	2	Peer Tx Counter	Number of transmitted packets from peer device to local device
7	2	Peer Rx Counter	Number of packets received on peer device
9	2	Local RSSI	RSSI value [dBm] of last received packet on local device
11	2	Peer RSSI	RSSI value [dBm] of last received packet on peer device
13	1	Local SNR	SNR value [dB] of last received packet on local device
14	1	Peer SNR	SNR value [dB] of last received value on peer device

3.3.3 Stop Radio Link Test

This message can be used to stop the Radio Link Test.

Command Message

Field	Content	Description
Endpoint ID	RLT_ID	Endpoint Identifier
Msg ID	RLT_MSG_STOP_REQ	Stop Test Request
Length	0	no payload

Response Message

Field	Content	Description
Endpoint ID	RLT_ID	Endpoint Identifier
Msg ID	RLT_MSG_STOP_RSP	Stop Test Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix

3.4 Hardware Tests

The firmware provides services for hardware test purposes. The test functions are mapped to an endpoint which is only accessible when the module operates in “Test Mode” (see System Operation Modes).

Note: This feature is optional and not available in all firmware versions.

3.4.1 Radio Tests

This message can be used to enable tests which are related to the transceiver section.

Command Message

Field	Content	Description
Endpoint ID	HWTEST_ID	Endpoint Identifier
Msg ID	HWTEST_MSG_RADIO_TEST_REQ	Radio Test Request
Length	n	n octets
Payload	Radio Test Parameter Field	see below

Response Message

Field	Content	Description
Endpoint ID	HWTEST_ID	Endpoint Identifier
Msg ID	HWTEST_MSG_RADIO_TEST_RSP	Radio Test Response
Length	1	1 octet
Payload[0]	Status Byte	see appendix

3.4.1.1 Radio Test Parameter Field

The following parameters are included in the Radio Test Parameter Field.

Offset	Size	Name	Description
0	1	Test Mode	Radio Test Mode: 0x00 = Test Off, all other parameters are ignored 0x01 = Continuous Wave Test (CW)
1	1	Power Level	see Radio Configuration
2	1	Modulation	see Radio Configuration
3	1	RF Carrier Frequency Least Significant Bts	see Radio Configuration
4	1	RF Carrier Frequency Intermediate Bits	see Radio Configuration
5	1	RF Carrier Frequency Most Significant Bits	see Radio Configuration
6	1	Signal Bandwidth	see Radio Configuration
7	1	Spreading Factor	see Radio Configuration

8	1	Error Coding	see Radio Configuration
---	---	--------------	-------------------------

3.5 Remote Control

Firmware V1.6 (and later) supports a simple remote control feature. Pressing of Button B1 on the WiMOD Demo Board will initiate an RF packet transmission. The radio packet will be sent to the configured Tx Group Address / Tx Device Address (see Radio Settings).

On receiver side two kinds of event indicators can be configured (see Radio Settings):

- LED D1 indicator – LED D1 toggles on every received Button pressed indication message
- HCI Button Pressed Indicator – a HCI message is transmitted to a connected host

3.5.1 Remote Control Button Pressed Indication

This message is sent to a host controller when the corresponding RF message has been received.

Field	Content	Description
Endpoint ID	REMOTE_CTRL_ID	Endpoint Identifier
Msg ID	REMOTE_CTRL_MSG_BUTTON_PRESSED_IND	Button Pressed Indication
Length	7	7 octets
Payload[0]	Dst. Group Address	Destination RF Group Address
Payload[1..2]	Dst. Device Address	Destination RF Device Address
Payload[3]	Src. Group Address	RF Group Address of initiator
Payload[4..5]	Src. Device Address	RF Device Address of initiator
Payload[6]	Button Bitmap: 0x01 = Button B1	Button Bitmap

4. Appendix

4.1 List of Constants

4.1.1 List of Endpoint Identifier

Name	Value	Comment
DEVMGMT_ID	0x01	Device Management
RLT_ID	0x02	Radio Link Test
RADIOLINK_ID	0x03	Radio Link Services
REMOTE_CTRL_ID	0x04	Remote Control
HWTEST_ID	0xA1	Hardware Test

4.1.2 Device Management Identifier

4.1.2.1 Device Management 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_SET_RADIO_CONFIG_REQ	0x11
DEVMGMT_MSG_SET_RADIO_CONFIG_RSP	0x12
DEVMGMT_MSG_GET_RADIO_CONFIG_REQ	0x13
DEVMGMT_MSG_GET_RADIO_CONFIG_RSP	0x14
DEVMGMT_MSG_RESET_RADIO_CONFIG_REQ	0x15

DEVMGMT_MSG_RESET_RADIO_CONFIG_RSP	0x16
DEVMGMT_MSG_GET_SYSTEM_STATUS_REQ	0x17
DEVMGMT_MSG_GET_SYSTEM_STATUS_RSP	0x18
DEVMGMT_MSG_SET_RADIO_MODE_REQ	0x19
DEVMGMT_MSG_SET_RADIO_MODE_RSP	0x1A
DEVMGMT_MSG_ENTER_LPM_REQ	0x1B (obsolete in V1.9)
DEVMGMT_MSG_ENTER_LPM_RSP	0x1C (obsolete in V1.9)
DEVMGMT_MSG_POWER_UP_IND	0x20 (Firmware V1.6)
DEVMGMT_MSG_SET_AES_KEY_REQ	0x21 (Firmware V1.10)
DEVMGMT_MSG_SET_AES_KEY_RSP	0x22 (Firmware V1.10)
DEVMGMT_MSG_GET_AES_KEY_REQ	0x23 (Firmware V1.10)
DEVMGMT_MSG_GET_AES_KEY_RSP	0x24 (Firmware V1.10)

4.1.2.2 Device Management 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 (check system operation mode)
DEVMGMT_STATUS_WRONG_PARAMETER	0x03	HCI message contains wrong parameter

4.1.3 Radio Link Identifier

4.1.3.1 Radio Link Message Identifier

Name	Value
RADIOLINK_MSG_SEND_U_DATA_REQ	0x01
RADIOLINK_MSG_SEND_U_DATA_RSP	0x02
RADIOLINK_MSG_U_DATA_RX_IND	0x04
RADIOLINK_MSG_U_DATA_TX_IND	0x06 (Firmware V1.6)
RADIOLINK_MSG_RAW_DATA_RX_IND	0x08
RADIOLINK_MSG_SEND_C_DATA_REQ	0x09
RADIOLINK_MSG_SEND_C_DATA_RSP	0x0A
RADIOLINK_MSG_C_DATA_RX_IND	0x0C
RADIOLINK_MSG_C_DATA_TX_IND	0x0E
RADIOLINK_MSG_ACK_RX_IND	0x10
RADIOLINK_MSG_ACK_TIMEOUT_IND	0x12
RADIOLINK_MSG_ACK_TX_IND	0x14
RADIOLINK_MSG_SET_ACK_DATA_REQ	0x15
RADIOLINK_MSG_SET_ACK_DATA_RSP	0x16

4.1.3.2 Radio Link Status Byte

Name	Value	Description
RADIOLINK_STATUS_OK	0x00	Operation successful
RADIOLINK_STATUS_ERROR	0x01	Operation failed
RADIOLINK_STATUS_CMD_NOT_SUPPORTEDED	0x02	Command is not supported (check system operation mode)
RADIOLINK_STATUS_WRONG_PARAMETER	0x03	HCI message contains wrong parameter
RADIOLINK_STATUS_WRONG_RADIO_MODE	0x04	Module operates in wrong radio mode
RADIOLINK_STATUS_MEDIA_BUSY	0x05	Firmware V1.0 Transmission not possible due to LBT result: "Media Busy"
RADIOLINK_STATUS_BUFFER_FULL	0x07	No buffer for radio transmission available
RADIOLINK_STATUS_LENGTH_ERROR	0x08	Radio packet length invalid

4.1.4 Radio Link Test Identifier

4.1.4.1 Radio Link Test Message Identifier

Name	Value
RLT_MSG_START_REQ	0x01
RLT_MSG_START_RSP	0x02
RLT_MSG_STOP_REQ	0x03
RLT_MSG_STOP_RSP	0x04
RLT_MSG_STATUS_IND	0x06

4.1.4.2 Radio Link Status Byte

Name	Value	Description
RLT_STATUS_OK	0x00	Operation successful
RLT_STATUS_ERROR	0x01	Operation failed
RLT_STATUS_CMD_NOT_SUPPORTED	0x02	Command is not supported (check system operation mode)
RLT_STATUS_WRONG_PARAMETER	0x03	HCI message contains wrong parameter
RLT_STATUS_WRONG_RADIO_MODE	0x04	Module operates in wrong radio mode

4.1.5 Hardware Test Identifier

4.1.5.1 Hardware Test Message Identifier

Name	Value
HWTEST_MSG_RADIO_TEST_REQ	0x01
HWTEST_MSG_RADIO_TEST_RSP	0x02

4.1.5.2 Hardware Test Status Byte

Name	Value	Description
HWTEST_STATUS_OK	0x00	Operation successful
HWTEST_STATUS_ERROR	0x01	Operation failed
HWTEST_STATUS_CMD_NOT_SUPPORTED	0x02	Command is not supported (check system operation mode)
HWTEST_STATUS_WRONG_PARAMETER	0x03	HCI message contains wrong parameter

4.1.6 Remote Control Identifier

4.1.6.1 Remote Control Message Identifier

Name	Value
REMOTE_CTRL_MSG_BUTTON_PRESSED_IND	0x02

4.2 Example Code for Host Controller

4.2.1 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 too 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
//-----
    
```

4.2.2 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 available
    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
            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 reception
        SLIP.RxState = SLIPDEC_START_STATE;
        break;
}
break;

```



```

        default:
            break;
    }
}

```

```

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

```

4.2.3 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 algorithm
#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, 0xFBEB, 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  
//-----
```

4.3 List of Abbreviations

DUT	Device under Test
FW	Firmware
HCI	Host Controller Interface
HW	Hardware
LR	Long Range
LoRa	Long Range
LPM	Low Power Mode
RAM	Random Access Memory
RF	Radio Frequency
RSSI	Received Signal Strength Indicator
RTC	Real Time Clock
SW	Software
UART	Universal Asynchronous Receiver/Transmitter
WiMOD	Wireless Module by IMST

4.4 List of References

- [1] iM880A_AN012_RFSettings.pdf

4.5 List of Figures

Fig. 1-1: Host Controller Communication	5
Fig. 2-1: HCI Message Flow	6
Fig. 2-2: HCI Message Format	7
Fig. 2-3: Communication over UART	8
Fig. 3-1: Ping Request	10
Fig. 3-2: Reset Request	11
Fig. 3-3: Tx Radio Message and HCI Message(not encrypted format)	34
Fig. 3-4: Rx Radio Message and HCI Message(not encrypted format)	36
Fig. 3-5: Confirmed Data Exchange	38
Fig. 3-6: Sniffer Raw Radio Message output format	42
Fig. 3-7: Echo Mode for Loopback test	44
Fig. 3-8: Radio Packet Format for encrypted messages	45
Fig. 3-9: : Rx Radio Message and HCI Message (encrypted radio data, decrypted HCI output)	46
Fig. 3-10: : Rx Radio Message and HCI Message (encrypted radio data, not decrypted HCI output)	47
Fig. 3-11: Radio Link Test	49

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.

Copyright © 2011, IMST GmbH

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